

Tool integration with consistency relations and their visualisation

René Freude

Daimler Chrysler Research and Technology, Germany
rene.freude@daimlerchrysler.com

Alexander Königs

Real-Time Systems Lab, University of Technology Darmstadt, Germany
alexander.koenigs@es.tu-darmstadt.de

Abstract

One aim of tool integration is designing an integrated development environment that accesses the data of different tools and keeps them consistent throughout a project being considered. Present approaches use a common database (data warehouse, repository) for storing all the tools' data which is unsatisfying. We will present a new approach which leaves the data at the tools being integrated. Our integration is based on reference objects and consistency relations between them. This approach has been implemented in the *ToolNet* project. Furthermore we will present a possibility for visualising the set of reference objects and consistency relations.

Keywords: *tool integration, data consistency, consistency relation, topic maps*

1 Introduction

Software development projects are subdivided into a number of phases. There are lots of different tools specialised in each of these phases. Thus, the data of a project as a whole is distributed over the tools being adopted. Tool integration tries to design an integrated development environment which offers uniform access to the data of the different tools and keeps them consistent.

ECMA TR/55 [ECMA] defines a reference model for frameworks of (integrated) software engineering environments. The so called "Toaster Model" shows that data integration is done by using a common database for storing the data of the different tools. This approach has been followed by lots of integration frameworks like PCTE [Pic90] in the 90ies. It suffers from the problem that databases are slow and form a centralised bottleneck in the framework.

In [BCM94] we find other approaches which offer other control-oriented mechanisms for tool integration based on message servers. These approaches lack the possibility to specify functional data dependencies between complex structured documents.

In this paper we describe a new approach which leaves the data at the integrated tools and allows the specification of functional dependencies. This approach has been implemented as part of the *ToolNet* project [ADS02]. The architecture of this project is similar to the architecture of the *(M)OTIF* project. It differs from the latter by not using so called *Semantic Translators* which convert the data

of a tool from the tool-specific format into a canonical format and back again. This canonical format is difficult to define without neglecting important tool-specific detail. Instead of this we use XML-technology [XML] to exchange the tools' data. We have our focus on directly specifying consistency relationships between the data of the tools as they are. If we compare the *ToolNet* project with the *Eclipse* project [ECL] we see that *Eclipse* does not provide data integration. Furthermore the user interfaces of the tools being integrated are reused in the *ToolNet* project instead of being redesigned as done in the *Eclipse* project.

2 Accessing data by reference objects

As explained above we do not want to store the data of all integrated tools in one common database to get integrative access to it. Instead of that we want to leave the data at the tools. Therefore, *reference objects* are introduced for each object in a tool to be able to access the contained data (*Fig. 1*). Furthermore *reference data pools* for each tool-specific data pool are introduced.

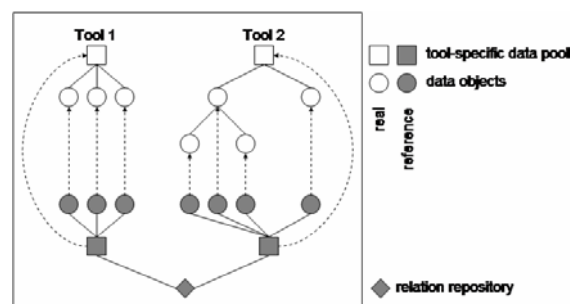


Fig. 1: Real and reference objects

3 Relations and the relation repository

We will now define directed and undirected *consistency relations* on the *reference objects* we just introduced to allow us to keep the data of the whole project consistent throughout all tools. Therefore, each *consistency relation* provides a *consistency condition* which should be fulfilled by the objects being related. We use these *consistency relations* to:

1. navigate from one object to any related object,
2. keep the related objects consistent by propagating changes and
3. detect related objects which violate a given *consistency condition* in order to fix this inconsistency.

Our *relations* are active in the way that they provide the functionality described above themselves.

The collection of instantiated *consistency relations* between objects of the project is stored in the *relation repository*. This *repository* provides additional functionality like finding every object which is related to another by a certain type of *consistency relation* or checking the *conditions* of all instantiated *relations*.

