Eclipse as a Requirements Engineering Environment

Vincenzo Ambriola Luca Del Carlo Vincenzo Gervasi

Dipartimento di Informatica
Università di Pisa
via F. Buonarroti 2, I-56125 Pisa, Italy
{ambriola,delcarlo,gervasi}@di.unipi.it

Abstract

This paper introduces CPE, the CIRCE *Plugin for Eclipse. The CPE adds to the open-source development environment Eclipse the ability of writing and analysing software requirements written in natural language. Models of the software described by the requirements can be examined on-line during the requirements writing process. Initial UML models and skeleton Java code can be generated from the requirements, and imported into Eclipse for further editing and analysis.*

1. Introduction

The Eclipse platform [5] has been proven on the field to be general and scalable enough to handle most chores in industry-grade software development. Although primarily used for code development, Eclipse can be used, thanks to its many plugins [4, 6], to create application designs, interface with databases, design user interfaces, prepare and run tests, manage the development process, and for many other development-related activities.

It can be said that Eclipse has sucessfully consolidated and integrated in a single, productive environment most software development tasks, from early design to coding, testing and deployment. However, the initial steps of the software development process — those related to *requirements* — are not yet taken care of. We believe that the Eclipse platform can provide an extremely productive environment for requirements engineering (RE) activities such as:

- writing and managing requirements
- creating domain models
- visualizing models of the requirements
- validating the requirements for consistency and completeness
- generating initial design documents from the requirements
- generating skeleton code in Java from the requirements.

In the following, we describe how these and other functionalities have been integrated into Eclipse by interfacing it with the Requirements Engineering Environment CIRCE (see [3, 7, 9] for details and further references), developed at the University of Pisa. Section 2 introduces CIRCE, and establishes basic nomenclature that is used in the rest of the paper. Section 3 provides a short technical description of the *Circe Plugin for Eclipse* (CPE), the plugin that lets Eclipse access the features offered by CIRCE. Finally, Section 4 presents an example of how the CPE can be used in production environment, with a particular emphasis on the productivity gains that can be obtained. Some conlusions and references complete the paper.

2. The Circe RE environment

CIRCE is an environment for the analysis of natural language requirements. It is based on the concept of successive *transformations* that are applied to the requirements, in order to obtain concrete *views* of *models* extracted from the requirements. These include:

- models of the *requirements document* itself, considered as a textual artifact (for example: document structure or correlation between different requirements);
- models of the *software system* described by the requirements, and of the interactions with its *environment* (for example: data flow diagrams or UML sequence diagrams);
- models of the *requirements development process* (for example: change traces or Function Points score evolution in time).

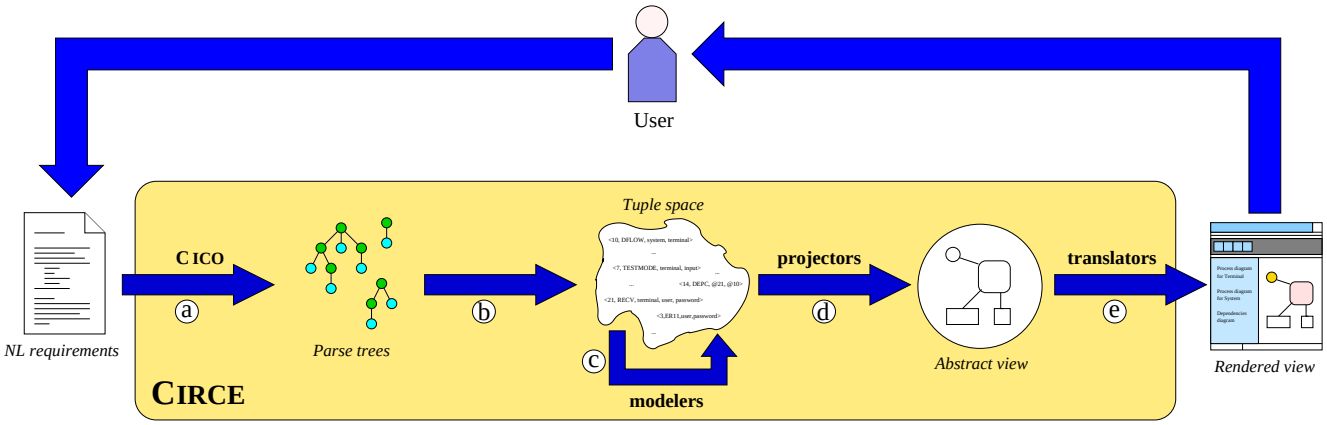

Figure 1. A general overview of the transformations on NL requirements operated by CIRCE.

