The Ophelia Traceability Layer

Mike Smith
Pauline Wilcox
Rick Dewar

Dept of Computer Science
School of Mathematical and Computer Sciences
Heriot-Watt University
UK

Dawid Weiss

Institute of Computing Science
Poznan University of Technology
Poland

Abstract

Assessing the impact of change on a software development project is a critical management activity. Traceability affords us opportunities to manage the change process through notification and synchronisation mechanisms. We present an architecture, developed as part of the EU funded Ophelia project, that supports traceability across all project artefacts.

1 Introduction

Traceability, in the context of software engineering, is defined as the ability to trace the life of an artefact from its inception to its use [11]. Here, artefacts could be requirements, code, models, reports and plans, etc, although, to date, much of the salient work in the literature has focused on requirements traceability, for example [4, 10]. [1] describes key benefits of requirements traceability from a software engineering point of view. It can help align changing customer needs with the software, reduce risk through the capture of critical knowledge, help determine the impact of a change to the system and help process improvement.

Moreover, determining the impact of changing requirements on a development project is critical to the management of that project [8] and is seen as a fundamental problem in software development [3].

Project requirements (functional, non-functional and implicit) may change for many reasons. In the context of software development we can identify both external and internal sources of change. External sources include factors such as changing functional requirements, business needs, bug reports, hardware environment, project budget etc. Internal change may be driven by needs of refactoring code base or design, internal bug discoveries, etc.

The two most common approaches to managing change include formal, paper driven change management processes and a family of so-called Agile processes [6]. The paper driven approach is usually based on an extensive set of rules, which members of the project must follow. In effect, every change to the project is well documented and traceable. Unfortunately, such an approach often leads to information overload and requires that a significant effort be put into managerial activities.

On the other side of the scale we have a family of light change management methodologies, commonly referred to as agile processes. Introduced for the first time in extreme programming, techniques such as advocate code sharing, rotation of developers, pair programming, self-explanatory code, frequent unit testing, integration and refactoring have proven to increase the efficiency of dealing with frequent changes just as well as their paper-driven counterparts. The net result of these techniques is that the development team has excellent, system wide knowledge about the project. Hence, incorporation of any change comes very natural and it is not required to meticulously administer it on paper. This approach is not without its shortcomings however, as it seems to be difficult to apply to large projects.

In addition to these process based approaches to change management there is some tool support for change activities, for example Rational XDE [13] supports synchronisation of a UML model with code. Any change to a UML model results in the automatic update of the code that is related to that model and vice versa.

2 Motivation

We are particularly interested in the notion of traceability as described in [11] and believe that, at a conceptual

level, such traceability affords us at least two opportunities to manage the change process. Firstly, human stakeholders and other artefacts in a project can be notified about changes to an artefact during its lifecycle; and secondly, inter-artefact notifications can be propagated and result in synchronisation. By synchronisation we mean the process through which we co-ordinate and implement the adjustment, reconciliation or bringing together of two artefacts that are related through traceability links but have somehow become inconsistent with each other. Inter-artefact change notification and synchronisation as described above are dependent on granularity and persistence of artefacts within an integrated development environment.

Our motivation to writing this paper was the fact that, as part of the Ophelia [9, 5, 14, 2] project, we have developed such an environment and believe that the traceability layer of the architecture is the first step in realising change notification and synchronisation. Thus, we want to introduce several ideas we employed in Ophelia and present them for public discussion.

The remaining part of this paper introduces the Ophelia project and describes how the traceability layer is realized.

3 The Ophelia Project

The Ophelia project is an EU funded initiative that aims to develop a platform to support software engineering in a distributed environment. Central to this aim is the integration of pluggable, heterogeneous software engineering tools.

There is extensive tool support for many modern software engineering processes. However, most of these tools use proprietary data formats and APIs, making it hard, if not impossible to integrate them into one software production environment. While there are some fully integrated solutions [12], these are usually very expensive, hardware-demanding and tightly coupled within a set of utilities delivered from one company. The Ophelia project aims to provide support for integrating heterogeneous tools by defining a set of standard interfaces for accessing their generic functionality. These interfaces are described in terms of CORBA IDL definitions [7] and are currently defined for such areas of project development as requirements, modelling, repository, bug tracking, project management, metrics and documentation management tools. Any particular piece of software falling into these categories can be wrapped to support the Module Interface Specification (MIS) and in this way become part of the integrated environment.

Ophelia also specifies several services utilizing the ability of accessing various elements of the project in a uniform manner (via MISs). Such services currently include knowledge management (cross-project documentation service) and our main interest - traceability. The Ophelia architecture is illustrated in Figure 3.