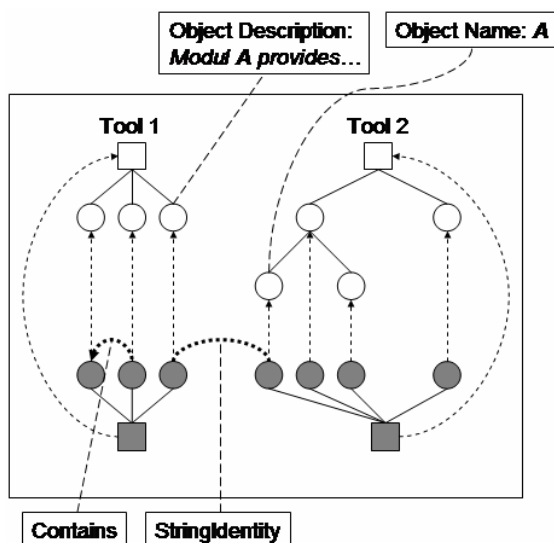


Fig. 2: Different consistency relations in a project

Example: Figure 2 shows two different types of *consistency relations*. *StringIdentity* is an undirected *relation* between objects of tool 1 and objects of tool 2. Its *consistency condition* is that the description of the object of tool 1 refers to the name of the object of tool 2. *Contains* is a directed *relation* between objects of tool 2. It has no *consistency condition*. It solely expresses the fact that the first object “contains” the second. By using this relation we can describe hierarchic structures even if the tool does not support them itself. Let us

now assume that an object of tool 1 is related to an object of tool 2 by an instance of the *StringIdentity relation*. In the case of changing the name of the object of tool 2 this can be automatically propagated to the description of the object of tool 1. Another possibility of using the *StringIdentity relation* is that we can find the object in tool 2 whose name is referred by the description of the object of tool 1.

4 Adding consistency relations to a project

Having defined *consistency relations* on the *reference objects* of the considered tools we integrate the data held by them by creating instances of those *relations* and putting them into the project's *relation repository*.

The creation of instances can be done in two ways:

1. A user manually adds a new instance of a *consistency relation* by choosing the objects he wants to relate and the type of the *relation* he wants to establish. If the chosen objects do not meet the *consistency condition* of the *relation* the user will be warned but not hindered creating the instance.
2. The framework automatically adds new instances of *consistency relations*. To do so it inspects all objects of the tools and tests whether they fulfil the *condition* of any *consistency relation*. If it finds objects which do so it relates them by an instance of the concerned *relation*.

5 Realisation of the approach

Our approach has been realised as a part of the *ToolNet* project [ADS02]. The *ToolNet* project is a tool integration framework. It makes use of our approach to realise *data integration* and *data consistency* throughout the integrated tools.

The framework provides a standardised API (e.g. *ToolAdapter*) to allow the integration of nearly every tool. Although the API is similar to *JMI* [JMI] it does not meet it exactly yet. The *ToolAdapter* encapsulates the access to the data of a tool. For demonstration purposes the framework provides realisations of the *ToolAdapter* for *DOORS*, *Matlab*, and *CTE*. Figure 3 shows how our approach has been embedded in the framework. The corresponding *ToolAdapters* are used to access the data of the tools *DOORS* and *Matlab*. Actually the data of a tool is accessed by requesting an XML-export of the data and working on it. The *VirtualObjectSpace (VOS)* and its XML-representation form the *relation repository* we described above. The *VOS* has information about the defined *consistency relations*. This definition is currently done manually. Furthermore it governs the information about the *reference objects* of the considered tools and created instances of

consistency relations existing between them. This information is saved in an XML-file as well. The *VOS* supports manually creating *relations* between *reference objects*. In addition it has a mechanism which allows the automatic creation of *relations*. This mechanism is batch-oriented and works as follows:

1. request all tools to export their data in XML,
2. inspect all objects for fulfilled *consistency conditions*,
3. create the corresponding *consistency relations*, and
4. export the result as an XML-document.

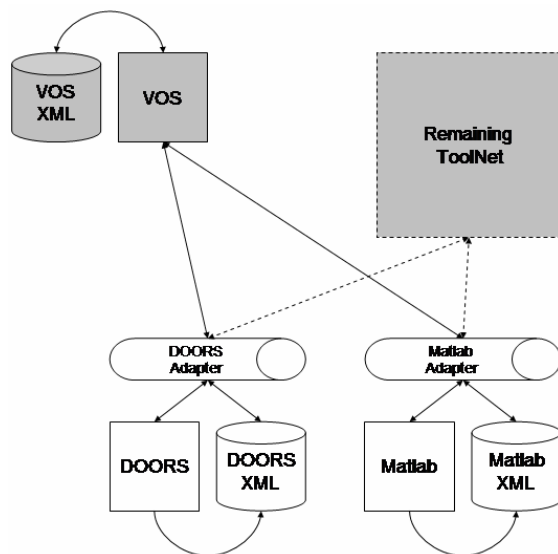


Fig. 3: The relation repository in the ToolNet project

6 Visualisation of the relation repository

As described so far the *relation repository* is one of the central components within the *ToolNet* framework. By managing the relationship between the data of adapted software tools which are used in different fields of system development it constitutes the basis for merging different development domains together into one integrated project.