The general architecture of CIRCE is shown in Figure 1. The whole transformation process (from NL requirements to concrete views of models) is divided into five steps, labeled ⓐ-ⓔ in the figure. The steps are briefly described in the following:

ⓐ First, the natural language text of the requirements is *parsed* and transformed into a forest of parse trees. These parse trees closely correspond to the original requirements, but abstract away many surface features, thus facilitating subsequent analysis.

ⓑ The parse trees are then encoded as tuples and immersed in a shared tuple space. This tuple space provides the extensional knowledge about the requirements, and serves as the basis for the next step.

ⓒ An embedded expert system is then called upon to enrich the tuple space with more refined information. The intensional knowledge about the basic structure and behaviour of software systems is provided by modular components called *modelers*. CIRCE includes a library of over one hundred modelers, covering such diverse aspects as static, functional and behavioural modeling, validation and metrication of such models, document structure analysis, synthesis of user interface components, etc.

ⓓ When a specific *view* on the requirements is desired, the needed information is extracted from the shared tuple space by a second class of components called *projectors*. This transformation produces an abstract (e.g., graph-theoretic) description of the desired view, starting from the extensional and intensional knowledge collected from the previous steps. This abstract view still exists only as an internal representation in CIRCE, and needs to be made concrete and offered to the user.

ⓔ This is the purpose of the last transformation. The abstract view is taken as input by other components called *translators*, and actual rendering is performed, thus producing a concrete view. Several different rendering modes may be offered for the same abstract view, to accommodate for different usage contexts. For example, a graph can be rendered either with interactive placement of nodes and zooming, to facilitate browsing, or as a static picture, for inclusion in paper documents.

From a user perspective, CIRCE is a system that reads natural language requirements as input, and — upon user's request — produces a vast number of views on different aspects of the requirements. The user can use the insights gained from the various views to correct, complete or perfect the original NL requirements. This induces a loop that includes the human as a driving force for the requirements development process, while offering him or her powerful tools to analyze the requirements in great detail.

CIRCE has been implemented as a web-based system. A central server acts as a repository for requirements documents. By using a standard web browser, users can edit their requirements, and ask for specific views. For each request, CIRCE performs the transformations[1] ⓐ-ⓔ needed to generate the requested view, and sends back the result as a web page (possibly enriched with active content: for example, by using Java applets for interactive graph browsing). This approach has the obvious advantage that users need no special software installed, and can work in a well-know environment (their preferred web browser). However, it also has some disadvantages:

- the quality of the interaction is rather low, being limited to basic text editing and web page browsing;

[1] Pervasive caching and differential parsing are used to improve performance.

- deep integration with other tools relies on manual techniques (copy&paste, download and import, etc.)

- users have to switch environment among different phases of the development process, introducing unneeded impedance in their workflows.

In the past we have used for teaching purposes CIRCE (for requirements analysis), IBM Rational Rose (for UML design), and IBM VisualAge for Java (for coding). Our experience is that these changes of environment have had an adverse impact on the productivity of the students.

By integrating CIRCE with Eclipse we obtain three main advantages: (i) a more powerful and comfortable environment for requirements writing and analysis, (ii) a smoother transition into design and coding, and (iii) the possibility of including requirements into any round-trip engineering needed during the development.

The integration is realized under the form of a new plugin for Eclipse, the Circe Plugin for Eclipse or CPE. This is the subject of the next section.

3. How the CPE works

From a user point of view, the CPE contributes several requirements-related items to the standard palette of Eclipse:

1. A new file type (extension `.circe`), for requirements resources. Actual documents are stored on the CIRCE server; local resources of this type store all the information needed for connecting to a given server.

2. A "New Requirements Document" wizard, to assist the user in the creation of a new requirements resource (see Figure 2).

3. A text editor for requirements document, providing three pages for Designations (defining domain-specific vocabulary), Definitions (defining domain-specific language constructs), and Requirements (describing the software system to be built) — see Section 4 for an example.

4. A menu, hierarchically listing the models, metrics, and validation views that CIRCE can generate. Once a view is requested, an in-memory resource is created with the view obtained by the CIRCE server, and the system-defined editor for HTML resources is invoked to present the view to the user.

5. A contributor to the "Problems" view, signalling to the user any violation of validity properties that CIRCE has found in the requirements.

