

Requirements to a Framework for Sustainable Integration of System Development Tools

Frank Altheide, Dr. Heiko Dörr
DaimlerChrysler Research and
Technology
{Frank.Altheide, Heiko Dörr}
@DaimlerChrysler.com

Prof. Dr. Andy Schürr
University of the Federal Armed Forces,
Munich
Andy.Schuerr@UniBW-Muenchen.de

Abstract. Tool integration is still lacking appropriate solutions. For an integration project at DaimlerChrysler a number of requirements have been identified which shall lead to a sustainable integration framework being open to further evolution. The requirements are based on a number of principles and cover several levels: user, system, architectural, and realisation requirements. Having these requirements in mind, a first prototype has been successfully developed which is open to further evolution.

INTRODUCTION

Many problems in the development of (embedded) system software are currently caused by strongly decoupled development strands which discourage co-operation between the engaged development teams. This separation leads to higher costs due to long feedback cycles and investments bound by tool-related development results. Quality is reduced by gaps within the used development processes and low interaction between development disciplines.

These problems cannot be solved without integrating actually used engineering tools into one system which is able to assist its users to keep the data stored in different tools in a consistent state and to coordinate their activities. Despite of the fact that Integrated Development Environments (IDEs) and tool integration mechanisms have been a “hot” research topic for more than 15 years now, practical solutions for building IDEs from existing tools have not been provided yet. Mostly, hand-crafted bilateral integrations have been introduced which support the batch-oriented translation of data exported by a tool A into a format needed by another tool B. These solutions are hard to maintain and therefore lasting just a short time. In an integrated project of DaimlerChrysler and EADS, a number of requirements have been identified which shall lead to a sustainable integration framework, called *ToolNet*.

Related work. Earlier work in this area goes back to the 80ies and resulted first in the development of so-called message servers. Typical examples of this kind are the academic ancestor of all message servers, called Field [Rei90], and commercial products like HP-SoftBench [Fro90] or ToolTalk

[Fra91]. These systems offered *control-oriented mechanisms* for the integration of already existing tools, but were not very helpful for maintaining dependencies between documents (development objects) manipulated by a regarded set of tools. They are nowadays replaced by well-known middleware technologies such as (D)COM, CORBA or SOAP [OH98, GS02], which still suffer from the same drawbacks.

Other tool integration projects followed a more *data-oriented* approach. They integrate tools by constructing so-called compatibility maps [Gar88] for a common database or views [Schie93] onto a common database. These approaches are useful as long as all considered tools store their data in one common database (repository). They provide only rather limited support for tracing and maintaining dependencies between development objects stored in different tools with often rather different syntax and semantics.

Books like [BCM94, Nag96, SB93] summarize the state-of-the-art of various tool integration techniques developed in the 90ies. They do not yet take recent standards like XML or UML for data exchange or modeling purposes into account and lack any pragmatic approaches for generating extendible data integration and transformation services from high-level descriptions.

Principles of a systematic tool integration. For the development of a sustainable integration framework, a structured and clear methodology based on generative programming techniques and new data modeling and exchange standards must be applied which combines the *service-oriented* tool integration approach of [BCM94] with *meta-model and rule-based* data integration techniques of [Nag96]. Services are partitioned into engineering (modeling, simulation, testing, ...), management (requirements tracing, configuration, ...), support (printing, navigation, ...), and framework services (messaging, locking, ...). A specific development activity uses services which may be realized by co-operation and integration of several tools.

In the following, the systematic, requirements-based approach to tool integration is exemplified by its application to a first functional prototype.

USER REQUIREMENTS

Capability. The major user requirement for our integration framework is directed to the core task of an IDE, the integration of development tools. From a user's point of view, an IDE must exchange and explore data between tools, maintain consistency on the data, and support process integration, all with a high degree of automation.

Flexibility of an integration has high priority and comes with several facets. The market of (software) engineering tools is rapidly evolving. Tools come and go and yesterdays market leaders may soon be replaced by their competitors. Thus tools with similar functionality must be exchangeable within an integrated development environment to cope with these changes. Besides total tool exchange minor changes must be easily realised to deal with migration to new releases.

