Change Propagation with the Change Notification Bus

Tobias Rodenbach[1] and Lena Wiese[2]

[1]Rosenhagen 12
31134 Hildesheim
tobias.rodenbach@udo.edu

[2]Fakultät für Informatik
Technische Universität Dortmund
44221 Dortmund
lena.wiese@udo.edu

Abstract: We present requirements and a basic architecture for change management across application boundaries. The Change Notification Bus addresses heterogeneity of applications and working environments as well as distribution of team members. Change Notifications are handled in a decentralised and event-based way.

1 Introduction

One key advantage of agile software development is its ability to flexibly react to changing requirements or conditions; this feature specifically allows for incremental development of prototypes with an increased range of functionalities. One crucial aspect are the interdependencies between documents of the different development stages – as for example the product specification depends on the requirements documentation and later on product tests depend on the specification. Ideally, the software product and all documents should be in a consistent state after processing a requirement change.
As a necessary condition for document and product consistency, software development requires "traceability": the ability to identify all artifacts that might be affected by a change starting from the changed artifacts. However, maintenance of traceability is a difficult task during the whole product life cycle which is costly and error-prone if done by hand. This challenge is even more evident as soon as distributed development teams are concerned as changes and follow-up steps cannot be communicated directly. Although management of traceability changes is a major task in software development, there is no off-the-shelf product available that offers comprehensive support and realising traceability involves a lot of manual work of developers.[1]
Although it may be desirable to provide an integrated solution which covers all needs of the development process, this is usually impeded by practical prerequisites; we propose to employ application coupling instead of a fully integrated environment.

[1]This is particularly the case for embedded software; see Section 2 for details.

2 Application Integration versus Application Coupling

In order to automatically support traceability and (at least approximately) achieve a consistent documentation state, all employed development and management applications have to be combined. As is customary in the literature (see [CHKT05, Kel02]), we consider two forms of application combination in this article:

- **Application Integration:** Original applications are altered (or are replaced) to form a single new application. As the data exchange in the new application is direct, external interfaces of the original applications can be removed during this process.
- **Application Coupling:** Existing external interfaces of original applications can be used to exchange data between them. Due to proprietary interfaces this typically involves creation of some glue software that transforms the data from one interface format to another. In this scenario the applications remain self-contained but still form a continuous environment.

When talking about application integration, do the stakeholders involved in the development process (as e.g. customers, managers, developers, testers and administrators) benefit from having an integrated all-purpose software development and business process application; that is a tool that supports and facilitates the whole software development process covering both business and engineering needs? An integrated development environment offers a unified look and feel for all tasks during the development process for as well manager as developers and testers and in the basic setting might reduce installation and maintenance effort for administrators.
However such an environment can only either be a predefined one which does not leave much options for customisation or a very comprehensive one that offers more than is needed which increases configuration effort and potential confusion. An optimal integrated environment can be custom-made for every field of application but in most cases the accompanied development cost and effort will bar this option.
Especially in embedded software development, some tools are only available as stand alone versions or in specialised environments. Frequently legacy systems need to be maintained using old-fashioned tools. These environments usually do not offer adequate possibilities to integrate other parts of the tool chain as it is known from more recent products. They are on the other hand hard to integrate into a larger environment, e.g. because they are lacing remote control or command line interfaces. Even very comprehensive modern integrated environments do not cover all aspects of the development cycle, in particular competitive project management components are rarely found.

There are several reasons why existing applications should be kept in use. First, there are commercial aspects to consider as tools have already been bought or licensed. Second, the users are already acquainted with the existing tools and switching to a new tool would mean starting a new learning curve thus resulting in additional cost due to lower efficiency. In some areas the choice of the respective tool can be even left to the user as long as the tool's output complies with the expected result. Third, working with several independent applications also increases flexibility. When new constraints demand switching to a new

tool for a specific task, the other tools can be kept in use unchanged. Lastly, companies can also choose products by proficiency and cost instead of being forced to use what comes packaged with an integrated environment.

