Olli Wirpi

SEAMLESS INTEGRATION OF METAEDIT+ AND ECLIPSE TO COMBINE MODELING AND CODING

Master's Thesis

in Information Technology

(Software engineering)

10/18/2012

University of Jyväskylä

Department of Mathematical Information Technology

Author: Olli Wirpi

Contact information: olli.wirpi@jyu.fi

Title: Seamless Integration of MetaEdit+ and to combine modeling and coding

Project: Master’s Thesis in Information Technology (Software engineering)

Page count: 32

Abstract: Tool integration tries to help users by automating processes in software development. In literature integration is mostly shown from technical point of view leaving user related issues outside. In this paper tool integration is looked also from the perspective of software engineering process and the user involved in the process. Another aspect will be integrating design models and code. A case of tool integration will be presented. The first tool is MetaEdit+ which is a Domain-Specific Modeling environment. The other tool is Eclipse which is used for traditional software engineering. The integration aims to combine the modeling and coding activities which together support metamodelers and modelers in their work.

Keywords: CASE Tools, Domain-Specific Language, Eclipse, MetaEdit+, Modeling, Software Engineering Process, Tool Integration

Contents

1 Introduction

Software development environments provide software engineers tools for different phases in software development cycle. They are reported to improve software development productivity by automating certain work phases and reducing the amount of errors during the development cycle. Growing number of different tools in software development industry has brought a need of integration for tools used for certain work phase in software development process.

Tool integration can be seen in different ways. For one, it means shared resources of two or more tools, on the other hand it might refer to two applications sharing the same user interface and operational behavior. It can also refer to tool extension that integrates some functionality (e.g. modeling) in an existing software development platform.

In this article software engineering tool integration is reviewed. More specific, integration between MetaEdit+, tool for domain-specific modeling and Eclipse is considered. The main research problem deals with software engineering tool integration. How and on which levels tools are integrated. This is not limited to the technical aspect of the integration solution. Also the process in which tools are used is under review. Another perspective is integrating models and code. The implementation integrates models and code that are being manipulated in these tools.

Section 2 reviews the computer aided software engineering (CASE) tools, models in software engineering and domain-specific modeling in particular. The modeling environment MetaEdit+ and integrated development environment Eclipse are also looked over. Section 3 reviews the history and current situation of tool integration. Section 4 describes the implemented integration for MetaEdit+ domain-specific modeling environment and Eclipse. In Section 5, the related work to this subject is reviewed. Section 6 contains the conclusion.

2 Background

This chapter reviews the history of software engineering tools. Also the backgrounds of modeling and traditional software engineering and the tools related to them and this work are introduced.

2.1 CASE tools

The different approaches and activities in software development lifecycles are supported with different tools. They help developers with automatic activities and provide information about the software that is being developed. The tools are traditionally called Computer-Aided Software Engineering (CASE) tools (Sommerville, 2008; Brown et. al., 1992). History of CASE tools begins from 1980's and they were used in 1990's for traditional software engineering. CASE tools are defined by Wasserman (1990) as "tools that address different aspects of the development process." Sommerville describes CASE tools in more detail: CASE tools support different development activities like requirements engineering, design, program development, and testing. He also mentions that CASE tools provide automation and information about the development process and the software (Sommerville 2008).

The history of tools used for software engineering activities starts from 1970 (Gruhn, 2002). At that time, tools supported single activities related to programming such as text editing, code compiling or debugging. The applications used for these activities were stand-alone and isolated from each other. They didn't provide any ready-to-use integration to other tools. Software developers had to build their own usage conventions for the tools they used. This was done with piping and redirecting tool outputs as inputs to other tools (Ossher et al, 2000).

The use of isolated tools led to tool integration. First, tools were integrated to use shared data (Gruhn, 2002). Later, user interface integration was also provided as part of integration. The first tools that combined and automated different activities were known as programming support environments (PSE). Ossher mentions *Pan* and *Gandalf* for the first

PSE’s (Ossher et al, 2000). As the PSE name stands for, environment combined tools that supported coding activities but left other software engineering activities outside.