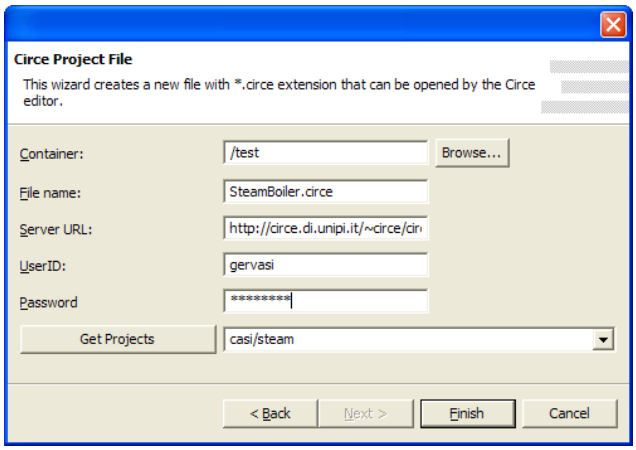

Figure 2. The "Create new requirements document" wizard.

Technically, the CPE is implemented as an Eclipse plugin, consisting mainly of (i) a new multipage editor, configured with custom syntax colouring, hover help and annotations; (ii) an editor contributor, dinamically updating the menu of available CIRCE views; (iii) a wizard to generate new `.circe` files; (iv) a background thread, periodically querying the CIRCE server for quality audits of the current requirements and displaying them in the "Problems" view; (v) import facilities, also accessible from the menu, to store CIRCE-generated UML models (as XMI files) and Java code into the local repository.

The CPE interacts closely with the CIRCE server through an HTTP-based custom protocol. To reduce network traffic, the CPE implements caching and pre-buffering wherever applicable. In particular, the CPE provides a buffered, network-based `IFileProvider` implementation that connects to the CIRCE server for loading and saving documents.

4. Writing requirements in Eclipse

Due to space considerations, we cannot present here a complete example of how the requirements for a real system are written in Eclipse. However, we can highlight some of the most relevant steps in the process. Examples will be taken from the Steam Boiler reference problem [1].

The first task consists in creating a new requirements document (i.e., a `.circe` resource) in a given project. This can be accomplished by invoking the "New Requirements Document" wizard (in Figure 2)[2].

Immediately after the creation, an editor is opened on the new document. The editor offers three pages, respec-

[2] A valid login and password for the user must be established beforehand on the CIRCE server

tively for editing designations, definitions, and requirements. Designations establish a domain dictionary, declare properties of the entities denoted by the various terms, and declare synonyms for the terms. For example, the declaration `steam boiler/ENTITY/IN/OUT` in Figure 5 indicates that "steam boiler" is a significant term in our domain, denoting an **entity** that can perform **in**put and **out**put w.r.t. its environment.

Definitions are used to extend the linguistic abilities of CIRCE. In most projects they are never used at all, for the basic language that can be recognized without definitions is normally sufficient for many systems. In the case of the steam boiler, we could state that "activating" some hardware component really means sending a start command to the component. This would be obtained by writing the definition `ACTIVATE x/IN` → `send start command to $x`, that establishes the meaning of "activating x" (provided that x is something that can accept commands) throughout the current requirements specification.

Finally, requirements describe, in natural language[3], how the software system to be built works and interacts with its environment. In this page both statements describing the environment (e.g., "The steam boiler is equipped with an evacuation valve") and those describing the actual working of the system (e.g., "Every 5 seconds, the controller reads the throughput of each working pump") can be entered. The editor assists the user in writing the requirements by highlighting words that are neither in CIRCE's basic dictionaries nor in the designations, as shown in Figure 3. A simple Content Assist option for enriching the designations with the new terms is also provided. Hover help is used to remind the user of the properties of a term occurring in the requirements. Deeper syntactic problems are identified asynchronously, by sending the text to the server every few seconds (behind the scenes) and using the results of the parsing performed by CIRCE to further annotate the document. Finally, those problems with the semantics of the requirements that can be detected by the expert system (step Ⓒ from Section 2) are identified when the document is saved, and used to populate both the document's annotation model and the "Problems" view of Eclipse. For each identified problem, all relevant details and suggestions on how the problem can be solved are presented to the user upon request (see, for example, Figure 4); moreover, for certain classes of problems a "QuickFix" — i.e., a wizard that automatically alters the requirements document to make it consistent — is provided, so that the most common solutions to typical problems are readily applicable.

Beside the lexical and syntactic support offered by the editor, and the validation checks shown through the "Problems" view, the user can apply the modeling facilities of CIRCE while working in Eclipse. For example, Figure 5

[3]English and Italian are currently supported.