3 An Application Coupling Framework for Change Notifications

This chapter describes the basic architecture of a Change Notification system coupling applications in a distributed environment. The following terms will be used to refer to the different aspects of change management:

Configuration Item	Any item that goes into configuration management. Most likely to be a file, but can also be a folder or all files belonging to a MS Project plan.
Configuration Atom	The smallest coherent part of a Configuration Item that can be affected by a change (subsection describing one function, a class diagram, one requirement, one test case).
Traceability Link	A directed dependency between Configuration Atoms signifying that a change of an Configuration Atom requires the linked Configuration Atoms to be checked for needed changes.
Change Notification	Notification that a Configuration Atom might be affected by the change of another Configuration Atom.

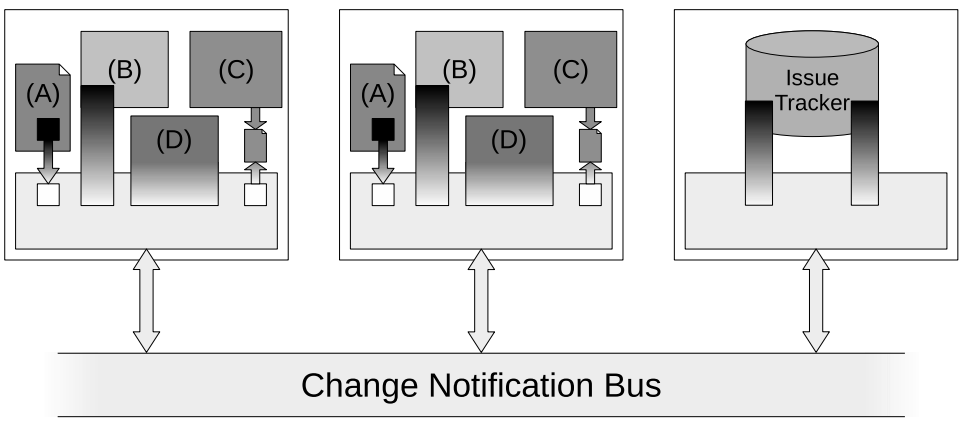

Figure 1: The Change Notification Bus

Figure 1 depicts the proposed architecture. It is divided into a (virtual) “Change Notification Bus” (CNB) and a local framework instance running on each workstation taking part in the development process, the “CNB environment”. Those two elements will be described in the following sections.

3.1 The Change Notification Bus

The CNB is not a physically existing data bus, although it is essential for the proposed architecture that all participating workstations are connected via a network. It is rather a model for the fact that all Change Notifications that are posted reach all connected CNB environment instances immediately. A message bus is a well known design pattern for application coupling, e.g. described in [HW03] as the "Message Bus" pattern.

3.2 The CNB Environment

Each CNB environment instance provides the applications with the means to

- post Change Notifications
- subscribe for Change Notifications
- mark a Change Notification as processed
- create/delete a Traceability Link [2]

For each type of document and/or application, a plugin is hooked to the CNB environment. The different types of coupling are depicted in Figure 1 and explained in the following:

(A) Document level coupling
Some applications allow for functionality augmentation by adding macros into their documents. Those documents can be coupled to the respective plugin in the CNB environment by inserting a macro that will detect changes in the document and report them to the CNB environment. Further, it will receive Change Notifications for the Configuration Atoms in the Documents and inform the user in an appropriate way, depending on the document type.

(B) Application extension
Other applications offer the possibility to extend them by plugin interfaces or application level macros. Those can be extended by bi-lateral plugins, being plugged into the application and the CNB environment at the same time. Analogously to case (A) for document level macros, Change Notifications can be posted and presented to the user in an appropriate way.

(C) Framework-internal handler for not extensible applications/document types
Applications that cannot be extended and handle only passive documents are hard to couple to the CNB environment. However, their documents can still be monitored by the respective plugins in the CNB environment. In this case, the plugin has to provide more functionality than offering the interface for Change Notification

[2] The last item is just mentioned for completeness as building and maintaining the traceability database is out of scope of this document (see Section 4).

handling; it has to be able to read the application's document format in order to detect changes. If relevant Change Notifications shall be displayed in the document, it has to write to the document file. The advantage of this kind of plugin is that document can be accessed "offline", i.e. without a running application.

(D) Native CNB application
For newly developed applications, the CNB environment's interface can be utilised directly without the need for plugin development.

If an application that is not explicitly designed for extension is available in source code it can be altered to access the CNB environment directly. This can be regarded as a blend of the types (B) and (D).

