ARCHITECTURAL DESIGN OF AN INTELLIGENT REQUIREMENTS ENGINEERING TOOL

Qian Zhang Armin Eberlein
Department of Electrical & Computer Engineering, University of Calgary
2500 University Dr. NW, Calgary, AB, T2N 1N4, Canada
Email: {zhangq, eberlein}@enel.ucalgary.ca

Abstract

Research in requirements engineering has resulted in various techniques, methods and frameworks but tool support is currently still very limited. It has been recognized that the lack of intelligent support tools for flexible, reliable, and adaptable requirements engineering processes is a major issue that prevents organizations from improving their requirements engineering practices.

In order to implement tools that offer intelligent support for requirements engineering processes, a web-based multi-tier software architecture is proposed in this paper. An intelligent support tool using this architecture can support scalable software projects and can be adapted to different requirements engineering processes.

Based on this architecture we have implemented a web-based tool for the "Requirements Acquisition and Specification for Telecommunication Services" (RATS) process. It can be configured to run on a standalone computer or in an enterprise environment.

Keywords: *Requirements engineering; Intelligent support; Architectural design.*

1. INTRODUCTION

For a long time, there has been a significant gap between academic research and industrial practice in requirements engineering. Even though researchers have spent tremendous effort to invent and introduce techniques, methods, processes, and frameworks for requirements engineering, software requirements are still poorly analyzed, documented, and managed in software industry [1][2]. The lack of suitable tools is one of the major obstacles to technology transfer from academia to software industry [2][3][4].

Requirements engineering tools are currently limited to requirements management tools and modeling tools. They do not yet offer intelligent support for flexible, reliable, and adaptable requirements engineering processes. Most of the requirements engineering tools from academia are not robust enough, not well documented, not scalable, nor integrated into the overall process [2]. In order to develop tools to support dynamic and scalable requirements engineering processes, the architectural design of the tools needs careful consideration.

In this paper, a web-based multi-tier software architecture is proposed to support the development of an intelligent requirements engineering tool for the RATS process [5][6]. This architecture facilitates enterprise software development and can be adapted to different requirements engineering processes. The requirements of the tool are discussed in section 2. In sections 3 and 4 the overall architecture and the detailed design are introduced. In section 5, we conclude our work and discuss possible future work.

2. REQUIREMENTS OF THE TOOL

In order to develop a tool that can offer intelligent support for requirements engineering, several issues need to be considered that are irrelevant for a general requirements management tool. These additional requirements put constraints on the architectural design and are discussed in the following sections.

2.1 Non-functional Requirements

In order to offer intelligent support for requirements development in a dynamic environment, the tool should be available, robust, understandable, scalable, integrated and expandable. All of these requirements are not only valid for any tool but especially for requirements engineering tools. Each of these requirements puts some constraints on the tool's architecture.

Available means the tool is immediately ready for use by requirements engineers in their work environments. The development environments of requirements engineers may not support them to run complex software. For example, when requirements engineers gather the requirements using the technique Designer-As-Apprentice

[7], they may not have the opportunity to install requirements engineering tools at the customer site. For this reason, it is necessary to have a lightweight installation such as a web browser on the user side but still offer enough functionality. *Robust* means the tool is running without major bugs, is easy to maintain and to upgrade. Daily maintenance might be necessary to achieve the required reliability. *Understandable* implies that the tool should have intuitive interfaces. To implement a tool with rich interfaces might require a separation of data processing and data presentation.

The number of requirements in industrial projects can reach several thousand. This requires tools to be *scalable*. A scalable tool must have the architecture to support customization to special environments without significantly decreasing performance. In order to collaborate with other requirements engineering tools to form a platform that supports the entire requirements engineering process, the tool needs to be *integrated*. This means the tool needs to be implemented using an open architecture with interfaces to talk to other tools. As requirements engineering is still immature, new technologies are emerging, requiring the tool to be *expandable*, i.e., the architecture needs to support the addition of new functionality without too much effort. An expandable tool can also assist requirements engineers to switch to a new process or handle fast changing environments.

2.2 Functional Requirements

Stakeholder involvement, incremental development, documentation support, and intelligent support are four important aspects that can be found in RATS and many other requirements engineering processes [3]. However, these aspects put special constraints on the architecture of requirements engineering tools.

To facilitate *stakeholder involvement*, the tool has to be easily accessibly to the tool users in their work environments. The tool also needs to offer up-to-date versions of the project requirements to all users and keep the data consistent and safe. *Incremental development* is widely used in current requirements engineering processes [8][9][10]. In order to support incremental development, tools need to trace the sources as well as changes of requirements. Requirements documents are the outcome of the requirements engineering process and the foundation of successful software development processes. Tools with the ability to support documentation also need to facilitate requirements tracing and the organization of requirements according to standards. This could help users to develop requirements with higher quality in shorter time. In order to offer intelligent support, the tool architecture must have modules that store knowledge about requirements engineering and efficiently execute checking rules according to a well-defined schedule.