The visualisation of the *relation repository* aims at assisting a user in analysing the structure of such an integrated project including all dependencies between single domains which are involved. In this process not only aspects of data integration are considered. The information model of the *ToolNet* also covers methodical integration approaches (modelled by typed relations) that have to be made visible and apparent to users. This functionality supports the integrated development and contributes to the implementation of one central requirement for the framework: “Users of an IDE must be able to browse and query the data of a project as if stored in a single database” [ADS02].

7 Use Cases for visualisation

The following use cases underline the benefit of a *visualisation component* for the *relation repository*.

1. **A *ToolNet* user wants to gain a complete overview of the project.** This is especially important for a project manager who has to bring together and co-ordinate individual development teams. The *visualisation component* helps him monitoring the project’s progress and planning the next steps.
2. **Two developers from different development domains want to discuss potential project enhancements.** Both developers are using the *visualisation component* to have a common basis for talking about the project and discussing their ideas.
3. **A user wants to predict or reconstruct effects on the whole project resulting from data changes within a certain participating development tool.** Thus, an unintentional semantic inconsistency concerning the project data resulting from a lack of understanding dependencies must be prevented.
4. **Editing project data by using the *visualisation component*.** Edit, add or delete relations between objects of the project. It is helpful to do that by using the *visualisation component* because the whole development project is clearly presented and accessible.

8 Requirements for the visualisation

Providing a whole-project-view: A user must be able to gain a complete overview of the project information which consists of objects from participating development tools and relations between them. Also the methodical aspects concerning the development process have to be considered and presented.

Compliance with *ToolNet*’s information model: Visualisations have to correspond to the *ToolNet*’s information model which the *relation repository* is an instance of. In order to support a correct mapping the data model for the graph information to be visualised must provide typed and directed edges and typed nodes.

Distributed Visualisation: *ToolNet* is a framework for distributed system developing. Hence the *visualisation component* has to work in a distributed way as well. This is why the *relation repository* must provide an export of its project information to make it accessible for other tools. Moreover, the data model for the graph information has to be exportable to XML for communication purposes.

Standardised Data Model: One aim of the *visualisation component* is to provide exchangeable, multifaceted views on a development project. That leads to a standardised data model which comprises already existing visualisation implementations. Software solutions concerning visualisation issues and graph model management may be integrated more effectively that way.

Interaction with Development Tools: A user may not only browse a project but may also perform interactions with the involved development tools (for example query data or highlight an object in the tool). This facet is directly derived from the already mentioned top-level requirement of a framework for integration of system development tools [ADS02].

9 Realisation of the visualisation

A distributed executable *visualisation component* that can be used on any system within the *ToolNet* has already been realised. The graph information for the development project to be visualised is requested from the *relation repository*. There it is transformed to a data model exportable to XML and transferred to the appropriated *visualisation component's* instance. In the following the most important aspects concerning the implementation of this component will be pointed out:

XML Topic Maps (XTM):

XTM [XTM01] is the chosen data model for managing and interchanging the graph information which is to be visualised. Topic maps in general are an ISO standard [ISOTM] for describing knowledge structures. They provide powerful possibilities for administrating and navigating large networked data. The data model mainly consists of topics (nodes) which are related by associations (edges). Topics and associations are typed. Topics play a specific role in an association which allows modelling directed relations between them. XTM is a product of *TopicMaps.Org*. It is fully compatible with the underlying ISO standard. Topic maps are stored in XML. The usage of XTM is mainly derived from the specified requirements “Standardised Data Model” and “Compliance with *ToolNet's* Information Model”

Graphical Visualisation

A development project may contain a huge amount of highly networked data. This leads to the conclusion that visualising the whole project at the same time could not be arranged in a clear way and therefore would be unpractical. Nevertheless a strategy had to be developed to provide a complete overview of the project information (as specified in the requirements). *Figure 4* gives an impression of the solution found and realised.

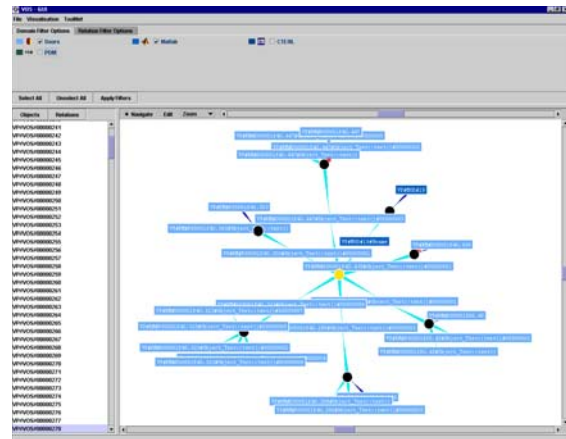


Fig. 4: Graphical visualisation approach

For rendering the graph information that describes an integrated project an already existing solution – *TouchGraph* [TGrph] – was used. Within this visualisation different nodes represent the data of participating development domains; typed and directed edges represent the *consistency relations* between them.