The next generation of environments included tools that supported activities from wider range than coding. Software engineering environments (SEE) included tools that provided support for design and testing (Ossher et al, 2000). Compared to the first generation environments, PSE’s, SEE’s provided more comprehensive guidance for the developers through supported process and were able to automate certain activities. However, their drawback was tight binding to certain software process they supported leaving no choice for the people using them to extend or change that process (Gruhn, 2002).

Strict process consistency of the second generation software engineering tools raised the software process to general discussion and people started to pay attention to the process itself. Attention was paid on whether the tools and environments should have built tightly to support only certain process or if developers should be able to choose the best suiting tools for their needs and routines. Usually software processes exists with different levels of rigidity and require different tools that fit in process (Gruhn, 2002), so the most important features of software engineering environments should be availability and flexibility for different types of processes.

In more general, Ossher et al (2000) describe the problems of tools used in software engineering. Mostly, tools are designed for one context. For example, they support particular programming language or database; they run on particular platform or use only particular compilers. They might also require some other tools to work properly. All these leave the choices and wishes of software engineers working routines outside. A major issue is to find or provide tools and environments that are enough adaptable and flexible for different contexts. Essentially, there are two options that serve the problem area: tools that are enough basic and general to be used, for example text editors like *vi* or *emacs*, or environments that support plug-ins which extend the environments capabilities for different purposes.

2.2 Eclipse

These days most used environments for software engineering are integrated development environments (IDE). They aim to provide full support through the software development process providing tools for designing, coding, debugging and, testing. Eclipse[1] is an open source IDE that has wide range of users. Mostly, it is being used for developing software with Java, but several other languages are supported as well. It is extensible with different plug-ins and provides environment for developing plug-ins also. Eclipse runs in all of the most common operating systems.

Eclipse environment consists of components that provide services for the user. They all operate on top of the Eclipse core that provides platform runtime and resource management for other components. Eclipse software development kit (SDK) provides *Eclipse plug-in development environment*[2] (PDE), which is toolkit for developing plug-ins for Eclipse platform. The plug-ins are written in Java.

Eclipse projects include frameworks for modeling and graphical editing of models. Together, they can be used as a graphical modeling environment that supports building domain specific language and tool. Eclipse Modeling Framework Project[3] (EMF) develops modeling framework and code generation environment for Eclipse. Eclipse Graphical Editing Framework[4] (GEF) provides graphical editors and tools for building and using graphical modeling languages. The third component is Eclipse Graphical Modeling Framework[5] (GMF) that makes use of both EMF and GEF, provides application framework for building graphical editors and tools in Eclipse with domain modeling and code generation capabilities.

[1] http://www.eclipse.org/
[2] http://www.eclipse.org/pde/
[3] http://www.eclipse.org/modeling/emf/
[4] http://www.eclipse.org/gef/
[5] http://www.eclipse.org/modeling/gmp/

2.3 Models in software development

When object-oriented programming languages increased their popularity in software development in late 1990’s and around 2000, Unified Modeling Language (UML) became the standard of modeling language for object-oriented software engineering and has been it up to these days. It supports modeling software’s structure and behavior, and it can be used for both design and documentation of software. Software engineering processes mostly consist of requirements engineering, design, development, testing, and deployment. In all these phases UML can be used for describing parts of the software for the particular practice of software engineering process. For example, at the development phase UML models give guidelines for the developers of how the software should be implemented.

Although UML stands for the standard of modeling language in software engineering it has limited capabilities of utilizing models. UML can hold the information of class structure and classes itself, from which class skeletons can be generated to source code. However, all the business logic needs to be written manually afterwards which, at the same time, rules out the possibility of developing the models and generating the class skeletons again without losing the manually written code inside previously generated classes (Selic, 2003).