Extensibility is a further major user requirement to an integrated development environment. This requirement draws its motivation from several sources. First, the roll-out of pre-configured IDEs becomes easier if these IDEs come in smaller units. Furthermore, the learning curve of users is smoother when the changes to their ordinary development environment stay below a certain limit. Finally, an easily extendible IDE can be adjusted stepwise to the needs of a certain development methodology and process.

Primacy of standard tools. Nevertheless, tool integration must be the integration of currently used standard tools, even if these tools do not provide appropriate APIs for their "a-posteriori" integration into an IDE. Often, one tries to replace functionality provided by standard tools with self-made solutions, which offer the needed services for tight integration purposes. These so-called "a-priori" tool integration projects, in the long run, cannot compete with the pace of the evolution of commercial stand-alone tools. Moreover, the investments in and the familiarity with standard tools are essential assets of any development department. Therefore, functionality which is built into off-the-shelf development tools must not be replaced by a solution to the integration of tools. In fact, the functionality of an integration solution must concentrate on the very integration tasks.

SYSTEM REQUIREMENTS

The systems requirements level provides the needed input for the design of the system architecture of the integration framework ToolNet. It identifies the major components which comprise an IDE and states their roles using the terminology (but not the techniques) of the famous ECMA toaster model developed in the European Software Factory Project ESF [FO90].

From the last user requirement listed above

follows that the essential components of an IDE are the tools which have to be integrated. Any other component must serve for integration purposes. In the following we identify the required integration service components and derive the requirements for them.

Data integration. Data dependencies are an essential base for tool integration. Without coincidences between development data held in distinct tools any integration makes no sense. These coincidences are constituted either by true redundancy of data (e.g. the same data are held in different tools) or set up by the particular development methodology (which states for instance that for any requirement there must be a number of tests). In any case, potential data dependencies must be identified in the configuration of an IDE. The results are to be documented as a global information model which documents all known kinds of dependencies or consistency relations between the data of regarded tools. Based on such an intensional definition of data dependencies, integration tools can be built which create extensional (explicit) representations of integration relations and use these relations for data navigation and update propagation purposes across tool boundaries. Since there are many different kinds of dependencies, which have to be treated differently at run-time, the created integration relations must be typed.

Therefore, one of the core services of an IDE is a repository which identifies and stores different types of integration relations between development objects (data) of its tool components.

As a consequence of the primacy of standard tools, which have to be used as they are, the integration data repository must not store complete copies of tool data for this purpose as e.g. proposed in [Pic90] for the tool integration DBMS PCTE. Exchange of tools would then be rather impossible. In fact, development data must be held in the primary development tools, and the integration data repository may only store placeholders (Virtual Objects) for the sources and targets of its integration relations. Hence, to be integrated tools must provide access to their development data via standard interfaces. These interfaces should offer services for (1) exporting and importing data fragments of any needed granularity, (2) the execution of in-place queries and updates, and (3) the registration of data-update events which are needed for tight integration purposes. For the sake of direct and proper access of all development objects unique and global identifiers must be introduced into the IDE. These global identifiers are e.g. needed for the manipulation of virtual objects in the central integration data repository.

Control integration. Within the category of control integration falls a large amount of

functionality of an IDE. On the lowest level any dynamic use of data dependencies can be subsumed under this category. Examples are for instance exchange or propagation of data, highlighting and processing recognized inconsistencies, etc. On higher levels, control integration offers its users additional help for following well-defined development processes, keeping track of deadlines, reestablishing consistency relations between development objects when required, and so forth.

Most of the IDE services mentioned above, which belong to the category of control integration, are based on the interactive use of stored integration relations. Relations between development objects must be displayed as an additional information provided for all participating objects. Users of an IDE must be able to browse and query the data of a project as if stored in a single database, despite of the fact that the regarded development objects are stored in and displayed by different tools.

