International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Impact Factor (2012): 3.358

Developing an Multi agent Integrated Development Environment on Cloud Platform

Paritosh Kumawat[1], Vinit Agarwal[2]

[1]Gyan Vihar School of Engineering and Technology, Suresh Gyan Vihar University, Jagatpura, Jaipur, Rajasthan, India

[2]Suresh Gyan Vihar University, Jagatpura, Jaipur, Rajasthan, India

Abstract: ***The software field is quickly migrating over the desktop to the World Wide Web. The Web serves a interactive user interface that enables ubiquitous access, quick collaboration, integrating the other online functions, and saves installation and configuration setting on individual desktops. For developing different software's, the Web provides a shift away from developing hub, and has the guarantee of closer collaboration and advanced feedback via various innovations in Web-based IDE's. Migrating these IDEs over the Internet does not just include "porting" the desktop IDEs; a basic reassessment of the known IDE architecture is essential in order to understand the full extent that the mixture of modern IDEs and the Internet can provide. This paper discusses challenges in research and chances in this area, followed by a brief study of a web IDE implementation.***

Keywords: Cloud, Agent Development, Clone-able Agent, Middleware**.**

1. Introduction

Software is migrating from the desktop machines to the Web or the internet. Online services are quickly substituting the basic wrapped and downloadable software articles. They implement in the Cloud, and use the browser software as a common UI that allows ubiquitous access, instant alliance, integration with other online services, and avoids installation and configuration on desktops.

Web-Oriented Software Development: It is common that the various software development tools follow this fashion, providing a Web-based pool for software development, supported by cloud-based services and storage. Many other software tools, which include issue tracking, version management, and build farms for continuous integration, are already given as Web-based services. Based upon the latest developments in Ajax technologies, hugely improvised JavaScript engines, and the coming of HTML5, there has been a small but growing collection of browser-based code editors.

Well developed integrated development environments (IDEs) are still falling behind in this competition towards the Web. Modern, desktop-based IDEs come with a wide available range of software engineering tools, and provide a platform for writing, maintaining, testing, building, running, debugging, and deploying the software. They increase the productivity of the developer by including many various kinds of editorial services specific to the syntax and semantics of any required language. These services guide the developers in understanding and moving through the code, they point developers to inconsistent or incomplete or hazy areas of code, and they also assist with editing code by giving automatic indentation, bracket insertion, and content completion and much more interactive features. The integration of complete tool sets for development of the software and the development of code-specific editor services took a impressive effort for the instant generation of IDEs like Eclipse and Visual Studio. Moving the next generation of IDEs to the Web is not just a topic of porting or migrating desktop IDEs; a basic reviewing of the IDE architecture is needed in order to realize the full potential that the combination of improved IDEs and the Web can provide.

The Web as a Software Making Platform. Being a platform for software development, the Web provides with a compelling combination of challenges and opportunities.

On the one hand, it possesses a distributed nature: computational nodes (server's vs. web browsers) have distinctly varied computational capabilities; over the browser-side, only JavaScript is properly supported; resources are distributed across un-trusted networks, with computing resources that may vanish and show up randomly and communication that is many orders of magnitude slower than inter-process communication.

On the other side, the Web also gives a new frontier for software development. The interconnection of clients on the Web enables closer cooperation between developers on a project through joint workspaces providing real-time collaborative modification and coordination of work. The centralized configuration and compiling of the cloud makes sure that all developers on a project use the same environment, since there is no need to locally and individually install new versions of the IDE, compiler, or various testing tools. The combination with other features enables user-extensible platforms based on embedded DSLs. The extensively scalable resources of the cloud enable speculative verification, compilation, and testing.

2. IDEs in the Desktop Era

From five decades, the first IDE was introduced, focusing the language- BASIC [20]. The IDE was entirely based upon command, and so did not appear much like the basic menu-

driven, graphical IDE's running today. Still, it has integrated the source code modification, compilation, debugging, and execution in a way very according to today's IDE.

Over the past fifty years, desktop IDEs have improved considerably, and are now very popular in today's software engineering practices. They give various tools for working with a variety of languages, collaborating different facilities for the version management, issue management, and so much more. They scale to large projects, big teams, and can be used with a variety of programming languages and tools for software engineering.

3. IDE Components

Modern IDEs exclusively elevate developer productivity by giving a vibrant user interface and tool support specialized for software development. They provide basic means for developing software and language-specific facilities for working with a specific programming language.

General IDE facilities incorporate support for managing source files, searching through projects, finding and replacing text, and much more. They also include collaboration with systems for version control, build management, and issue tracking. The latter facilities can be reiterated independently for a specific language, and usually operate at the level of whole set of projects, not single files.