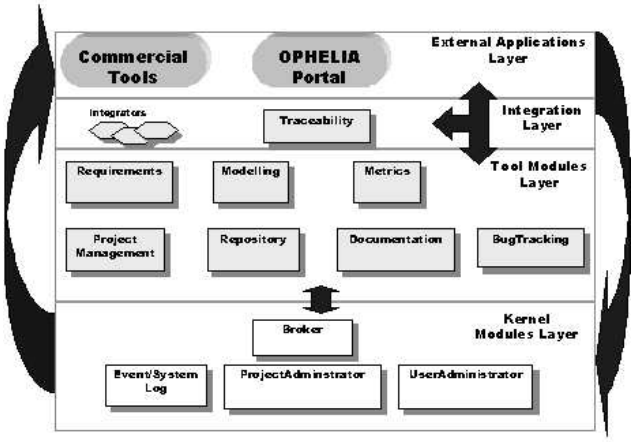

Figure 1. The Ophelia Architecture

The Kernel Layer is composed of modules that provide general services to tool modules and external applications. The broker is central to the architecture; individual modules discover the broker through the CORBA naming service and register with it. At a later date, references to these modules can be retrieved through the broker.

The Ophelia Tool Modules Layer comprises a series of interfaces that define the services offered by different genres of software engineering tools, for example modelling or metrics collation. CORBA [7] MIS define the objects and services that these modules offer. A concrete realisation of an MIS makes the services of one module transparently available for use by another, thus facilitating interoperability between tools.

The Integration Layer provides additional functionality to support integration of the Tool Modules. Integrator applications facilitate specific inter-module communication in addition to data transfer synchronisation and marshalling. Additionally, this layer provides traceability, in the form of event notification to users, and relationship management between external applications and modules. These first three layers form the Ophelia platform definition.

External Applications use the Kernel Modules to access the functionality provided by the Tool Modules. These applications can be commercial products or bespoke tools. A specific toolset, and corresponding Integrators, define a particular distribution - known as an Ophelia solution. The ability to select external applications enables users to establish customised Ophelia solutions that best fit their development environment

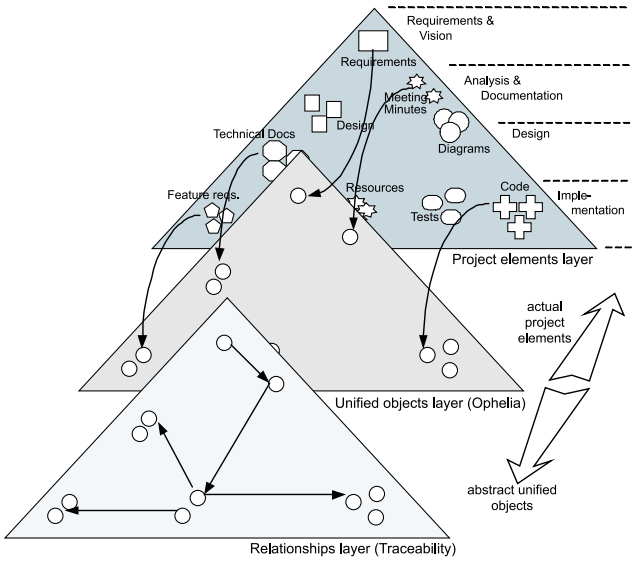

Figure 2. The Traceability Layer

4 The Ophelia Traceability Layer

The traceability module of the Ophelia project is founded on the fact that we represent artefacts of the software engineering process as CORBA objects.

By considering all the artefacts of the project we are able to capture all the relationships present and not just those derived from requirements, for example relationships between defects and code, defects and models, documentation and code, code and unit tests etc. As we represent the artefacts of the project as CORBA objects, the traceability layer can be seen as an essential part of project meta-data - see Figure 4. Meta-data is, in our opinion, an extension and generalisation of the concept of traceability. Project meta-data could be defined as those parts of the project, which are not explicitly elements of the software development, but form some additional, useful and potentially profitable information about this project.

It should be stated that project meta-data is not limited to relationships among its basic elements. It could also include information about versions of artefacts, about physical localisation of objects, etc.

Let us go back to the specific slice of the meta-data layer, notably the graph of relationships. There are several benefits of having such a graph. First of all, it can be used as an excellent starting point for browsing elements of the project. Navigating the structure of object inter-relations is much more intuitive than finding objects by specific type. For instance, starting with a requirement, one may easily follow all relationships that connect this requirement to the source code, documentation or even unit tests. This relationship-oriented navigation is a very tempting idea, especially for intranet, highly distributed environments, where elements of the project may reside on repositories in different countries.

Having an abstract uniform interface to all elements of the project and an explicit graph of relationships among these elements gives an excellent opportunity to implement automatic notifications about changes occurring on them. In Ophelia we are planning to implement a simple mechanism of notifications, where objects signal a change in their status and this change is propagated along relationships present in the traceability layer to other elements of the project. Users may sign up (or may be signed up by other users like project managers) to receive such notifications. In this way, no change on a depending object will go unnoticed, because some user must always accept it.

There is also a further perspective of automatic synchronization tools, taking benefit from the notifications sent from traceability layer. Let us imagine a system, where all the code is written in CWEB, a programming language written by Donald Knuth, where the code is accompanied by documentation already in source files. One could easily imagine an automatic synchronization tool, which in response to a change on a source code file, generates documentation (using TeX), generates code (using standard C compiler), deploys a test script and possibly even notifies person in charge of integration that a new build has been finished and tested.