2.3.1 Model-driven development

Model-driven development aims to utilize models for software implementation. Models can be represented at different abstraction levels lower levels being more specific to the application being developed. Model-driven development (MDD) primary focus is on model development instead of programs. The models are expressed using real world problem domain concepts which are not bound to the implementation technology. The higher abstraction level and bindings to the domain area dissociates the models from e.g. UML models used in design phase of software development. Still, MDD applies for any kind of modeling language, not necessarily for example full code generation from design models. (Selic 2003)

2.3.2 MDD Standards

Model Driven Architecture[6] (MDA) is a standard for MDD of Object Management Group (OMG). The key standard of MDA is Unified Modeling Language (UML) which is used for model-based software design. Other standard technologies included in MDA are Meta-Object Facility (MOF) to express the models on a metalevel, XML Metadata Interchange (XMI) to enable metamodel and model transformation, Common Warehouse MetaModel (CWM) for data modeling, and Common Object Request Broker Architecture (CORBA) that enables distributed software components to work together.

2.3.3 DSM

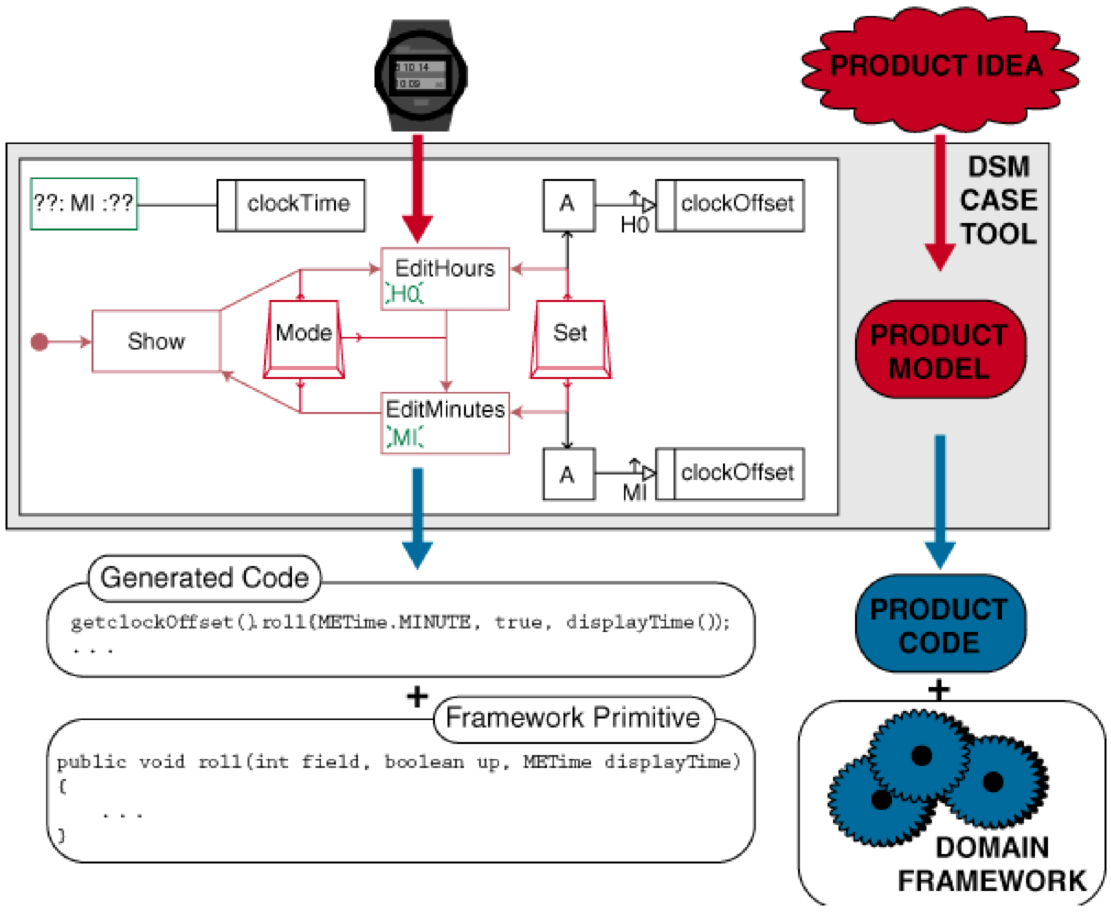
Domain-specific modeling is an evolution of MDD. It utilizes the MDD approach, aiming to full code generation based on the software design models of the problem area. The models are based on metamodel, a modeling language, which is designed for the narrow domain for which the software is being developed. Models are views to the software being developed that follow the rules and specification of the metamodel behind them. The models and their elements refer usually more to the real world concepts rather than the code itself. A DSM solution includes also a code generator designed for the same domain. This enables the development to be done by modeling the application and generating the code from the models. A domain-specific modeling language usually covers only a narrow class of problems, rather than trying to be a general solution for all problems. By being designed for a specific task, it can offer the best possible support for that task. (Kelly & Tolvanen 2008)

