A CORBA-Oriented Approach to Heterogeneous Tool Integration; OPHELIA

P. A.Wilcox, C.R.Russell, M.J.Smith, A.D.Smith, R.J.Pooley, L.M.MacKinnon, R.G.Dewar
Department of Computer Science
Heriot-Watt University, Edinburgh, UK

{ P.A.Wilcox, C.R.Russell, M.J.Smith, A.D.Smith, L.M.MacKinnon, R.J.Pooley, R.G.Dewar}@hw.ac.uk

D.Weiss
Institute of Computing Science
Poznań University of Technology
Poznań, Poland

dawid.weiss@cs.put.poznan.pl

ABSTRACT
In this paper, we describe an approach to integrating heterogeneous software engineering tool sets using CORBA. The framework we have developed to bring about the integration is called OPHELIA and we report on an instantiation of that framework and how we have been able to create added-value service on top of our core architecture. These services, such as event driven metrics, are brought about by our success in being able to expose the different genres of software engineering data artefacts. Using these exposed artefacts, we can build relationships between them, users and other applications - so called traceability relationships.

Keywords
Tool Integration, CORBA, Traceability, Event-Driven Metrics

1. INTRODUCTION
Traditionally the software industry has developed systems to support its activities across the software lifecycle and there is extensive tool support for many modern software engineering processes. Such tools have been aimed at co-located teams and many use proprietary data formats and APIs. This means it is difficult, if not impossible to integrate the tools into one software production environment. Recent trends toward increasing globalisation, multi-company projects and component based software engineering have seen a shift toward tool support for distributed teams and individuals working on collaborative software engineering projects across traditional geographic and temporal boundaries [1, 2]. This paper is an introduction to the OPHELIA platform, a CORBA-oriented [3] architecture that supports such distributed and collaborative software engineering activities. Our aim is to provide an insight into the somewhat unexpected added value consequences that we have discovered by pursuing tool integration.

2. THE OPHELIA PROJECT
The OPHELIA Project [1, 2, 4, 5, 6, 7] is an EU funded initiative to develop an open source platform that will allow heterogeneous software engineering tools to be integrated and support collaborative working in a distributed environment. The project consortium comprises a range of partners encompassing both professional IT organisations and academia - the consortium details are included in the acknowledgements section of this paper.

Software projects make both extensive and intensive use of tools and, as identified in [1,5], there is an increasing tendency for projects to be undertaken across geographically distributed locations. There are some commercial solutions that partly address the issues of integration, for example [8], however, these can be very expensive and are tightly constrained to a set of utilities delivered by one particular software vendor. There is limited tool support for distributed software engineering activities and, as it currently exists, this is specific to particular toolsets [5, 9].

The OPHELIA project aims to provide support for integrating heterogeneous tools by defining a set of standard interfaces to access and operate on the artefacts that the tools produce (we consider generic tool artefacts commonly seen across the software development lifecycle such as requirements, models, code, project schedules, bugs and documentation). The interfaces are described in terms of CORBA IDL definitions [3]. We have coined the term Module Interface Specification (MIS) to describe the interface to a particular artefact type and these are currently defined for requirements, UML Models, repositories, bugs, project management schedules, metrics and documentation. By wrapping the data that a particular type of tool produces as a CORBA object, exposing interface methods defined in the MIS, the artefact becomes seamlessly available to the rest of the OPHELIA environment. In addition, and as a consequence of standard interfaces that all objects in OPHELIA conform to, OPHELIA specifies several added value services including traceability, event notification, event driven metrics and knowledge management. As part of our proof of concept we have developed a prototype implementation called Orpheus to test the interface definitions and the platform architecture, this is currently at the beta release stage.

3. THE OPHELIA ARCHITECTURE
The OPHELIA Architecture is illustrated in Figure 1 and comprises four distinct layers. The Kernel Layer provides 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. Project Administration and User Administration provide facilities to manage projects and users of the OPHELIA platform.

The OPHELIA Tool Modules Layer comprises the set of interfaces that expose artefacts of a particular tool type. Individual CORBA servers, one per artefact type, implement the MIS. This

makes the data and services of one module transparently available for use by another, thus facilitating interoperability between tools.