Language-specific facilities are editor services and tooling customized towards a specific language. Modern IDEs often support a few or more language-specific editor services for a language, which includes the basic syntax highlighting, code navigation, documentation popups, content completion, (realtime) type checking and compilation, code outline view, refactoring, code formatting and other forms of language-specific support.

Figure 1 below gives an idea about various editor services in a desktop-based IDE. By repeatedly parsing and analyzing the source code, these services give quick feedback during editing a program, for example by highlighting the possible mistakes or giving suggestions to complete an expression. Other different language-specific functionalities include merged tools such as compilers, interpreters, and debuggers.

In the core of the modern IDE is the plugin model. It provides a collective framework for extending the IDE with new services. The user can install and upgrade the plugins for his IDE installation. All plugins run inside the IDE process, and share access to the same resources, such as the workspace, the projects and the files on disk.

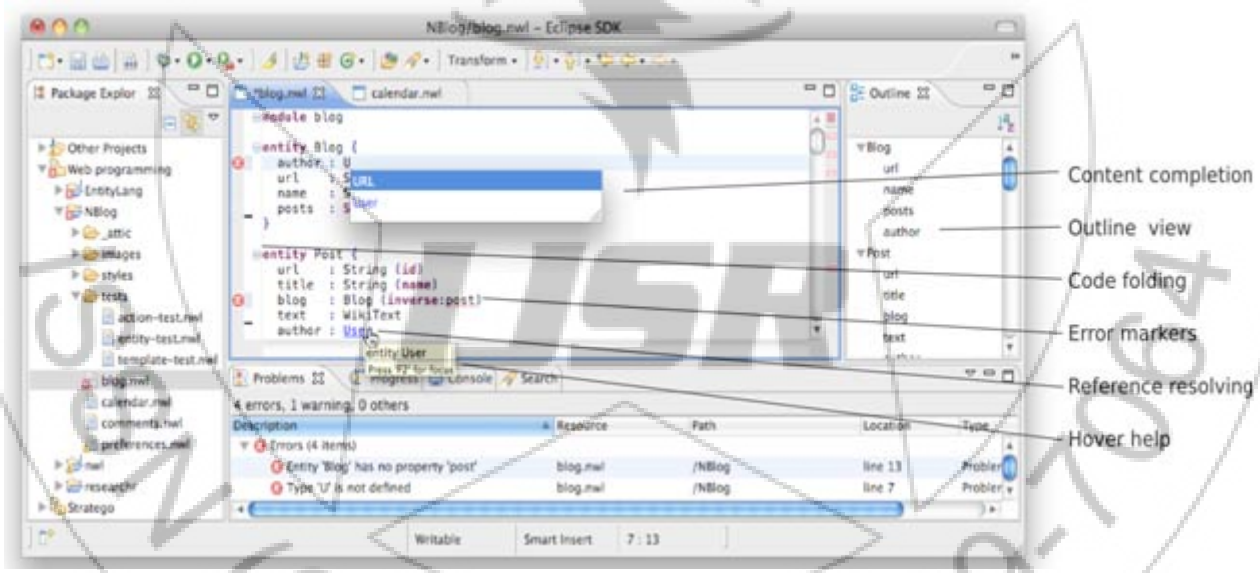

Figure 1: A Desktop IDE

4. Software Development in Context

Development of a Software and its maintenance is a highly associative effort. The important role of efficient and exact communication among the developers, developers and testers, and developers and end-users is well prevalent. It is also a well accepted reality that developers follow the path of low resistance. If the easily available tools make collaboration difficult, collaboration will happen less, or not at all.

Except many achievements and innovations of desktop IDEs, they still function within the limitations of the desktop paradigm: individual developers work on different machines, needing the installation, configuration, and maintenance of separate IDE instances for each developer. In a way, the desktop environment changes the developer workstation into a warehouse. Communication with the outside world works well for some aspects of development: software artifacts may be moved to outside machines for deployment, source code flows freely into and out of version control systems, issues (bugs, feature requests) are recorded into established issue trackers and day-to-day communication flows over instant messaging and/or e-mail.

These "sharing pipelines" have usually been set up before the project starts, and mixture is mostly limited to framework that fit into these "pipelines". A typical case where this breaks down is when developer A has encountered a hard-to-reproduce bug. Even if developer B wanted to help out,

recreating the exact state to trigger the problem on B's machine is usually so time-consuming as not to be worth the effort. It is usually better to join forces at the physical machine of developer A. None of the major IDEs provide real-time collaborative features to mitigate this problem—even though technology for doing so exists [5].