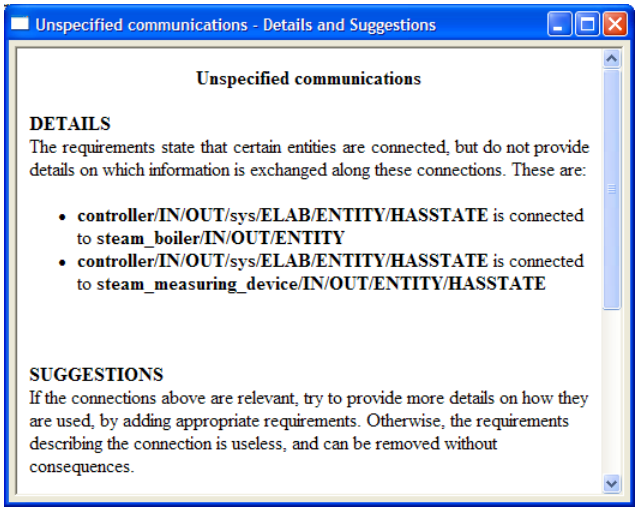

Figure 4. Details about one of the problems identified by CIRCE on the requirements shown in Figure 3.

shows, in the bottom right corner, a diagram of the communications path needed by the steam boiler, synthesized by CIRCE from the NL requirements given. A number of modeling views are offered, both on the requirements document (e.g., lexical homogeneity of the structure of the document) and, more importantly, on the system described by the requirements (including functional, static and dynamic behaviour). The interested reader can refer to [3] for details.

The capability of looking at a development artifact from different point of views can be invaluable, as witnessed by the fact that Eclipse natively offers multiple views, typically on source code. The CPE brings the same ability to requirements, offering over one hundred views wich the user can peruse at will during the development.

Also, at any stage of development the user can inspect or generate UML models [2] (to be imported into appropriate plugins of Eclipse like Omondo [12]), Abstract State Machines specifications [8], or skeleton Java code (to be imported into the Java Development Tools environment of Eclipse). For a restricted class of systems, the generated Java code provides a directly executable prototype for the system.

5. Conclusions

By interfacing Eclipse with CIRCE, the CPE adds powerful requirements analysis capabilities to Eclipse, while at the same time adding an extensible, comfortable, integrated development environment to CIRCE.

We believe that by consolidating under a single umbrella requirements writing and analysis, system design, and cod-

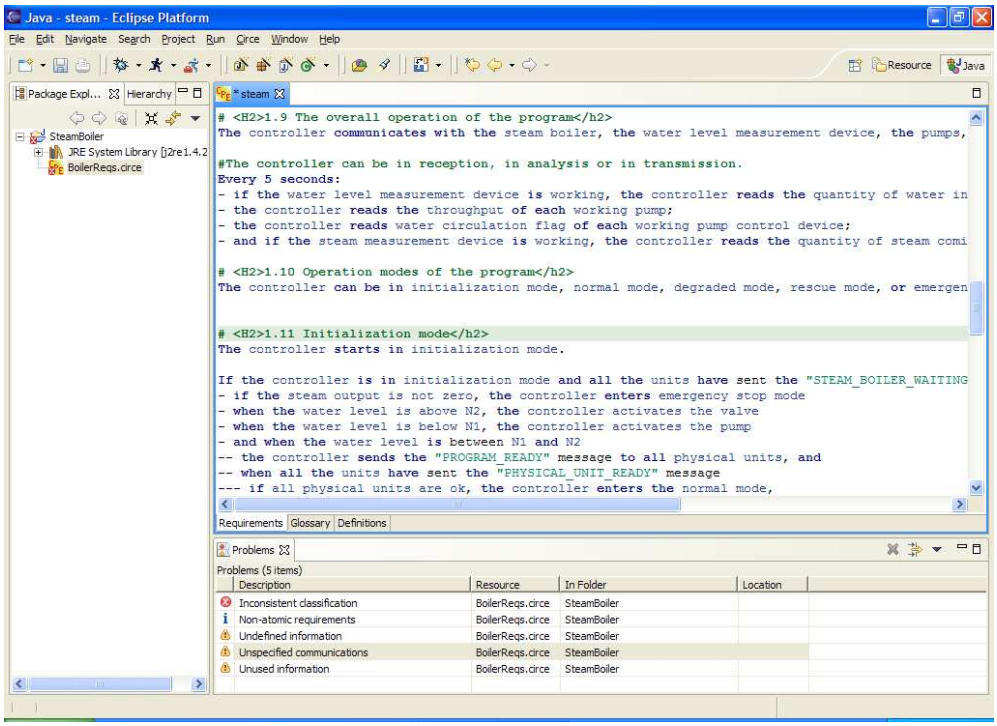

Figure 3. Editing of a requirements document. In the "Problems" view below the main editor, the list of problems identified in the current requirements.