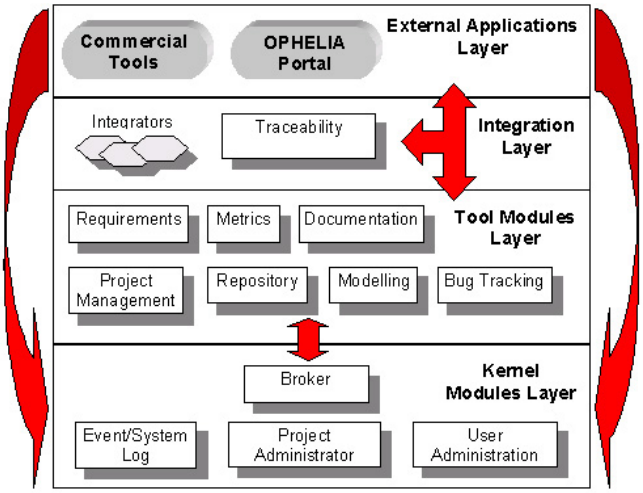

Figure 1. The OPHELIA architecture

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, event notification and event driven metrics, all of which are discussed in the Consequences section.

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. The one caveat that applies to integrating the data of a particular tool is that there must be either a plugin API, programming "hooks" or open source code available to enable the tool to be connected to the OPHELIA environment.

Figure 2 illustrates a typical example of how a tool (in this case the UML Modeling tool, ArgoUML [10]) is integrated with OPHELIA via a plugin. Project data is exchanged between the plugin and modeling server using an MIS interface. The modeling server physically persists model data in an XML repository.

We have successfully plugged in other tools, for instance MS-Project. We have not yet integrated Eclipse [11] through its implicit plugin-oriented architecture, although we see this is a relatively simple piece of future work.

4. CONSEQUENCES OF THE CORBA APPROACH

Tools integration using CORBA is far from new as can be shown by a simple trawl of the Web. Indeed, the CORBA approach is often criticised, sometimes unfairly, from complexity and performance perspectives [12].

Furthermore, here we are merely presenting a sanitised description of our endeavors. Empirical work is underway to benchmark OPHELIA using process-oriented distributed tasks.

OPHELIA MISs enable the artefacts of a project to be accessed in a uniform manner. This project wide, seamless access to all artefacts, has allowed us to develop several services on top of the Tools Modules layer. Such services currently include knowledge management (a cross-project documentation service that, for brevity, we will not consider here) traceability, event notification and event driven metrics.

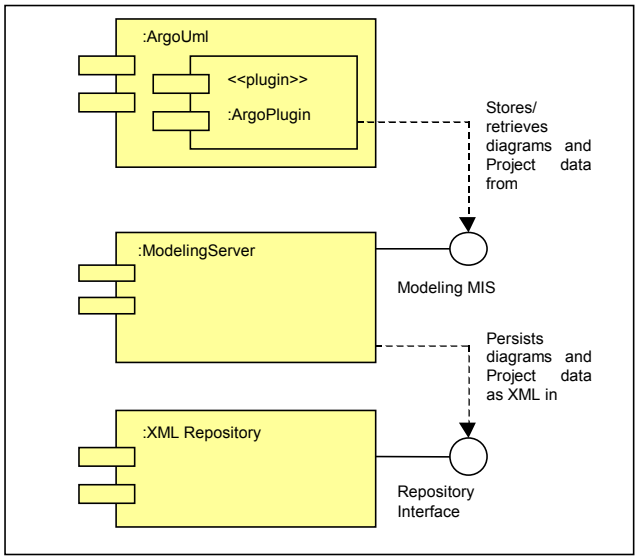

Figure 2. Modeling tool integration

4.1 Traceability and Notification

Ramesh [13] defines traceability as the ability to trace the life of an artefact from its inception to its use. Much of the relevant work in the literature is focused toward requirements traceability [14] although by definition, artefacts could be any project elements like requirements, code, models, documentation, bug reports or plans. Current tools provide limited support for traceability and consistency management between artefacts [15, 16]. In particular, they do not provide developers with the flexibility to create relations across all project elements regardless of their type [17, 18, 19]. Project-wide traceability can help align changing customer needs with the software, reduce project risk through knowledge capture, help determine the impact of a change to the system and help process improvement [20].

