Active Artefact Management for Distributed Software Engineering

Cornelia Boldyreff David Nutter Stephen Rank
Research Institue for Software Evolution
Department of Computer Science
University Of Durham, UK
{*cornelia.boldyreff,david.nutter,stephen.rank*}
@durham.ac.uk

Abstract

We describe a software artefact repository that provides its contents with some awareness of their own creation. "Active" artefacts are distinguished from their passive counterparts by their enriched meta-data model which reflects the work-flow process that created them, the actors responsible, the actions taken to change the artefact, and various other pieces of organisational knowledge. This enriched view of an artefact is intended to support re-use of both software and the expertise gained when creating the software. Unlike other organisational knowledge systems, the meta-data is intrinsically part of the artefact and may be populated automatically from sources including existing data-format specific information, user supplied data and records of communication.

Such a system is of increased importance in the world of "virtual teams" where transmission of vital organisational knowledge, at best difficult, is further constrained by the lack of direct contact between engineers and differing development cultures.

1 Introduction

The GENESIS environment is intended to support distributed software engineering by providing a lightweight process-agnostic tool-set incorporating work-flow description and enactment, agent-based interaction between clients and an artefact management system, OSCAR [16]. The environment shall be released as Open Source Software at the end of the first development cycle.

In this paper we discuss only the artefact management subsystem in GENESIS. This paper considers the notion of active software artefacts in Section 2 and in Section 3 proposes an architecture for the distributed repository system that shall be developed to manage them. As GENESIS is in the early stages of development only high-level requirements and goals are considered.

2 Introducing the Active Artefact

OSCAR active artefacts consist of two major components: the *meta-data* that describes their properties and the *artefact data* which contains the full data (source code etc.) described. Software artefacts naturally consist of more than just software code; they include items produced in all parts of the software process, both informal and formal and much conceptual information that aids comprehension [10]. A similar notion for software code alone is the concept of Literate Programming [15] which aims to combine software code with its documentation and other useful information to ensure that documentation is updated alongside the software. The change in practice required when employing literate programming is large and therefore active artefacts aim to provide some of the benefits (traceability, rationale capture etc.) without additional overhead.

Alongside such traditional software engineering support processes as configuration management and version control, the active artefact is intended to facilitate re-use and transmission of organisational knowledge between engineers working with the artefact and consequently greater productivity through collaborative learning [17]. To this end the artefact shall record and present process information including the actors responsible for changes and, if desired, the rationale associated with those changes. Other presence information shall be stored with the intention of encouraging encounters between engineers interested in the same artefact; a similar goal to that described by Boyer et al [5]. This information will be largely retrospective as it shall be gathered by monitoring user activities.

The relationships between artefacts and between the tools used to manipulate them shall be recorded to allow a consistent approach to working with the artefact. Any dis-

cussion or informal communication about artefacts shall be recorded as annotations

The artefact structure may be extended to add or specialise properties and add additional structural information. An open data format such as XML shall be used when exposing the artefacts to the outside world, allowing easy transformation of artefacts to other data types. To this end artefact properties should be extensible and inheritable. All artefacts shall be specialised from an abstract base type describing various general properties that all artefacts have. Artefacts may also be defined recursively (one artefact containing several sub-artefacts) allowing inheritance of properties from the parent artefact. Active artefacts should be largely independent of their underlying storage mechanism; in particular export/import of data from existing systems should be possible albeit with the risk of losing some of the artefact's functionality. Finally, artefacts may be manipulated by both human and machine actors within GENESIS. This requirement obviously places some constraints on the structuring of the artefact to ensure they are comprehensible to both.

2.1 Example

To illustrate how we envisage active artefacts behaving, consider the example of a software component providing a GUI widget. Below the properties of the component when encoded in each of the two artefact forms are compared:

A **passive artefact** merely contains the software code, change information and documentation including simple usage examples and may be spread over several distinct files. In addition to this the **active artefact** records logs of who and what has accessed the artefact and (if possible) their purpose in doing so, any informal notes they added during the access and any metrics output related to the artefact.

Active artefacts also possess links to process information and related artefacts in the manner of the Debian Project's software packages [12]. In a search situation, similar artefacts will also be indicated.