Another key benefit of the graph of relationships is the ability to process it analytically. Processing of this graph may become what data mining is for data stores. For example, performing clustering on the relationships could yield groups of highly interconnected artefacts, which are likely to be difficult to maintain (a change on a single element would propagate to the entire group). In the case of badly administered projects, where the graph of relationships is very dense (there is no clear separation of requirements, or components in the project), one could make an attempt to automatically calculate the cost of getting rid of some relationships, in order to make it more modular (i.e. split it into two or more low-overlapping subgraphs). Such an approach could be useful when thinking about refactoring of an existing code base.

Analytical processing of relationship graphs could also be performed in order to calculate complex software metrics, involving not only the complexity of elements of the projects, but also relations among them. Finally, there is a very tempting idea of having a rule-based workflow process defined on top of traceability graph. Assuming that for each type of relationship we have a set of rules, we can imagine a situation when, for instance, creation of a new source code file triggers the action of creating a new test case, documentation and linking them all together with appropriate

relationships.

Possibilities of utilizing project meta-data and traceability layer are numerous. We hope Ophelia will become a test bed, where can further explore these ideas. Thus far no process of using and defining relationships among project elements has been defined. Without such a process, it is impossible to control and maintain the graph of relationships, which in turn can get so complex and dense that it would no longer be useful.

5 Conclusions

In summary, the Ophelia traceability layer considers all the artefacts of the software development lifecycle and provides a mechanism to create and maintain relationships between these artefacts. These associations can be tracked between any given artefacts at any one time and are always consistent. This has been difficult to achieve in the past for the following reasons:

1. Tools from different vendors are usually closed applications with no possibility of integration with other tools.

2. New elements of the project, and at least some of the associations between these elements, would have to be automatically added to the traceability graph in order to maintain consistency.

3. A physical repository of relationships must hold references to artefacts in proprietary tools databases.

4. Tools are needed to utilise and manage the relations between project artefacts (e.g. visualisation of the traceability graph, change notification and integrity of relationships).

As discussed previously, one potential use of the traceability relations is automated change propagation. Although this may be a Utopian ideal in totality, the traceability layer does provide a mechanism through which any advances in this area can be accommodated. The novelty of the Ophelia solution is that the entire project becomes the scope for traceability rather than a more restrictive subset as is the case with existing solutions.

Acknowledgements

The authors are grateful for the support of the Ophelia project (funded by the Information Society Technologies Programme within the European Union's Fifth RTD Framework Programme, award IST-2000-28402 and IST-2000-28402D).

References

[1] Scott Ambler. Tracing your design. Software Development Magazine http://www.sdmagazine.com/, April 1999. last accessed 16 October 2002.

[2] R G Dewar, L M MacKinnon, R J Pooley, A D Smith, M J Smith, and P A Wilcox. The ophelia project: Supporting software development in a distributed environment. In *Proceedings of the IADIS International WWW/Internet Conference*, Lisbon, Portugal, November 2002.

[3] Jr. Frederick P. Brooks. No silver bullet: essence and accidents of software engineering. *Computer*, 20(4):10–19, 1987.

[4] O Gotel and A W Finkelstein. An analysis of the requirements traceability problem. In *Proceedings of the International Conference on Requirements Engineering*, pages 94–102, Colorado Springs, Colorado, 1994.

[5] Maciej Hapke, Andrezej Jaskiewicz, and Sergio Perani. Ophelia - open platform and methodologies for development tools integration in a distributed environment. In *Proceedings of the 3rd National Conference on Software Engineering*, pages 189–198, Otwock, Warsaw, 2001.

[6] Ivar Jacobson. A resounding yes! to agile processes – but also to the next big thing. The Rational Edge http://www.therationaledge.com/, March 2002. last accessed 16 October 2002.

[7] OMG. Home Page. http://www.corba.org/. last accessed 16 October 2002.

[8] James S O'Neal and Doris L Carver. Analyzing the impact of changing requirements. In *Proceedings of the IEEE International Conference on Software Maintenance*, pages 190–195, Florence, Italy, November 2001.

[9] Ophelia. Home Page. http://www.opheliadev.org/. last accessed 16 October 2002.

[10] Balasubramaniam Ramesh. Factors influencing requirements traceability in practice. *Communications of the ACM*, 41(12):37–44, 1998.

[11] Balasubramaniam Ramesh. Process knowledge management with traceability. *IEEE Software*, 19(3):50–52, 2002.

[12] Rational. Rational Enterprise Edition Home Page. http://www.rational.com/products/entstudio/index.jsp. last accessed 16 October 2002.

[13] Rational. XDE Home Page. http://www.rational.com/products/xde/index.jsp. last accessed 16 October 2002.

[14] P A Wilcox, M J Smith, A D Smith, R J Pooley, L M MacKinnon, and R G Dewar. Ophelia: An architecture to facilitate software engineering in a distributed environment. In *Proceedings of the 15th International Conference on Software and Systens Engineering and their Applications*, Paris, France, December 2002.