The OPHELIA platform unifies programmatic access to project elements and creates a global view of all artefacts, no matter what specific tool has been used to create them or where the artefact is stored. We can therefore create relationships between any given project elements, for example defects and code, models and code, code and unit tests, even if the tools used to create them would not normally support such functionality.

We have developed an application called Traceplough [7, 21] that allows users to define such relationships. Relationships are considered as directed, with an associated type and they may or may not be able to propagate events. The semantic meaning of a relationship is conveyed in natural language through the type of that relationship. Types of relationship are defined by the user in

the Traceplough application. So, for example, we could create a relationship between a requirement and a UML Use Case model with the type «documented in». The Traceplough application also includes an object relationship browser that displays the graph of relationships.

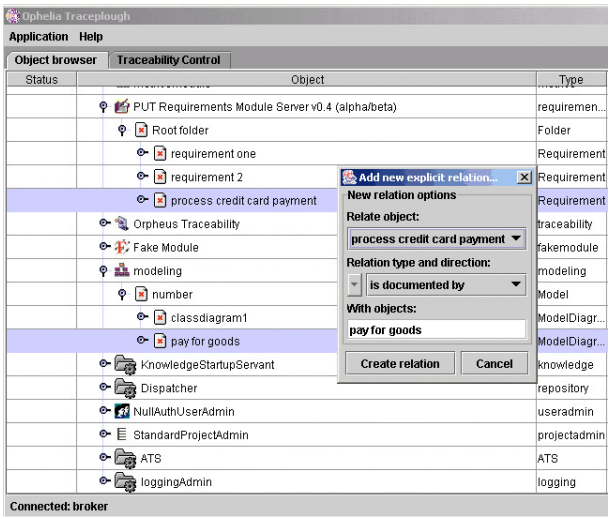

Figure 3. Adding a named relationship between a model and a requirement in Traceplough

The relationships are stored in a relational database, independently of the objects they are defined on. It speeds up processing such as propagation of events, displaying related objects and, most of all, allows tracking relationships to objects which are not physically available (i.e. one of the servers has been shut down or there is a network failure).

As well as being able to create and maintain traceable relationships, the OPHELIA platform makes use of a standard event mechanism that allows users and applications to subscribe to events on OPHELIA objects. These events are translated from internal module-specific events into abstract definitions that all other components of the platform can interpret (we assume all modules can and do translate their internal events). Through a combination of notifications and traceable relations that propagate events we are able to alert users about changes in the project that may impinge directly on their work.

Obviously, traceability defined on top of the OPHELIA architecture is constrained by the granularity of objects exposed by concrete modules implementing MIS interfaces. Currently this granularity varies between modules; requirement module exposes it down to the level of a single requirement, modeling module exposes model elements like use use-cases and classes, documentation module exposes only top-level documents (although splitting structural documents into sections is planned).

We believe that the required granularity level will be highly dependent on the development processes applied within an organization using OPHELIA. It may be sufficient for some projects to have relationships defined at a coarse level, while others may need a finer granularity in order for traceability to be useful. The practical usability study of traceability is not yet available, but we recognise the issues regarding granularity in relation to relationships, graph complexity, density and potential explosions of propagated events. Planned usability trials will address these issues.

4.2 Event Driven Metrics

Software metrics are measurements of quantitative attributes of the software system [22] - typically related to specific project artefacts, for instance schedules, bugs or code. They are useful in defining a standard way of measuring system characteristics such as software size, cost and project difficulty [23].

Many existing metrics engines (tools to collect, calculate, update and display metrics) are configured to generate metrics at periodic time intervals or even after human intervention. However, software development and maintenance activity is sporadic and unpredictable. As a result periodic updates of metrics merely sample the data. Indeed, sampling may well miss vital changes and does not capture the full history of a metric over time - time being a key dimension of metric data [24]. Moreover, Gopal *et al.* [25] found that increasing the frequency with which metrics are captured had a positive impact on project managers' opinion of the metric validity, consequently encouraging the use of such metrics in decision-making. On the other hand, updating metrics too frequently, when nothing has changed, is inefficient.