This functionality can be improved, if related objects are not only exposed, but in addition the focus is set to the related objects in the appropriate tool. Using this navigation functionality the developer may also query the related object in its primary environment using the native user interfaces of the integrated engineering tools. If necessary, required modifications of regarded development objects may then be executed in place.

Presentation integration. To fully introduce a navigation service within an IDE, appropriate presentation functionality must be provided by all involved engineering tools. Any tool which takes part in the IDE must implement a basic interface for display of and navigation to related objects as already explained above. Additionally, stand-alone viewers must enable navigation between and display of “Ersatz”-representations of development data even if the owner tool of a regarded development object is temporarily not available (for a certain group users or an a specific platform).

Non-functional requirements target at flexibility and extensibility. For instance no monolithic integration of tools but a framework of co-operating components must be provided. Furthermore, the integration functionality of an IDE must be generated and highly configurable. Thus, a specification approach based on UML class diagrams for data modeling purposes and OCL constraints for the definition of integration relations is needed. From these specifications the scenario-specific pieces of code should be generated, which adapt the integration framework to specific tools and valid data consistency relations.

ARCHITECTURAL REQUIREMENTS

This section introduces the core components that

build the ToolNet framework. As the user and the system requirements point out, it should be possible to model relations between data of the to be integrated development tools. This leads to a component, that serves as a relations repository.

In order to avoid redundancy/replication of data, this component does not replicate data models, but introduces references to development objects in the form of Virtual Objects. The component, that is called Virtual Object Space (VOS) accordingly, stores n-ary relations defined on these virtual objects.

For the sake of extensibility the mapping between virtual objects and development objects is delegated to adapters that have specific knowledge of the domain, they were configured for. Each development tool has its own adapter that stores information about the tool’s role in ToolNet and the capabilities of the tool, e.g. if it is primarily a viewer or if it can be used to modify the regarded data. On a superficial level all needed tool adapters use the same data access interface that is derived from the interfaces of DOM-oriented XML parsers to their internal syntax trees [McL01]. The adapters either access XML export copies of development objects if the owner tools of the needed data fragments are not available, they forward queries and updates to the responsible tool otherwise.

For the first prototype, the presentation of all regarded development objects is done by general purpose viewers. One presents XML data in form of a tree view and was written from scratch for the project. The other is capable of presenting content data in Scalable Vector Graphics (SVG) format [Ei02]. It is built on top of the CSIRO SVG toolkit, but many extensions were necessary.

Both viewers provide the same basic functionality wrt. the integration framework. This functionality is modeled as the interface of Fig. 1 which must be implemented by any other presentation component.

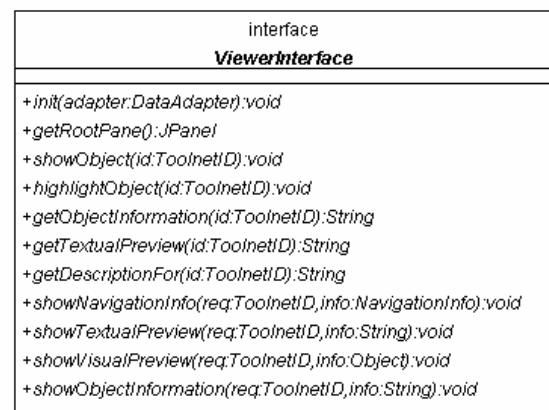


Figure 1: Obligatory interface for ToolNet viewers

In order to support the top level scenario, every viewer, that is part of the ToolNet framework must

be able to handle requests for showing/highlighting objects. If these functions are implemented the viewer can serve as a presentation service in the ToolNet framework.

All components described above must use a general messaging service for all (asynchronous as well as synchronous) communication purposes. Despite of the fact that the first ToolNet prototype is not a really distributed IDE, a communication service component is useful in order to be able to use different middleware technologies in the future based on broadcasting techniques of message servers or point-to-point communications realized via standard “remote-procedure-call” technology.