To design software with models a metamodel is needed. The metamodel is a definition of the modeling language. It defines the concepts of language that are related to the domain area. The properties and bindings of the particular domain concepts together define the syntax and semantics of that particular language. The rules defined in the metamodel, prevents the modelers to produce illegal models. By specifying rules to the metamodel itself, the need for error checking’s in generation phase, or testing the output, decreases. (Kelly & Tolvanen 2008)

[6] http://www.omg.org/mda/

The third part of the DSM solution is the generator. Basically a generator says how to crawl through the models and what to output. Like the metamodel, the generator is always a domain-specific and cannot be used as a general solution. The generator is always specific to the models it's used with and certain programming language and platform. Depending on the domain area, the metamodeler can define the code generator to output the code that is required.

Domain specific models capture usually the behavioral parts of the software. Besides that a program contains usually static parts, for example user interface components that are part of the program but are written as code only once and used when needed through software. In DSM, this is called framework code. The code generator will not generate the framework code but it makes use of the interfaces that the framework provides. (Pohjonen & Kelly, 2002).



```
getclockOffset().roll(METime.MINUTE, true, displayTime());
...
```

```
public void roll(int field, boolean up, METime displayTime)
{
    ...
}
```

Figure 1 Parts of a DSM project (Pohjonen & Kelly, 2002)

When speaking about domain-specific modeling we refer it to modeling. Although generators and the additional framework code would require continuous development work as well, modeling is referred to development in DSM context. Models can be represented as text, graphical diagrams, matrices or tables. Use of the graphical diagrams, matrices or tables gives the developer a better mapping to the real world domain that the text structure does. In many times the graphical modeling can be easier to work with compared to the textual modeling (Kelly & Tolvanen 2008).

Looking on the history and evolution of software engineering and programming languages, DSM has put the productivity of creating software in new level by providing more efficient and easier way to design software. The problem faced with different tools before DSM, was that the high level modeling language tool had fixed notation that where rarely suitable for the user's needs. The raise of abstraction level is a similar step as programming language taking the place of assembler in 1970's and 1980's, or moving to third-generation programming language in 1990's. The same phenomenon of raising productivity that is seen between assembler level programming and 3GL's is seen also when moving from 3GL to DSM. DSM allows developers to stop working on the previous versions of source code and lets them concentrate fully on the domain-specific modeling language only in similar that was seen when there were no sense to edit the Assembler code generated by a compiler, but C code and then compile it again to generate new Assembler code.

2.3.4 MetaEdit+

MetaEdit+[7] is Domain-Specific Modeling environment. It supports full DSM process from metamodeling to modeling and code generation offering graphical interface to create and edit models. MetaEdit+ runs in all most common operation systems.

Modelers and metamodelers use MetaEdit+ through its user interface, without need for hand-coding. However, for cases where other programs need access its data and operations, MetaEdit+ provides an application programming interface (API). The API is implemented as Web Service that receives SOAP calls from its clients. Clients make use of the API by making remote function calls to the MetaEdit+ API server. This solution

[7] http://www.metacase.com/products.html

enables large range of different programming language to use the MetaEdit+ API. The API functions are described in web service definition language (WDSL) file, an XML description of functions available in the API server. Lots of different languages provide tools for reading a WSDL file and generating language like method stubs for calling the SOAP server as HTTP requests (Kelly 2004). For developers, making the SOAP calls look the same as the normal function calls at code level.

Among the API server, MetaEdit+ provides a way to start new MetaEdit+ instances from command line with different parameters and operations. This functionality is used for example starting MetaEdit+, logging in and opening projects.