3. ARCHITECURE OF THE RATS TOOL

Fig. 1 shows the architecture of the RATS tool based on the rationale discussed in the previous section. There are five parts in the architecture. *Representation* is concerned about the output of information and sends the input from tool users to application support. *Application support* dispatches the information to intelligent support and gathers the checking results from intelligent support. After receiving all the necessary checking results from intelligent support, application support will combine the information from the tool user with the checking results and puts them into the data storage. *Intelligent support* contains a set of executable checking rules that offer warnings and guidance to the tool user. *Data storage* keeps all persistent information in a database and offers unified interfaces for other parts of the tool to get up-to-date consistent information. *Management* allows the configuration of the tool to fit in a customer environment and controls the execution of the tool.

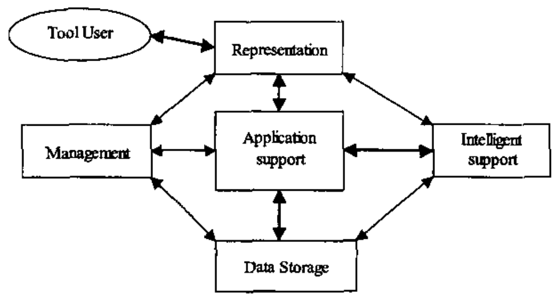

Fig. 1. Architecture of the tool [3]

This architecture separates the components that display, process, and store information. All the components can be installed on a single computer to offer a low cost solution or they can be installed into several servers connected with a network to offer better performance. Tool users with web browsers can access the tool from anywhere, which helps to achieve high availability and facilitates user involvement. The management component facilitates maintenance to offer better usability.

By separating and allocating different functionalities to different parts, this architecture helps locate and fix bugs easier. Separating the representation from the information and other functionalities can help the tool implementers to improve the user interfaces without affecting the processing of information. Data storage and intelligent support have open interfaces which can support information exchange with other parts of the tool or other requirements engineering tools to support tool integration.

As all the persistent information is stored in a central database, the tool can offer unified requirements to all users. With the help of application support and management, the intelligent support can be configured to load or unload dynamically. With this mechanism, it is easy to add or change functionalities without disturbing other parts of the tool.

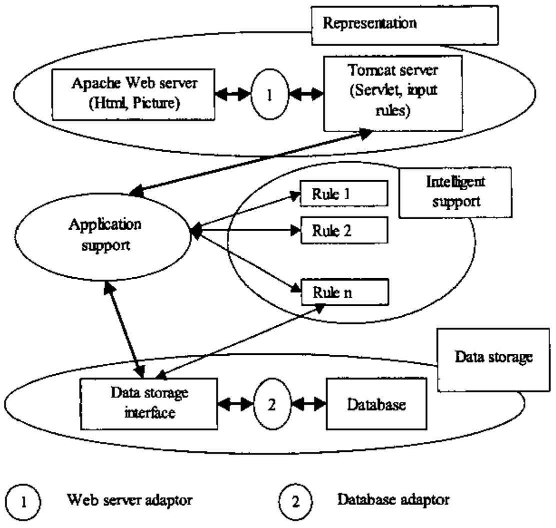

Fig. 2. Design of the RATS tool

Fig. 2 shows the current design of the RATS tool following the proposed architecture. The Apache server and Tomcat server together with the programs, web pages and pictures form the representation of the tool. The RATS tool receives the information from the user and interacts with application support to offer dynamic support. Input rules are also implemented in this part. The intelligent support consists of checking rules which represent human knowledge about the requirements engineering domain. These rules can be configured to immediately offer support (instant support) or with some delay (offline support). The data storage interface is an interface between the database and the rest of the support tool. It interprets the requests from the other parts of the tool and transforms them to a format which the database adaptor can understand.

4. DESIGN GUIDANCE

Although strongly related, each part of the architecture focuses on only a few aspects of intelligent support. The representation part is directly related to offering a user-friendly interface and is highly related to user involvement. The design of the data storage focuses on organizing the information to support incremental development. It is also related to offering the information in a unified manner to support different stakeholders that participate in the development. Application support focuses on offering a platform to schedule the execution of checking rules and support dynamic rule configuration. Intelligent support focuses on handling information efficiently. The design of the management part should carefully consider offering the tool administrator a convenient way for configuring and maintaining the tool. Data storage and intelligent support are the two parts that have the greatest effect on the performance of the tool.

Since requirements are created, refined, combined, and rejected during the requirements engineering process, the amount of information stored in the database increases rapidly. In order to retain the performance, the schema of the database needs to be carefully planned. Although there are many books [11][12] discussing how to design the schema of a database, there are some special issues that need to be considered for requirements engineering tools. Based on the size and changeability of the information the data storage should be optimized according to the suggestions in table 1.

Table 1. Database optimizing suggestions
(Volatility of information vs. database size)

	Volatile	**Stable**
Size is small	Optimize for create and query,	Optimize the storage for query
Size is large	Optimize the storage for create, size, and query	Optimize the storage for query and size

Intelligent support plays the central role in the RATS tool. It contains the checking rules that process the information coming from the user. The location of specific checking rules in the system has been optimized based on their computation time, the number of database accesses necessary and the functionality of the rules. The rules are categorized into five groups as shown in Fig 3.