REALISATION REQUIREMENTS

State-of-the-art technologies for component based development must be applied. The specification of the components, their interfaces, and the integrated data uses UML. Data are represented and transformed by XML or XSLT respectively. The components themselves are realised in Java.

The specification layer indicates the specific properties of the required integration instance. From the specification, relation types as well as patterns for the automated generation of links should be derived. These patterns can then be applied to specific development data to compute links between development data.

CURRENT PROTOTYPE

The currently developed ToolNet prototype is the needed “proof of concept” for first the integration services and components, second the types of relations, and third distributed navigation. It forced us to define the framework architecture more precisely and to develop first versions of its core components.

The resulting prototype integrates two typical development tools. They have been selected based on interviews with developers of various business units. Furthermore, they represent essential development activities in system development. The requirements management tool Doors¹ and the modelling tool Simulink/Stateflow² were chosen.

The prototypical integration has been evaluated by application to real project data. A number of relations between development data have been specified. For instance, the relation “IDENTITY” links multiple occurrences of a single entity which have identical denominators together. Other types of relations strongly base on certain methodologies of the regarded engineering process. The relation “REALISES” makes e.g. use of the methodological approach that in a state-oriented requirements specification each

requirement object is realised by a state. Hence, states and requirements can be linked automatically if the developers of requirements documents use appropriate patterns, i.e. reference states of Simulink/Stateflow models in a certain way.

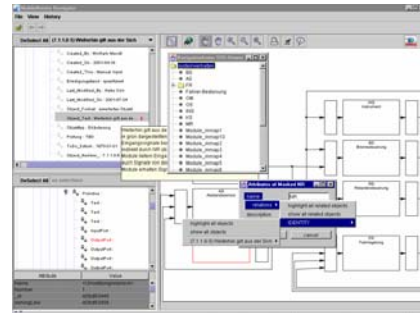


Figure 2: Screenshot of the first ToolNet prototype

The screenshot of the prototype shows a typical use case of ToolNet: a user explores the current state of a development project by navigating along relations set up between requirements and model. The screen is divided into three main views. The left third of the screen is shared by two instances of the XML viewer, the upper showing the requirements from the Doors database and the lower one showing the XML export of the Simulink/StateFlow model. The right hand side is filled by the SVG viewer, showing a sub-model of the Simulink/Stateflow model.

The user has selected an object in the SVG viewer and then requested to see all available navigation information for the selected object. A context menu is presented which allows one to select one of the related target objects. These targets are endpoints of relations that are attached to the selected object. They can either point to requirements, that are realized by this object, or to other objects in the Simulink/Stateflow model that are used by this object. Upon selecting a target, the corresponding viewer will react by showing or highlighting the target.

Architecture of the current prototype.

The architectural requirements introduce a number of basic components which have been realized for the current prototype.

The messaging mechanism offers a publish/subscribe type of communication. Each component uses a sub-component (*plug-in*) to wrap the messages that can be exchanged. This enables the framework to be open to adopt Java RMI or SOAP as communication methods.

The *VOS* is used as a server that handles requests about relations and related objects. Its functionality is encapsulated by a component called *NavigationRequestBroker* that also handles complex navigation operations between an arbitrary number of viewer instances. It is also

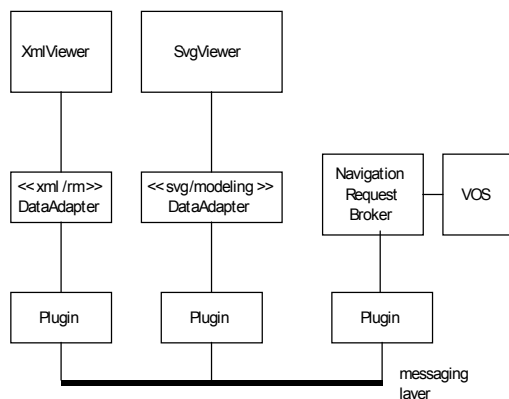
¹ <http://www.telelogic.com/products/doorsers/doors>

² <http://www.mathworks.com/products/simulink/>

responsible to manage change/delete operations that require co-ordination of the active viewers.