When a user wishes to deploy the component, they will first retrieve the artefact via a search. In the case of the passive artefact, search terms are limited to terms in the source and documentation while in the case of the active artefact search terms can also include things such as quality requirements (from metrics), presence related information etc. Immediately this gives users a richer set of search terms to describe the properties of the artefact they require.

Once the component has been retrieved and deemed suitable for use, the next step is to integrate it with the rest of the project. In the case of the passive artefact all the information provided is the formal documentation, which may be incomplete, inaccurate or simply inapplicable to the current problem domain. In the case of the active artefact the experiences of previous users will have been recorded and consequently the developer may discover whether previous users were interested in that artefact, their reactions to it and any case studies or examples that they have annotated the artefact with.

Thus, the potential re-user may find out what has happened to the artefact recently. If for example the artefact is of reasonable quality but hasn't been updated for some time this may indicate that the component it encapsulates is moribund and hence a poor re-use candidate. If however the quality is low but significant effort is being expended to solve the problems (and the rate of improvement is commensurate to that effort) the artefact is likely to be a better long term choice. Open source and similar loosely collaborative projects have similar problems; in particular it can be difficult to ascertain whether a project is effectively dead or merely moribund.

These records provide the developer with additional technical and organisational information about the software artefact. As the developer works with the artefact such meta-data shall be augmented with details of that developer's activities and if desired the developer may add their own items of data to increase artefact's value to other users. In a non-invasive environment such as GENESIS this type of data-collection cannot be mandatory. Instead opportunistic or automated data collection and reliance on the goodwill of users to improve "their" artefacts becomes necessary, potentially leading to a slight bias towards older artefacts which will naturally have accrued more information about themselves than newer artefacts.

Such activities are common in the everyday work of our industrial partners. For example, the enterprise resource planning software vendor LogicDIS has provided several use cases which they hope to automate with the GENESIS platform. In particular, the use case describing production of project deliverables incorporates a stage where previously created artefacts are examined by the developers. These artefacts may have been prepared within the current iteration of the software process, or in a previous iteration of the process, or as part of a completely different process or even through activities outside a GENESIS-defined process. In the latter two cases, additional organisational knowledge is essential for the effective re-use of the artefacts, for without it developers will be unable to fully understand the capabilities and original goals of the artefact.

2.2 Related Work

Though software repositories in environments such as Oz [13] are common in many software projects, there has been little application of such systems in the Open Source world. Additionally, none of the existing repositories are

suitable for active artefacts though some work, notably the KAPTUR project [1], implements the rationale-recording aspect of active artefacts. Most of these repositories rely on significant user involvement in the management process, precluding their use within GENESIS.

Nevertheless the case for collaborative environments in open hypermedia development [11] has been convincingly argued. Therefore a clear need exists for a repository to provide the benefits of distributed artefact management to the GENESIS environment. Such environments are often based on some form of database [14, 2, 3] or structured file set like the ProcessWEAVER [8] tool and process management system.

Integrating databases into software engineering support environments has many advantages including a structured query system, ACID [1] properties for changes and open APIs so external programs may easily access the data. Unfortunately retrieving data from the database so unintegrated file-based tools may operate on it is inefficient as files must be reconstructed, operated on by the tool and re-inserted into the database. The storage of large quantities of unstructured, or at least unparsed, data in the database is likewise inefficient in space and access time. The inflexibility of the database schema ensures that such operations become necessary as unforeseen data formats are added to the system.

The Process-centred Software Engineering environments MARVEL, its successor Oz and OzWeb and the SPADE system all referenced above utilise database-type systems only. SPADE and OzWeb choose to employ third-party object oriented databases while the older MARVEL environment relies on a relational-style database built over flat files. The ADELE [4] software engineering environment kernel and work-flow system takes a similarly strict approach (extending the Entity Relationship Attribute model) but does not employ a database system directly.

Ordinary files provide a significant speed advantage over any database management system and remove all the problems associated with the storage of unstructured formats. The ProcessWEAVER tool system takes this approach, with the obvious disadvantage of the loss of the query and transaction services provided by database systems.