An analogy can be made in the case of authoring a document. Co-writing a document using a traditional desktop based word processor requires a substantial amount of machinery and ceremony. The co-authors must agree on a "protocol" for sharing documents among the participants, for example partitioning of the document and timely exchanges of the partitions by e-mail. A policy for conflict resolution must exist when multiple authors have edited their own copies of the same document and want to merge it. Contrast this with a co-writing documents in Google Docs. The real-time, online document collaboration offered by Google Docs requires no setup, no ceremony. Every participants sees the most up to date document at any time (modulo a few milliseconds to seconds, due to network latency).
Desktop word processors such as Libre Office are now acquiring similar collaborative editing features. This is not because collaborative editing was impossible before, because people write larger documents today, or because people did not collaborate in the past. It is more likely because real-time collaboration did not fit well in the silo-like mentality of the desktop paradigm, where every machine is an island.

5. IDE Deployment and Installation

The desktop paradigm dictates the local deployment, installation, and configuration of software on client machines. The time and effort required for setting up an IDE from scratch is not insignificant. We timed the setup process for an Eclipse installation with plugins for an issue tracker, a version control system, and a custom programming language to be around 18 minutes, start to finish.

The next stage is to set up the development workspace. In our experience, this is easily the most time-consuming part. For larger applications, especially Java web applications, it can take almost an hour to configure everything properly for a skilled developer, even when all necessary plugins are already present. This problem is exacerbated by Eclipse's relatively poor capabilities for sharing configurations between workspaces, and non-existent support for safely cloning workspaces.
Local deployment and installation imposes the burden of maintaining and upgrading the installation on the developer. While this allows the individual developer to manage the risk and time involved in upgrades, the recurring costs of upgrades are usually paid by all developers.

Resolving conflicting version requirements for plugins is a well-known headache for most IDE users, as is intermittent regressions due to accidentally incompatible plugin updates.

Once everything is set up, it might have to be redone, if the developer works on more than one machine, and especially if he works on more than one platform. Moving your development workspace from the Windows machine at work to the Linux machine at home requires installation of the entire setup from scratch.

6. Migrating to the Web

In the following sections we report on our current experience with proofs of concepts for realizing some of the fundamental services of a web IDE (code editing services, semantic analysis services, and execution services). Migrating from the desktop to the Web may be likened to solving a multivariable equation. The next sections outline what the known variables are, i.e. where we can reuse knowledge directly from the desktop paradigm, such as for parsing and type checking. For other variables, we suggest probable solutions based on analyses of the desktop solutions in the context of a Web architecture. For yet other variables, such as how to best design a distributed service model for a web IDE, we can only offer some fundamental research questions that might eventually lead to a solution.

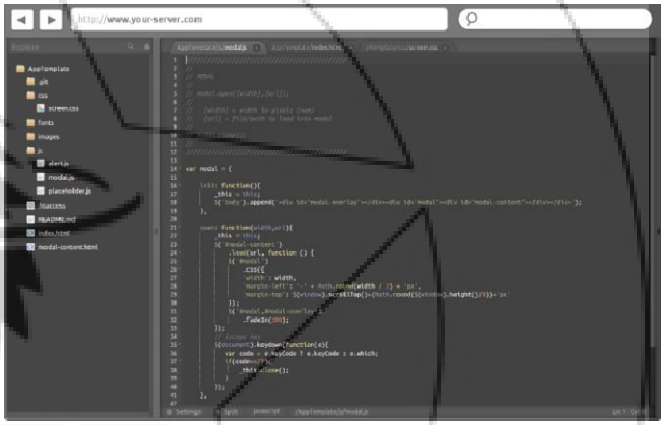

Figure 2: Server Migration

7. Web-Based Code Editors

Crucial technologies that enable the implementation of Web based code editors are (X)HTML, CSS, and JavaScript. These are available in any modern browser and provide a high degree of compositionality and adaptability for use within different Web pages. By contrast, browser plugins such as Flash and Java applets require an additional client side installation step, and may not be supported on all platforms such as portable devices. They also provide a much lower degree of compositionality and adaptability.

Figure 3: Proof-of-concept a web based editor

8. Conclusion

Agent based softwares are an popularizing and rapidly developing area under study. In the last ten years, a big number of essential developments happened in the design and development of software agent languages and over the implementation of the multiple agent systems. In this brief study area, we have concentrated our focus on a few key concepts, languages, tools and the different platforms and made a reference area to a large amount of body of literature. We intended to state and focus on the essentially required features of chosen languages, tools and platforms, instead of criticizing them. We consider an honest classification by gazing over the manner, agent programming languages are utilized during the process of their system development. On one side, we all can find agent.

languages very much useful for developing the agents that can be used as key building blocks for the deployment of complex shared applications, commonly based upon the agent or other appropriate middleware platforms. However, these languages are not useful immediately for developing real systems, but are rather mostly deployed for research work for understanding the complex systems using agent architecture and various simulation tools, as agent simulation languages. Note that this class of languages is very often commonly forgotten by the today's works that overview developments in agent programming.