Fig. 3 shows the runtime architecture of the prototype instance corresponding to the screendump of Fig. 2. The potential extension by the actual development tools Doors and Simulink is depicted as dashed lines.

The architecture of Fig. 3 hides the fact that the appropriate data are provided in advance. Currently, export filters are used to dump project information out of Doors and Simulink into XML-files. These files then are processed by XML parsers for generating the integration relations stored in VOS. Also, these files are presented by



the viewers.

Figure 3: Run-time architecture

Experiences with the prototype have shown that the integration components work properly, the framework architecture is appropriate, and the component interfaces are stable. Further prerequisites for a successful integration emerged. They are not directed towards the integration framework itself, but to the specific methodology applied within the development project. For instance the development activities must be methodologically integrated to gain a powerful integration, and relations between development data must be systematically specified to be exploited.

CONCLUSION

The development of the first ToolNet prototype has proven the consistency of the proposed requirements for a sustainable tool integration framework. The real power of the framework and hence the appropriateness of the requirements will be proven by further development which stretches out over several dimensions. New project instances may be integrated reusing the current prototype. New development domains may be added based on the current services. New services may be realised as there are change propagation or integrated reporting.

The requirements discussed in this paper to a framework for sustainable tool integration helped us to successfully control the development of the

current prototype and will direct further stages of prototype development.

References

- [BCM94] Brown, A. W., Carney, D.J., Morris, E.J., Smith, D.B., and Zarella, P.F., *Principles of CASE Tool Integration*. Oxford Univ. Press, New York (1994)
- [CL93] Casais, E. and Lewerentz, C. (Eds.): *Issues in Tools' Integration*, STONE Project Monograph, Karlsruhe (1993)
- [Ei02] Eisenberg, J.D.: *SVG Essentials*, O'Reilly & Ass., (2002)
- [Fo90] Fernström, C. and Ohlsson, L.: The ESF Vision of a Software Factory, in: [MSW90], 91-100 (1990)
- [Fra90] Frankel, B.: *The ToolTalk Service*, Sun Microsystems Inc. Mountain View, California (1991)
- [Fro90] Fromme, B.D.: *HP Encapsulator, Bridging the Generation Gap*, Hewlett-Packard Journal, 59-68 (June 1990)
- [Gar88] Garlan, D.: *Views for Tools in Integrated Environments*, Doct. Diss., Computer Science Department, Carnegie Mellon University, Pittsburgh (1988)
- [GS02] Graham, S. and Simeonov, S.: *Building Web Services with SOAP, XML and UDDI*, Sams, (2002)
- [McL01] McLaughlin, B.: *Java and XML*, O'Reilly UK, (2001)
- [MSW90] Madhavji, N.H., Schäfer, W., and Webe, H. (Eds.): *SD&F1 – Proc. 1st Int. Conf. On System Development Environments & Factories*, Pitman, London (1990)
- [Nag96] Nagl, M. (Ed.): *Building Thightly Integrated Software Development Environments: The IPSEN Approach*, LNCS 1170, Springer Verlag, Berlin (1996)
- [OH98] Orfali, R. and Harkey, D.: *Client/Server Programming with Java and CORBA*, John Wiley & Sons, Chichester (1998)
- [Pic90] Picard, P.: SFINX: Tool Integration in a PCTE based Software Factory, in: [MSW90], 219-228 (1990)
- [Rei90] Reiss, St.: *Interacting with the FIELD Environment*, in: *Software Practice and Experience* 20, S1, 89-115, John Wiley & Sons, New York (1990)
- [Schi93] Schiefer, B.: *Supporting Integration and Evolution with Object-Oriented Views*, in: [CL93], 129-149 (1993)
- [SB93] Schefstrom, D. and van den Broek, G.: *Tool Integration*, John Wiley & Sons, Chichester (1993)