In summary, databases are excellent choices where the data stored is highly structured, that structure is fairly static, and any interaction with that data will be performed by a known set of tools fully integrated with the database. SPADE in particular takes this approach by forcing the integration of system tools with the FUSE tool service. Though SPADE/FUSE permits the invocation of arbitrary tools (via the notion of "black transitions" in the process model) the intention is clearly to force tight integration of tools into the environment.

[1]The OMG's transaction specification: *A*tomicity *C*onsistency *I*ndependence and *D*ependability

GENESIS intends to provide a different notion of software artefacts than that found in process-centred environments such as those described above. Instead of providing an object-oriented view of artefacts by default as SPADE, Oz etc. do, OSCAR shall provide a document-oriented view of the artefacts it stores. While a structured document is not a "plain file" it is closer to that than to an object and can be manipulated in much the same way.

Since the document is already structured, building objects or other high-level data structures above the document is easier than building comparable structures above a plain file. This approach is intended to provide a middle ground between the heavyweight, powerful and expressive but inflexible object-oriented SEEs and the lightweight, flexible, but less powerful file based systems.

3 OSCAR

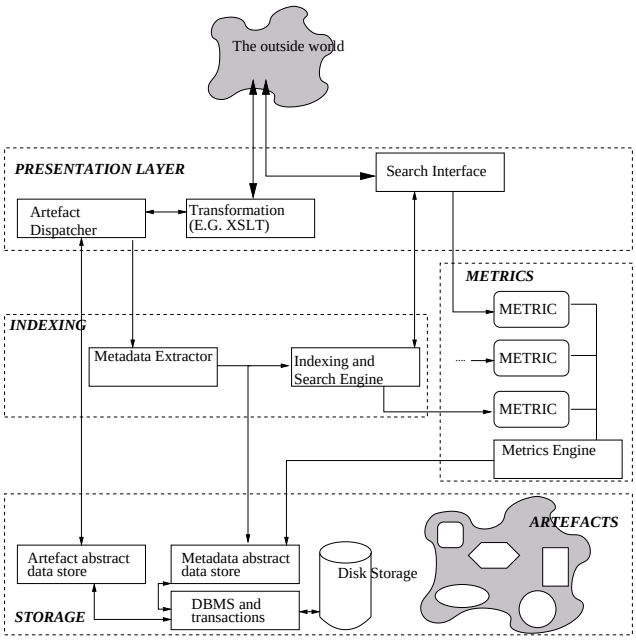

Figure 1. OSCAR Architecture

Figure 1 proposes an architecture[2] for a distributed repository system to manage the active artefacts described earlier. The four key modules within this architecture are described below.

The **presentation** layer is responsible for presenting artefacts in an applicable form for the various clients of the system and providing higher level interfaces to OSCAR functionality such as search and artefact modification. Additionally the presentation layer shall allow the exposure

[2]Reproduced from our requirements paper referenced earlier.

of different interfaces depending on the deployment of OSCAR. For example, while access via WebDAV [9, 6] might be convenient for distributed workers, tighter access methods such as RPC may be more appropriate for workers on a single LAN.

Within the **indexing** layer artefacts have their meta-data extracted to allow the fulfilment of search requests received from the presentation layer.

The **storage** layer handles the decomposition of artefacts into *meta-data* and *artefact data* and consignment of this data to separate data stores which suit their particular storage requirements. At this level most of the distribution is expected to take place and consequently transaction services are localised within this layer.

Several storage solutions have been examined for OSCAR and while WebDAV may seem attractive as a combined storage and access solution it should be noted that the DAV protocol provides only limited services for configuration management and version control at the present time. The GENESIS platform requires more flexibility than permitted in the DAV specification at the present time; in particular the rigid check-in/check-out model of version control is not necessarily the best solution for a non-invasive environment as it enforces a very particular style of working.

The current lack of widely deployed and tested DAV solutions is also a barrier to fully adopting such software in GENESIS, though projects such as the Jakarta "Slide"system and the addition of DAV to existing Open Source content management systems indicate that the situation is improving. The modular architecture of OSCAR shall permit the integration of any required WebDAV modules at a later date.

Finally the **metrics** engine shall enact user-defined metrics in the background to study activity within the repository and will augment the artefacts studied with the results.