Alternatively, if we could automatically react to any changes that happen to artefacts (feature requests, class diagrams, etc.), instead of relying on periodic updates, we could optimise the process and capture a complete metric change history. Such a history would be useful to maintenance tools that detect and predict problems with software [26, 27]. We call these artefact change reactions event-driven metrics.

The OPHELIA metrics module is designed to be event-driven and process metrics when existing artefacts are changed or new artefacts are added. This is achieved by utilising the notifications that are built on top of the traceability relationships. Therefore, not only do we have application-user relationships; we also have associations between applications. In this way we are able to build up an exact change history of a development project. The current implementation of the metrics module processes project schedule artefacts, UML model artefacts and requirement artefacts. As an example, we can consider the relationship between the OPHELIA modeling module and metrics module as illustrated in Figure 4.

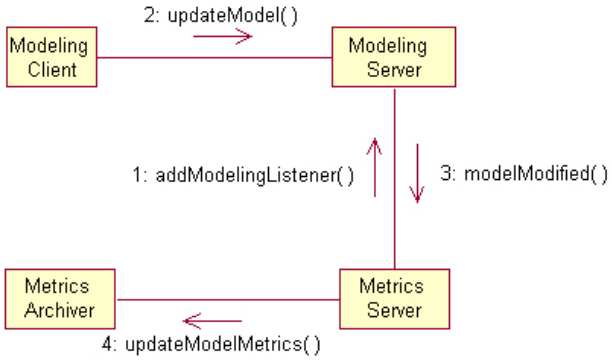

Figure 4. Modeling-metrics interaction in OPHELIA

The metrics server registers as a listener on the modeling server to receive model-added and model-changed events. Upon receiving

these events, the metrics for that model artefact are recalculated and archived so as to build up the entire change history for a given model. The same approach is used to generate the metrics' history for the other artefact genres discussed above.

5. SUMMARY

The OPHELIA project has created a platform that enables organisations to use a custom set of commercial and/or bespoke tools in a distributed environment. The platform facilitates collaborative working across such a distributed environment. We have described the four tier architectural model that supports the platform and provides mechanisms for integration, traceability and interoperability.

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. Thus, the entire project becomes the scope for traceability rather than a more restrictive subset, as is the case with existing solutions. We have described an event-driven architecture to capture the temporal dimension of metrics data based on software artefact change. This architecture is implemented within the OPHELIA project. This approach has advantages over manual or automatic sampling of data in terms of providing a complete artefact history and potentially encouraging project stakeholders to make more use of metrics.

Future short term work for the final release of the Orpheus solution, which is due to be delivered towards the end of 2003, includes performance and memory management, as well as increasing the system's functionality. We are currently considering how to combine the event driven metrics with the notion of "triggers", where updated metrics are compared with threshold values and notifications are only sent when thresholds are breached. Use of such "triggers" can effectively focus and filter the use of metrics information [28]. This mechanism would enable project managers, developers and maintenance engineers to align their metric notification with their individual concerns.

As we have already stated, we are planning benchmarking and usability trials to help us compare OPHELIA with other solutions and to help plan the evolution of the technology ahead of marketing trials.

ACKNOWLEDGEMENTS

OPHELIA consortium partners are: Azertia, Spain; GUTURA, Germany; Heriot-Watt University, United Kingdom; I.C.C.C., Czech Republic; OMEGA, Italy; Poznan University of Technology, Poland. The authors are grateful for the support of our partners and the EU (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] C Boldyreff, M Smith, D Weiss, D Nutter, P Wilcox, S Rank, R Dewar, Environments to Support Collaborative Software Engineering, 2nd Workshop on Cooperative Supports for Distributed Software Engineering Processes (CSSE), March 25, Benevento, Italy, 2003.

[2] 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, 15th International Conference on Software and Systems Engineering and their Applications (ICSSEA), December 3-5, Paris, France, 2002.

[3] OMG CORBA WWW site, http://www.corba.org/ (current June 2003).

[4] OPHELIA Project WWW site, http://www.OPHELIAdev.org (current June 2003).

[5] Hapke, M. et al. OPHELIA – Open platform and methodologies for development tools integration in a distributed environment. In Proceedings of the 3rd National Conference on Software Engineering, Otwock/Warsaw, pp. 189-198. 2001.