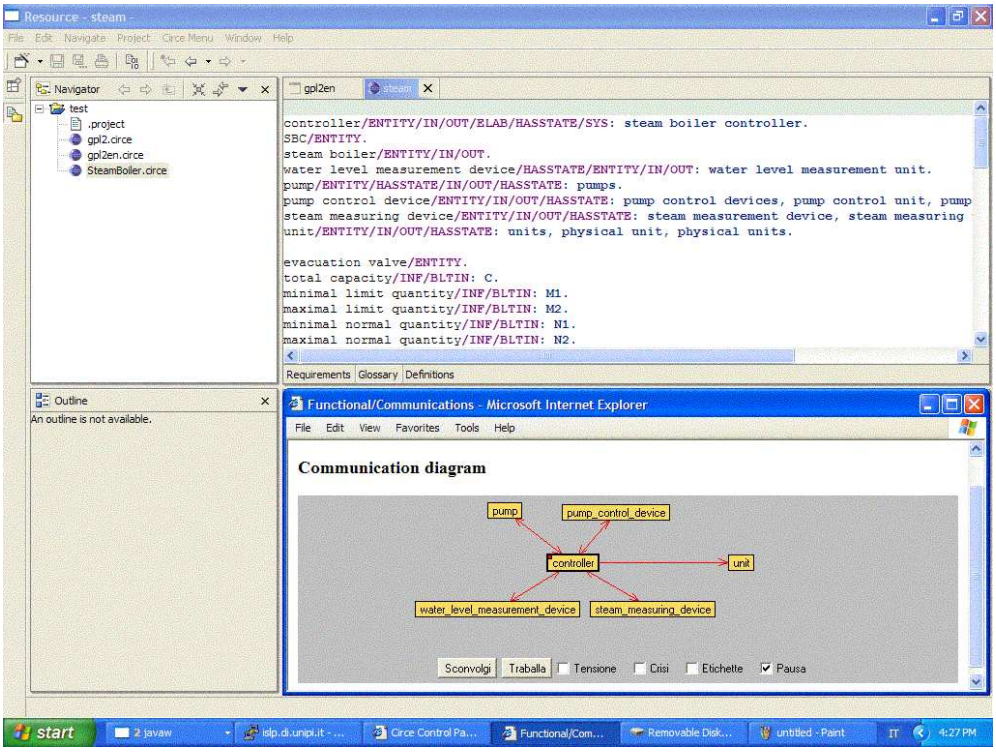

Figure 5. Eclipse showing the communication diagram generated by CIRCE from the NL requirements.

ing, great productivity gains can be obtained during the development process.

There are many aspects that have not been touched by this work. Among others, the issues of how requirements are *managed*, and how collaborative work on the requirements can be supported. Other projects being pursued by the Eclipse community have close ties with these issues: for example, collaborative work is the subject of the Koi subproject [11], while generation of Java code could be integrated in the Generative Model Transformer subproject [10]. We intend to investigate these issues as part of our future work in this area.

Acknowledgments. The authors wish to acknowledge the financial support of IBM through its IBM Eclipse Innovation Award programme. The authors would also like to thank IBM for its many contributions to the Open Source community at large.

References

[1] J.-R. Abrial, E. Börger, and H. Langmaack. *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control*. Number 1165 in Lecture Notes in Computer Science. Springer-Verlag, Oct. 1996.

[2] V. Ambriola and V. Gervasi. On the parallel refinement of NL requirements and UML diagrams. In *Proc. of the ETAPS 2001 Workshop on Trasformations in UML*, Genova, Italy, April 2001.

[3] V. Ambriola and V. Gervasi. The Circe approach to the systematic analysis of NL requirements. Technical Report TR-03-05, University of Pisa, Dipartimento di Informatica, Mar. 2003.

[4] K. Beck and E. Gamma. *Contributing to Eclipse*. (to be published).

[5] Eclipse.org home page. http://www.eclipse.org/.

[6] Eclipse plugins categorized list (eclipse-plugins.info). http://www.eclipse-plugins.info/eclipse/index.jsp.

[7] V. Gervasi. *Environment Support for Requirements Writing and Analysis*. PhD thesis, University of Pisa, March 2000.

[8] V. Gervasi. Synthesizing ASMs from natural language requirements. In *Proc. of the 8th EUROCAST Workshop on Abstract State Machines*, pages 212–215, February 2001.

[9] V. Gervasi and B. Nuseibeh. Lightweight validation of natural language requirements. *Software: Practice & Experience*, 32(2):113–133, Feb. 2002.

[10] Gmt home page. http://www.eclipse.org/gmt/.

[11] Koi home page. http://www.eclipse.org/koi/.

[12] Omondo home page. http://www.omondo.com/.