OSCAR will be responsible for managing the software artefacts produced by users of the GENESIS system leading to the following specific high-level requirements. Firstly, every artefact will possess a *unique identifier* in a name-space associated with a particular GENESIS deployment. Visibility of an artefact within this name-space depends on both the role permissions of the artefact and the permissions of the user. One **global name-space** will exist per network of OSCAR repositories.

Secondly, alongside a standard keyword search mechanism, OSCAR shall provide a similarity matching system for artefacts. Both systems shall operate on the artefacts' meta-data to retrieve details of candidate artefacts. Once the client has selected an artefact it may be retrieved directly from the repository where it is stored without further reference to the meta-data.

Thirdly, within OSCAR everything shall appear as an artefact, even superficially different entities such as actors or software tools that OSCAR employs. This approach has much precedent (UNIX etc.) and provides benefits such as a consistent method for manipulating and accessing *all* items in the repository.

Fourthly, though the meta-data will undergo continuous modification, artefact data will be altered rarely (though often read). Consequently, when artefact data is written the current state of the meta-data should be replicated within the artefact data, allowing the meta-data service to be rebuilt in the event of failure.

Fifthly, every GENESIS client may optionally instantiate OSCAR to store artefacts locally, connect to a remote OSCAR or both. Since the OSCAR system is formed of two separate physical data stores, OSCAR networks may follow several distribution models, including:

- Centralised meta-data and artefact data store (star topology).
- Multiple artefact stores and single meta-data store.
- Fully distributed: multiple artefact and meta-data stores with replication.

Obviously there is a tradeoff here: consistency of information against the dependability of the repository network as highly distributed and redundant networks will be resilient to nodes failing but will incur a high overhead in keeping all nodes up to date. The centralised model will have no consistency problems but possesses a single point of failure.

The storage requirements for the meta-data and artefact data differ. The former requires fast random access read/write/query operations but little space, the latter requires plenty of space and fast sequential read/write operations only. Consequently two physical repositories of data must be maintained and presented as one virtual repository.

The requirement for a non-invasive system suggests performing consistency updates on a "best effort" basis as the difficulty of maintaining multiple synchronous system views without a centralised system has been noted [7]. Consequently the view of OSCAR for certain clients may often be somewhat outdated; however whenever a critical change must be made the clients concerned should synchronise their view of the repository to ensure no unintended changes are made. Due to this requirement OSCAR must collect any information about user activities passively without demanding user for input. OSCAR must integrate easily with existing software tools in the organisation where it is deployed. A distinction shall be made here between tools that are invoked by OSCAR itself to perform a task and systems that act upon the repository from outside. The latter may be considered a type of actor while the former requires a specialised artefact type to represent it. The location where

tools are invoked is important too; tools invoked on a client machine can gather user input while tools invoked remotely cannot.

OSCAR shall also provide a set of predefined artefacts that will be useful in most situations. These shall include *Software* which represents any intended output from a software development effort, not just source code; *Annotation* which encapsulates supplementary information added by users; *Tools* to represent external programs; *Actors* which store skill, security and role information for repository users; *Process Elements* and *Process Instances* which represent reusable process segments and executed processes respectively.

3.1 Simple Example Artefact

Figure 2 illustrates the different artefact representations in our initial OSCAR design and prototype implementation. For clarity and brevity the UML, XML document and entity relationship diagrams are all incomplete and concentrate only on certain key features of the artefact.

At the highest level, a set of classes with various operations and attributes exist, mirroring the relationships between artefacts and their contents and providing methods to manipulate the properties of the artefact. The first class, **Artefact**, contains the basic artefact structure and its data, acting as a fly-weight for the other artefact-related classes. Secondly, the **Property** class represents a simple property of the artefact, in this case a string, though there are other property types available.

For certain specialised properties custom classes are essential. The **Version** class contains the version information for an artefact. A single instance of an artefact will have one version as shown by the UML; however, an artefact within the system will exist in multiple versions.

Finally the **Relationship** class expresses a relationship between one or more artefact instances. The class shown is a simplified representation of that used to encode simple links (analogous to a hyper-link) between artefacts. More complex link types similar to advanced XLink links shall be present in the final version of OSCAR. Properties of relations between artefacts include the version of the artefact referred to (not shown), the link type (a type of *depends* is "stronger" than a type of *related* for example) and the expected behaviour of OSCAR clients when they encounter the link.