The visualisation only shows a cut-out of the entire information. The user is able to navigate the graph to get the whole-project-view step by step. He may also reset the focus by selecting an item from a list of all objects and relations available.

Filter for graph information:

A user is able to define filters for the visualisation in order to get different views on the project. This feature helps the user to concretise the visualisation graph according to his interests in the information he wants to analyse or modify.

Calling and providing *ToolNet*-functionality

The *visualisation component* does not only present the project information but may also interact with *ToolNet*-participating development tools. This aspect is the basis for further features supporting an integrated development process: A user may query data for an object from the visualisation graph (e.g. descriptions or previews of remote objects). It is also possible to perform functionality calls such as highlighting a certain object in a development tool by performing a request in the *visualisation component* or vice versa highlighting a certain object in the *visualisation component* by performing a request in a tool.

10 Perspective

Planned activities for the development of the *relation repository* and its visualisation:

1. Currently the mechanism for automatic generation of *consistency relations* suffers from the problem that it does not generate new *relations* on an incremental basis. The *VOS* has to recalculate all *consistency relations* which takes a long time.

Additionally it loses any manually created *relations*. In the near future we will offer a more incremental mechanism which allows us to save time and the manually created *relations*.

2. Presently the framework asks the tools for changes which requires a new XML-export of the data. At a later date the tools should announce any changes via an event-based system.
3. Additionally we will use any API provided by the tools to access their data.
4. In future the API of the *ToolNet* will be redefined to meet the *JMI* standard.
5. Furthermore the definition of *consistency relations* is manually done and requires the use of an unintuitive text-based language which is similar to XPath-expressions. We will replace that by specifying *relations* on the basis of *UML-object-diagrams* [WK99] with *OCL* [FS00] and automatically generating the needed code. To do so we will make use of the meta-models of the tools' data on the basis of the *Meta-Object-Facility* [MOF]. The theoretical background for this rule-based graphical approach is formed by triple-graph-grammars [Sch94].
6. Moreover we will enhance the functionality of the *visualisation component* by adding the possibility of editing a tool's data within it.
7. Finally, additional visualisations will be implemented (e.g. a textual representation).

11 References

- [ADS02] Altheide, Dörr, Schürr, *Requirements to a Framework for sustainable Integration of System Development Tools*, in: Stoewer, Garnier (eds.), *Proc. of the 3rd European Systems Engineering Conference*, AFIS (2002), 53-57
- [BCM94] Brown, Carney, Morris, Smith, Zarella, *Principles of CASE Tool Integration*, Oxford Univ. Press, New York (1994)
- [ECL] IBM, *eclipse project*, <http://www.eclipse.org/eclipse/index.html>
- [ECMA] ECMA International, *Reference Model for Frameworks of Software Engineering Environments*, <http://www.ecma-international.org/publications/techreports/E-TR-055.HTM>
- [FS00] Fowler, Scott, *UML distilled*, Addison-Wesley, 2000
- [ISOTM] ISO, *ISO/IEC 13250 Topic Maps*, 1999
- [JMI] Sun, *Java Metadata Interface (JMI) Specification*, <http://java.sun.com/products/jmi/>
- [MOF] OMG, *Meta-Object Facility ((MOF)), Version 1.4*, <http://www.omg.org/technology/documents/formal/mof.htm>
- [Pic90] Picard, *SFINX: Tool Integration in a PCTE based Software Factory*, in: Mahavji, Schäfer, Webe (eds.), *SD&FI – Proc. 1st Int. Conf. On System Development Environments & Factories*, Pitman, London (1990), 219-228
- [Sch94] Schürr, *Specification of graph translators with triple graph grammars*, in: Mayr, Schmidt (eds.), *Proc. WG '94 Workshop on Graph-Theoretic Concepts in Computer Science*, LNCS 903, Springer, Herrsching (1994), 151-163
- [TGrph] Touchgraph LLC, *Touchgraph Technical Overview*, http://www.touchgraph.com/tech_overview.html
- [WK99] Warmer, Kleppe, *The Object Constraint Language: Precise Modeling with UML*, Addison-Wesley, 1999
- [XML] W3C, *Extensible Markup Language (XML) 1.0 (Second Edition)*, <http://www.w3.org/TR/REC-xml>
- [XTM01] Steve Pepper, Graham Moore, *XML Topic Maps(XTM) 1.0*, TopicMaps.Org (2001)