[6] Dewar RG, MacKinnon LM, Pooley RJ, Smith AD, Smith MJ, Wilcox PA (2002) The OPHELIA Project: Supporting Software Development in a Distributed Environment, IADIS International WWW/Internet 2002 Conference, Lisbon (November 2002).

[7] M Smith, D Weiss, P Wilcox, R Dewar, The OPHELIA Traceability Layer, 2nd Workshop on Cooperative Supports for Distributed Software Engineering Processes (CSSE), March 25, Benevento, Italy, 2003.

[8] Rational. Rational Enterprise Edition Home Page. http://www.rational.com/products/entstudio/index.jsp (current June 2003).

[9] GENESIS Project WWW site, http://www.genesis-ist.org/english/default.htm (current June 2003)

[10] ArgoUML, http://argouml.tigris.org/ (current June 2003).

[11] Eclipse, http://www.eclipse.org/ (current June 2003).

[12] Juric MB, Rozman I, Hericko M, "Performance Comparison of CORBA and RMI" Information and Software Technology Journal, Elsevier Science, October 2000, vol.42, no.13, pp. 915-933.

[13] Ramesh B. (2002). "Process knowledge management with traceability". IEEE Software, 19(3):50–52.

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

[15] Gray J. P., Liu A. and Scott L. (2000). "Issues in software engineering tool construction". Information and Software Technology, 42(2):73–77.

[16] Harrison W., Ossher H. and Tarr P. (2000). "Software engineering tools and environments: A roadmap". The Future of Software Engineering.

[17] Nuseibeh B., Easterbrook S. and Russo A. (2000). "Leveraging inconsistency in software development". IEEE Computer, 33(4):24–29.

[18] Grundy J. C., Hosking J. G. and Mugridge W. B. (1998). "Inconsistency management for multiple-view software

development environments". IEEE Transactions in Software Engineering, 24(11):960–981.

[19] Nentwich C., Emmerich W. and Finkelstein W. (2001). "Static consistency checking for distributed specifications". In Proceedings of the 2001 Automated Software Engineering Conference, pages 26–28, San Diego, CA, USA.

[20] Ambler S. (1999). "Tracing your design". Software Development Magazine http://www.sdmagazine.com/. last accessed 16 October 2002.

[21] Krzysztof Kowalczykiewicz, Dawid Weiss, Traceability: Taming uncontrolled change in software development. IV Krajowa Konferencja In¿ynierii Oprogramowania, Poznañ, 2002

[22] DeMarco, T., Controlling software projects: management, measurement and estimation, Prentice Hall, New Jersey, 1982.

[23] R. Pooley, D. Senior, D. Christie, "Collecting and analyzing Web-based project metrics". IEEE Software, Volume 19, Issue 1, IEEE Computer Society , Jan.-Feb. 2002, pp. 52-58

[24] R. A. Paul, T. L. Kunii, Y. Shinagawa, M. F. Khan, "Software Metrics Knowledge and Databases for Project Management". IEEE Transactions on Knowledge and Data Engineering, Volume 11, Issue 1, IEEE Computer Society, January 1999, pp 255-264.

[25] A. Gopal, M. S. Krishnan, T. Mukhopadhyay, D. R. Goldenson, "Measurement Programs in Software Development: Determinants of Success". IEEE Transactions on Software Engineering, Volume 28, Issue 9, IEEE Computer Society, September 2002, pp 863-875.

[26] H. A. Sahraoui, R.Godin, T. Miceli, "Can Metrics Help to Bridge the Gap Between the Improvement of OO Design Quality and its Automation". in Proceedings of the International Conference on Software Maintenance 2000, IEEE Computer Society, San Jose, California, USA, October 2000, pp 154-162.

[27] T. L. Graves, A. F. Karr, J. S. Marron, H. Siy, "Predicting Fault Incidence Using Software Change History". IEEE Transactions on Software Engineering, Volume 26, Issue 7, IEEE Computer Society, July 2000, pp 653-661

[28] Software Program Managers Network (SPMN), "16 Critical Software Practices", http://www.spmn.com/16CSP.html (current June 2003).