The fly-weight **Artefact** class shall maintain internally a structured document similar to that shown in the figure's mid-part. The parts of the document directly related to the UML above are labelled, as are the places where document extension may take place.

Also visible is a location for artefact data such as software code; a property omitted from the UML. While a

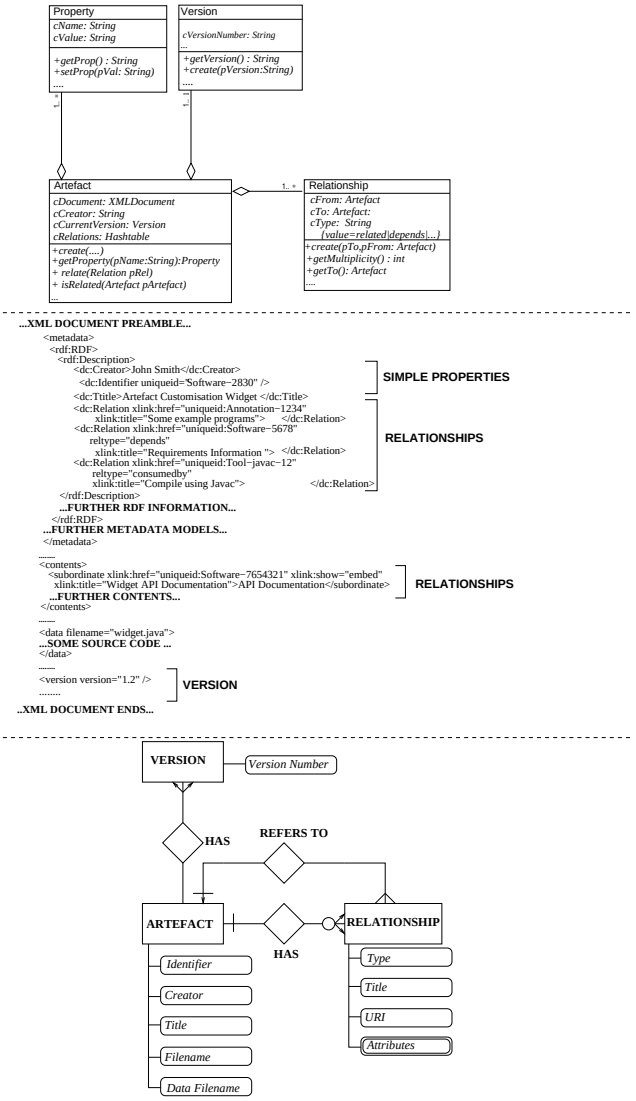

Figure 2. Example artefact

structured document such as XML lacks the powerful capabilities of objects such as strong typing, task-domain specific APIs, true inheritance and so forth it does have sufficient structure for those capabilities to be built upon it and possesses flexibility; whilst modifying the behaviour of non-trivial objects requires significant effort, modifying the structure and content of an XML document is relatively easy assuming one does not break any validity constraints. The Document Type Definitions (DTDs) within OSCAR are sufficiently modular and flexible to permit content model modification for almost all elements.

Greater interoperability is also useful; objects are usually tied to a specific language (e.g. Java) or platform (e.g. CORBA) but XML is intended as an interchange format be-

tween disparate tools. Consequently our structured document is easier to transform into other formats than the relatively heavyweight objects built upon it, allowing systems not completely integrated with GENESIS to participate in projects where GENESIS is in use.

Below the XML document the artefact is further decomposed into the entity relationship (ER) model with a view to inserting it into a relational database. Obviously at this point some enforced constraints are reintroduced by the DBMS which are not present in the XML document, however, assuming the document is valid against the artefact DTD, the information should not conflict. The entities and attributes illustrated by this ER diagram correspond to the highlighted sections in the XML document.

Due to the two-tier storage model adopted by OSCAR and in light of the related work, only properties with a clear correspondence to the relation model need be stored in the database. The rest (and a backup copy of the database information) shall be stored as files under revision control.