3 Tool integration

Tool integration history begins from 1990's decade. Integration was done by building integration for existing, independent tools produced by different vendors and building integration for them (Losavio et. al., 2002). Another approach was to build a framework that integrates tools by providing common data models, shared repositories or message passage system. This kind of integration is presented in (Rangarajan et al, 1997; Sum & Ibold, 1994; Lefering, 1993; Yimin & Horowitz, 1996). The main goal for all of them was to provide better support for the development process that was involving several software engineering tools for different tasks. An example of the integration frameworks is the Portable Common Tools Environment (PCTE) tool interface specification, a standard of European Computer Manufacturer Associations (ECMA). It provides specifications of data management, process control, security, network management, auditing, and accounting (Long & Morris, 1993). An example of using PCTE to integrate tools is presented in (Huw, 1991) paper.

At that time, tool integration had mostly technical aspect under review. Wasserman (1990) defined tool integration in five categories: *platform integration* makes tools to work on same platform. *Presentation integration* is used for providing a common user interface between the tools. *Data integration* shares the same data resource between tools, *control integration* defines roles of tools, where one takes control of another and make use of its functions. *Process integration* provides linkage between the tools and combines two separate tool processes together. In addition to Wasserman's integration dimensions, Thomas and Nejmeh (1992) extend that viewpoint towards software engineering environments in their tool integration study: The main focus on tool integration is on the relationship between tools. The subject of integration is one or more of the Wasserman's definitions but the focus should always be on the relationship between tools. In here, noteworthy is that neither of the studies concentrates on the user's experience or the software engineering process, in which tool integration is built for.

Tool integration is tightly bound to integrating work phases of a process where the tools are used. Former and present, the primary goal is to integrate the activities through the software development process lifecycle. However, the target is not reached by just looking on techniques of how to integrate software's together, although it is still needed as part of designing the integration. Wicks and Dewar (2007) define tool integration to a binding between tools that provides environment that supports part or all of the activities involved in the software engineering process. From technical aspect, the definition is similar to what Wasserman defined almost 20 years before. However, Wicks and Dewar go further by extending their definition from technical point of view: tool integration involves, besides the technical viewpoint, also the social and economic viewpoints that are in many cases undertaken. That includes business decision and goals when considering tool integration and the net values the integration delivers to its users as well. Based on their literature review, research work approaches tool integration mostly from technical view and lack of other perspectives.

The value delivered to users by integration is rarely discussed in academic works although the very first integration solutions made for existing tools dealt with that subject. They are described in Harrison's (Harrison et al, 2000) article: because there were no environments or co-operating tools, users created their own conventions and routines for editors, compilers and other tools to work together with, for example, piping and redirecting. Those basic operations brought value to their users and helped to raise the productivity of certain phases in the development process.

3.1.1 Support of tool integration

The integration seems to be still often weakly supported. Broy (Broy et. al., 2010) reports from embedded industry: "the engineers adopt ad-hoc integration solutions that are far from a disciplined engineering". This happens because tools don't provide proper ways to integrate them. Maalej's (2009) case study shows that there exists no common or unified forms of integration. Integration solutions should support heterogeneity and changes of the surrounding environment.

There is need for common and standardized forms of integration. The integration for different activities is often reached by loose integration between tools and their support for the certain part of software development process. This is because of continuous evolution of the software development process. Processes evolve in unpredictable way due to mix of affection of business processes, new domains, users, hardware and software. All this sets new requirements for tools and environments. They cannot be specific to one aspect or process; instead they need to be adaptable for new contexts (Ossher et. al. 2000).

3.2 Technical views for tool integration

The Wasserman's tool integration dimensions are still useful for looking an integration implementation from technical point of view. They can help developer to collect and see possible solutions and challenges when designing tool integration. However, when designing tool integration, one has to keep in mind that they don't answer to the question how the tool integration will fit into the processes where tools are used. In next sections the data and control levels of integration are introduced since they are the most relevant for the integration work that is introduced in this paper.