3.3 The Issue Tracker

Although it is convenient to immediately inform the user about Change Notifications referring to parts of a document that he is currently working on, the majority of Change Notifications will refer to currently inactive documents. To manage these pending Change Notifications, they are added as trackable items to the issue trackers database. The issue tracker will monitor the activity on the CNB and close the respective item as soon as a processed message for the Change Notification is posted. Using an issue tracker to track the Change Notifications allows to utilise the means of classical change management for our approach. In particular those changes can be mapped to baselines or branches, so that the respective Change Notifications will also refer to the baseline or branch. This means that the version of the linked Configuration Items can be determined.
The reflection of the Change Notifications in the issue tracker also allows for controlling the changes to the artifacts. Different policies can be applied for different notification types:

- Change Notifications triggered for *tight* links (e.g., from implementation specification to source code) may be processed without further checking; this is the main advantage of our bus architecture because events posted on the bus can be handled by registered application without delay and inside the artifacts directly (as opposed to server-based notifications of [CHCC03]).
- Change Notifications for artifacts with a more complex link structure (e.g., horizontal traceability between equally levelled requirements) need to be reviewed and approved before further action is taken; in this manner the issue tracker supports the implementation of a specified workflow in the framework.

Moreover, the issue tracker is a normal application connected to the CNB as described in Section 3.2. Whenever Configuration Atoms in tracked items like defect reports or change requests are changed, it will post a Change Notification on the CNB.

4 Conclusion and Related Work

We presented a distributed framework for Change Notification handling in a heterogeneous tool environment. The framework focuses on coupling established applications to support a high "return on investment". Notifications about changed Configuration Atoms are based on instantaneously posted events. Event-based notification for change management is also proposed by Cleland-Huang et al. in [CHCC03]. While they rely on a event server, our proposal utilises a bus architecture for broadcasting Change Notifications directly among the participating applications if no approval of the change is needed; otherwise change management workflows can be modelled in the issue tracker. Further, instead of listing the pending Change Notifications in a separate event log, we aim for a *tight* integration with the utilised development and management applications as realised by our four different coupling types; this specifically allows for handling changes *inside* an artifact (e.g., by highlighting affected sections in a text document by coupling of type (A)).
Retrieval of initial traceability information is out of the scope of this article. This topic is addressed by Poirot ([CHBC+07]) and RETRO ([HDS+07]) using information retrieval techniques. Update of traceability information based on predefined policies is covered by ArchTrace ([MvdHW06]).

Acknowledgements
We thank the anonymous reviewers for their helpful comments.

References

[CHBC+07] Jane Cleland-Huang, Brian Berenbach, Stephen Clark, Raffaella Settimi, and Eli Romanova. Best Practices for Automated Traceability. *IEEE Computer*, 40(6):27–35, 2007.

[CHCC03] Jane Cleland-Huang, Carl K. Chang, and Mark J. Christensen. Event-Based Traceability for Managing Evolutionary Change. *IEEE Transactions on Software Engineering*, 29(9):796–810, 2003.

[CHKT05] Stefan Conrad, Wilhelm Hasselbring, Arne Koschel, and Roland Tritsch. *Enterprise Application Integration*. Elsevier, München, 2005.

[HDS+07] Jane Huffman Hayes, Alex Dekhtyar, Senthil Karthikeyan Sundaram, Elizabeth Ashlee Holbrook, Sravanthi Vadlamudi, and Alain April. REquirements TRacing On target (RETRO): improving software maintenance through traceability recovery. *Innovations in Systems and Software Engineering*, 3(3):193–202, 2007.

[HW03] Gregor Hohpe and Bobby Woolf. *Enterprise Integration Patterns*. Addison-Wesley, 2003.

[Kel02] Wolfgang Keller. *Enterprise Application Integration. Erfahrungen aus der Praxis.* Dpunkt, Heidelberg, 2002.

[MvdHW06] Leonardo G. P. Murta, André van der Hoek, and Cláudia Maria Lima Werner. ArchTrace: Policy-Based Support for Managing Evolving Architecture-to-Implementation Traceability Links. In *21st IEEE/ACM International Conference on Automated Software Engineering*, pages 135–144. IEEE Computer Society, 2006.