4 Conclusions and Further Work

We have presented the notion of active software artefacts aware to a certain extent of the changes they undergo and their position within a set of artefacts. Additionally we have proposed an architecture for a distributed software repository capable of supporting these artefacts within a process aware software engineering environment.

Within the context of the GENESIS project OSCAR will provide support for the software process alongside normal software engineering activities. GENESIS itself shall have two development cycles, resulting in two official open-source releases of OSCAR by the end of the project. It is intended that the second cycle of development shall be in the open to nurture interest in the open source community about GENESIS. To this end we intend to produce a description language and meta-data vocabulary specialised for active artefacts and to design and implement the OSCAR system to support their use.

Acknowledgements

The research described here has been carried out with the support of the European Commission under the GENESIS project. The collaborators in this project are University of Salerno (CRMPA), University of Sannio (RCOST), University of Rome (UNIROMA), University of Durham (RISE), LogicDIS, Mathematical Models and Applications (MOMA) and SchlumbergerSEMA. We acknowledge contributions from our colleagues in all of these organisations.

References

[1] S. C. Bailin, J. M. Moore, R. Bentz, and M. Bewtra. KAPTUR: knowledge acquisition for preservation of tradeoffs and underlying rationales. In *Proceedings of the 5th Annual Knowledge-Based Software Assistant Conference*, CTA Incorporated, September 1990. Rome Laboratories.

[2] S. Bandinelli, A. Fugetta, and C. Ghezzi. Software process model evolution in the SPADE environment. Technical report, CEFRIEL- Politecnico di Milano, December 1993. GOODSTEP ESPRIT-III Project 6115.

[3] S. Bandinelli, M. Fugetta, A. Fugetta, and L. Lavazza. The architecture of the SPADE-1 process-centered see. Technical report, CEFRIEL- Politecnico di Milano, February 1994. GOODSTEP ESPRIT-III Project 6115.

[4] N. Belkhatir, J. Estublier, and W. L. Melo. Cooperative work in large scale software systems. *Journal of Software Maintenance: Research and Practice*, 1993.

[5] D. G. Boyer, M. Cortes, J. Herbsleb, and M. J. Handel. Virtual community presence awareness. *ACM SIGGROUP Bulletin*, 19(3):11–14, 1998.

[6] G. Clemm, J. Amsden, C. K. T. Ellison, and J. Whitehead. Versioning extensions to WebDAV. IETF, 2002.

[7] J. Estublier. Objects control for software configuration management. In *Proceedings of CAISE2001, Interluken, Suisse*, June 2001.

[8] C. Fernstrom. Processweaver: Adding process support to unix. In *2nd International Conference on the Software Process*, pages 12–26, Berlin, Germany, February 1993. IEEE CS Press.

[9] Y. Goland, E. Whitehead, A. Faizi, S. Carter, and D. Jensen. HTTP extensions for distributed authoring WebDAV. IETF, 1999.

[10] T. Green and D.Benyon. The skull beneath the skin: entity-relationship models of information artifacts. *Internation Journal Of Human-Computer Studies*, 44(6):801–828, 1996.

[11] J. M. Haake. Openess in shared hypermedia workspaces: The case for collaborative open hypermedia systems. *ACM SigWEB Newsletter*, 8(3):33–45, October 1999.

[12] I. Jackson and C. Schwarz. The Debian policy manual. http://www.uk.debian.org/doc/debian-policy/ch-relationships.html, 1998.

[13] G. E. Kaiser. WWW based collaboration environments with distributed tool services. *World Wide Web Journal*, 1(1):3–25, 1998.

[14] G. E. Kaiser, N. S. Barghouti, P. H. Feiler, and R. W. Schwanke. Database support for knowledge-based software engineering. *IEEE Intelligent Systems and Their Applications*, 3(2):18–23, 26–32, 1988.

[15] D. E. Knuth. Literate programming. *The Computer Journal*, 27(2), 1984.

[16] D. Nutter, S. Rank, and C. Boldyreff. Architectural requirements for an Open Source Component and Artefact Repository System within GENESIS. In *Proceedings of the Open Source Software Development Workshop*, pages 176–196. University Of Newcastle, February 2002.

[17] P. Sachs. Transforming work: Collaboration, learning, and design. *Communications Of The ACM*, 38(9):36–44, September 1995.