3.2.1 Data integration

Data integration focuses on the data handled by tools. It has five properties: *interoperability* defines what data manipulation is needed to share data between tools. *Non-redundancy* identifies data redundancy of the data tools share and use. *Data consistency* indicates how well tools keep data structured. *Data exchange* defines how well the tools understand each other's data or how much work is required to manipulate data produced by one tool for the second tool to be able read and work with that data. *Synchronization* property defines the synchronization level of shared data resource between two tools.

Lewis (1990) divides data integration in three categories. The loosest integration is *data linkage*, which means two data sources used by different tools sharing some kind of common semantics. *Data interchange* means sharing data by transferring it between the tools. This requires some commonly agreed conventions or standard message protocols for both of the tools. *Data sharing* is considered when two tools have access to same data resource. This does not only require ability to handle the also to know the other tools capabilities and to take them into account when operating the shared data resource.

3.2.2 Control integration

Control integration allows tools to communicate directly to each other and let one tool to call operations of the second tool. It has two properties regarding the same thing from two sides. *Provision* defines the level of tool's services that are used by another tool. *Use* defines to what extent a tool uses the services the other tool provides. This requires standard message protocols and interfaces to access tool operations from another (Gautier et. al., 1995).

3.2.3 Synchronous and asynchronous integration

Damm (Damm et al., 2000) concentrates on the process where integration is used. The integration is *synchronous* or *asynchronous*. In synchronous integration tools run simultaneously by changing data concurrently or using replicated data and communicating changes to it. Asynchronous integration means that tools run at different times executing operation that may base on the result of other tool’s operation. Tools are using shared data at different times.

The aspect of synchronous and asynchronous integration includes three characteristics: *coordination*, *collaboration* and *communication*. Coordination includes rules and delegations handling the overall process of integration. Tools collaborate at some level to some artifacts. Those might be, for example, some kind of data models that tools use to execute tasks they support. Communication is handled by a messaging component or in data semantics.

3.3 Integration by extension

In addition for integrating two tools together, another way of tool integration is integration by extension. This means building a new tool for an existing integration environment. Instead of taking existing tools that support certain activities, the wanted feature is implemented as an extension inside an existing tool.

Integration by extension is common in IDEs such as Eclipse and Visual Studio. Both of these IDEs offer framework for adding extensions and include particular framework for building modeling tools. For modeling activities, Eclipse Modeling Framework (EMF)[8] allows to build modeling tools based on models. Different solutions utilize this framework by building their own modeling tools, both textual and graphical. As examples of Eclipse modeling framework, Biermann et al (2009) describe a DSML tool they've implemented on top of EMF. Smith (2010) shows an Eclipse plug-in for Performance Evaluation Process Algebra (PEPA) modeling language. Ye et al (2011) report a workflow modeling tool implemented of top of Eclipse Graphical Modeling Framework (GMF). Dubois et al (2009) describe Eclipse environment called Papyrus which is a modeling tool for requirements engineering.

Another major IDE is the Microsoft's Visual Studio. Similar to Eclipse, it is possible to build tool extension and use them inside the IDE. Like Eclipse, with Visual Studio it is possible to create and develop modeling languages and models. Microsoft DSL Tools is an environment for creating model-based development tools and languages. It operates as part of Microsoft's Visual Studio development environment. It provides tools and editors to build and design domain-specific modeling language in graphical notation and code generators for generating application code from models. DSL Tools is run as part of Visual Studio and cannot be used without it.

3.4 Main features of tool integration revised

In this chapter, the properties of tool integration have been outlined. Basically, tool integration properties can be divided in two main categories. First, tool integration features can be seen rising from the requirements and restrictions of the software engineering process in which the tool integration is being used. Tool integration is always a construction that helps the user to work in a software engineering process. If the integration is strict for a certain process, it is often bound only to that single process leaving other possible use cases outside. On the other hand, too flexible and adaptive integration may became hard to configure and use for most of the users in processes they are involved. Tool integration should be always designed to meet the requirements that are

[8] http://www.eclipse.org/modeling/emf/

rising from the software engineering process and not to force user to adopt parts of the process that he wouldn’t use otherwise.