Mandatory rules are working closely with the data storage. They are applied to the requirements to guarantee the consistency of the data and do some basic process checks required by other operations. If the mandatory rules are not satisfied, the data cannot be stored in the system. For example "*Each requirement has a unique ID*" is a mandatory rule that prevents the insertion of a requirement into the database if it does not have a unique ID number.

Local rules are working closely together with application support. These rules share only one database connection as well as the computational resources with application support. For this reason it is not appropriate to put a complex rule in this group that might require significant computing time or database connections. An example of this kind of rules is "*Each requirement needs to have its rationale specified.*"

Remote rules do not share their resources with application support. They can offer more delicate support than local rules without reducing the whole performance of the system too much. An example of these rules is "*An actor has to be involved in at least one use case*".

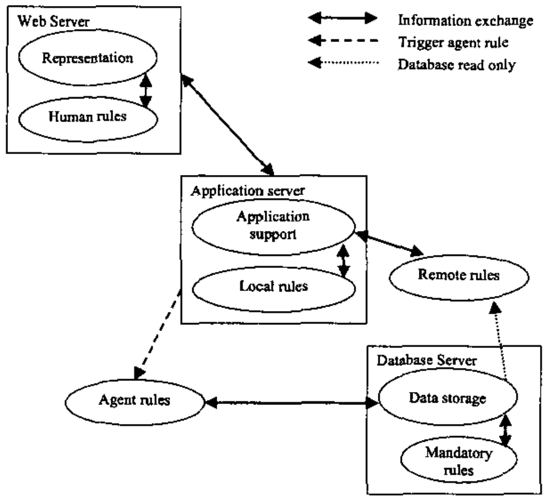

Fig. 3. Intelligent support framework

An agent rule is a rule which offers off-line support. These rules do not send their results to the application server instantly. They just send notices to the application server to indicate that they started to handle the information. After an agent rule gets the result, it will store the result directly in the database. As the agent rules run independently from the main threads, the tool-users cannot get instant help from these rules. The rules belonging to this group consume a lot of computational resources and ask for lots of database connections. An example of these rules is "*The requirements history links do not form a loop*".

Human rules are a mechanism that offers cooperation between different tool-users. They support the users to share their experiences and knowledge with each other. This mechanism gives the requirements engineer a way to handle vague, uncertain, or unclear issues. For example, a stakeholder may add warning messages to certain requirements which are created by requirements engineers, or a requirements engineer may add some guidance for the requirements created by stakeholders. This mechanism provides requirements engineering support based on human knowledge.

5. CONCLUSIONS

The proposed architecture can help tool implementers to develop scalable, integrated, expandable and intelligent tools that support requirements engineering with high availability, usability, and rich interfaces. It can also help to implement tools to offer better stakeholder involvement, incremental development, intelligent support, and documentation support.

An intelligent support tool using this architecture has the features to support enterprise software development and can be adapted to different requirements processes. Based on this architecture we have implemented a web-based tool for the RATS process. It can be configured running on a standalone computer or in a network environment. In order to implement more checking rules, the running schedule of the rules needs further analysis.

References

[1] The Standish Group, *Chaos Report.* West Yarmoth, MA: Standish Group International Inc., 1995.

[2] H. Kaindl, S. Brinkkemper, and J.A. Bubenko Jr, "Requirements engineering and technology transfer: obstacles incentives and improvement agenda," *Requirements Engineering*, vol. 7, no. 3, pp. 113-123, Sep. 2002.

[3] Q. Zhang, and A. Eberlein, "Deploying good practices in different requirements process models," *Proc. 6th International Conference on Software Engineering and Applications*, pp. 76-80, 2002.

[4] Q. Zhang, and A. Eberlein, "An architecture of an intelligent requirements engineering support tool," *Proc. 3rd ASERC Workshop on Quantitative and Software Engineering*, pp. 17-18, 2003.

[5] A. Eberlein, and F. Halsall, "Telecommunications service development: a design methodology and its intelligent support," *Journal of Engineering Applications of Artificial Intelligence*, vol. 10, no. 6, pp. 647-663, 1997.

[6] A. Eberlein, *Requirements Acquisition and Specification for Telecommunication Services PhD Thesis. Swansea.* UK: University of Wales, 1998.

[7] L.A. Macaulay, *Requirements Engineering.* London: Springer-Verlag Limited, 1966

[8] B.W. Boehm, "A spiral model of software development and enhancement," *IEEE Computer*, vol. 21, no. 5, pp. 61-72, May 1988.

[9] J.A. Goguen, and F.A.C. Pinheiro, "An object-oriented tool for tracing requirements," *IEEE Software*, vol. 13, no. 2, pp. 52 -64, Mar. 1996.

[10] K. Elmam, S. Quintin, and N.H. Madhavji, "User participation in the requirements engineering process: an empirical study," *Requirements Engineering Journal*, vol. 1, no. 1, pp. 4–26, 1996

[11] J. Teorey, *Database Modeling & Design 3rd Edition.* San Francisco, CA: Morgan Kaufmann, 1999

[12] J. Harrington, *Relational Database Design Clearly Explained.* New York, NY: Morgan Kaufmann Publishers, 2002