9. Future Scope

The Technology of cloud computing is rapidly growing over time. By providing scalable server environment, everybody nowadays is trying to switch over cloud platform, when it comes to their server needs. The Integrated Development Environment offered on web, needs to be handled intelligently by a software program, in order to handle the incoming requests from the clients. Hence there has to be a middleware, or a broker kind of a program which handles the privileges offered to various subscribers or clients. There has to be an intelligent body acting between the clients and the cloud servers. There generates a need for Software Agents. Hence cloud offers a futuristic scope of nearly unlimited server space, Software Agents also have their future when it comes to handle the subscriber requests. These subscribers are very large in number, and open multiple windows for their running services, the Software Agents have to be multiple or Clone-able.

References

[1] Atlas. http://280atlas.com/.
[2] N. Ayewah and W. Pugh. The googlefindbugsfixit. In P. Tonella and A. Orso, editors, Nineteenth Int. Symposium on Software Testing and Analysis, ISSTA 2010, Trento, Italy, July 12-16, 2010, pages 241–252. ACM, 2010.
[3] K. Birman, G. Chockler, and R. van Renesse. Toward a cloud computing research agenda. News, 40(2):68–80, 2009.
[4] M. Bravenboer, K. T. Kalleberg, R. Vermaas, and E. Visser. Stratego/XT 0.17. A language and toolset for program transformation. Sci. of Comp. Programming, 72(1-2):52–70, June 2008. Special issue on experimental software and toolkits.
[5] L.-T. Cheng, C. R. B. de Souza, S. Hupfer, J. Patterson, and S. Ross. Building collaboration into IDEs. Queue, 1(9):40 – 50, Dec. 2003.
[6] Cloud9 IDE. http://www.cloud9ide.com/.
[7] CodeMirror. http://codemirror.net/, Apr. 2012.
[8] CodeStore Inc. Coderun. http://coderun.com, 2010.
[9] S. Efftinge and M. Völter. oAWxText - a framework for textual DSLs. In Modeling Symposium, Eclipse Summit, 2006.
[10] The google web toolkit documentation. http://code.google.com/webtoolkit/, Apr. 2012.
[11] N. Fraser. Differential synchronization. In U. M. Borghoff and B. Chidlovskii, editors, 2009 Symposium on Document Engineering, Munich, Germany, September 16-18, 2009, pages 13–20. ACM, 2009.
[12] C. Ghezzi and D. Mandrioli. Incremental parsing. ACM Transactions on Programming Languages and Systems (TOPLAS), 1(1):58–70, 1979.
[13] L. Grammel and M.-A. Storey. The smart internet. chapter A survey of mashup development environments, pages 137 – 151. Springer-Verlag, Berlin, Heidelberg, 2010.
[14] T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy. Predicting fault incidence using software change history. IEEE Trans. Softw. Eng., 26(7):653–661, July 2000.
[15] The google web toolkit documentation. http://code.google.com/webtoolkit/, Apr. 2012.
[16] J. Heering, P. R. H. Hendriks, P. Klint, and J. Rekers. The syntax definition formalism SDF: Reference manual. SIGPLAN Not., 24(11):43–75, 1989.
[17] Z. Hemel and E. Visser. Declaratively programming the mobile web with mobl. In K. Fisher and C. V. Lopes, editors, 2011 Int. conference on Object oriented programming systems languages and applications, OOPSLA 2011, pages 695– 712. ACM, 2011.

[18] S. I and K.-H. A. Research agenda in cloud technologies. http://arxiv.org/abs/1001.3259. LSCITS technical report, 2010.
[19] jsCoder IDE for the Apple iPhone. http://stuff. techwhack.com/9946-jscoder.
[20] jsFiddle – an online editor for the web. http://jsfiddle. net.

Author Profile

Paritosh Kumawat, born on August 23 1990, in Jaipur, Rajasthan; pursued B.Tech. in Information Technology from Suresh Gyan Vihar University, Jaipur, and currently pursuing M.Tech in Software Engineering from Suresh Gyan Vihar University, Jaipur, Rajasthan, India.

Vinit Agarwal is a Jaipur based Asst. Professor in Suresh Gyan Vihar University, Jaipur, in the Department of Computer Science.