The second perspective concentrates on the integration from technical point of view. Tool integration should work without any extra functional redundancy or data duplication. This is achieved with clear vision of the process in which the integration is made. The parts of the process are divided for single tools as well as the data that is handled and produced in the process. Some of the features are not measured, but identified to define the characteristics of integration. Tool synchronization defines whether the tools work in a single process as synchronized or if they operate on separate processes as asynchronous.

4 Integration for MetaEdit+ and Eclipse

This chapter describes the implemented integration for MetaEdit+ and Eclipse. First, the requirements and challenges are outlined in Section 4.1. In Section 4.2 the requirements and challenges of the integration are reviewed. The implemented work is looked at Section 4.3 and the overall summary is provided in Section 4.4.

4.1 Requirements and challenges

Integration for MetaEdit+ and Eclipse aims to offer more seamless process for developers that use modeling and traditional software engineering tools. This is achieved in a process, where integrated tools collaborate with each other. A process that involves both of the tools includes modeling activity done in MetaEdit+ and at least some of the activities that are usually carried out with Eclipse or other development environment. This could be coding, debugging or testing. Maybe the most important shared resource between the tools is code, which is generated from design models with MetaEdit+ and manipulated and tested in Eclipse.

By integrating two tools instead of extending one's functionalities the principle of loosely coupled tools is taken into account. Speaking of modeling and coding, Eclipse community has several projects going on that develop modeling frameworks for Eclipse. Basically, modeling activity could be implemented into Eclipse without MetaEdit+. As this would leave all the technical integration issues faced in this work outside, the negative effect would be more restricted to the DSM tool compared to MetaEdit+.

Programming languages and techniques are open for any in MetaEdit+. Eclipse modeling frameworks support Java or XML based mapping languages for generation target format. By designing integration for DSM tool and integrated development environment in a way which allows heterogeneous environments, it is possible to extend the integration to cover other platforms, programming languages, or development environments as well. By using separate domain-specific modeling environment and development environment, the end user is always able to change between tools, techniques, and environments. For example, similar to this solution, there could be also integration for Microsoft's Visual Studio. Being integrated to two different environments the value of separate modeling environment can be seen more clearly.

Looking at the process of DSM, we can see that DSM is not necessary interested of code but models. Code can be regenerated whenever needed so it is not important to maintain or refactor. Why is then integration for integrated development environment and DSM tool needed? Following examples describe cases where integrating modeling and development environment is reasonable.

- The generator builder needs to know the metamodels, models, generators and code. Being integrated to domain-specific development environment, the development environment gives better support for the generator builder to design and test the code generators.
- The domain framework builder prefers development environment where he can host and build the framework code. With integrated modeling environment, the framework builder can always run code generator from modeling environment and test the framework functionality with generated code.
- The generator needs the framework code to debug the generated code to develop the generators.
- Despite using DSM as primary development method, there might still be parts in the applications what are not being implemented at the model level. Those parts need to be written to programming language by hand.
- Many developers are used to doing the development work in IDEs and the integration to Eclipse allows them to keep using the IDE when needed.

Integration that includes more than shared resources between tools requires some kind of interface for one tool to take control of another. By letting one tool to take control of another, the need of switching between tools is reduced. This makes the whole process, where tools are included, clearer. If developer needs to switch between the tools too often, too much effort is put on the process instead of actual work that is done in the process.

Eclipse provides mechanism to extend it internally but does not allow using it from outside. MetaEdit+ provides API server[9] that allows Eclipse plug-in to connect and use functions of MetaEdit+. The API server provides a wide interface for functional commands. To support heterogeneous tools, API server uses SOAP[10] protocol as the standardized message protocol. The SOAP protocol enables a very wide range of languages to implement the SOAP client side (Kelly 2003). However, all the information going from MetaEdit+ to Eclipse, except the return values of calls initiated from Eclipse, need to be transferred in some other way since MetaEdit+ has no capability to send SOAP calls and nor does Eclipse offer a SOAP interface.

4.2 Key requirements for the integration

The engineering process involves domain-specific modeling and traditional software engineering with the tools. Modeling is done with the domain level concepts in MetaEdit+ from which the code is then generated. Often some parts of the program, the framework, are written manually instead of modeling and generating the code (Kelly & Tolvanen 2008). The generated code contains function calls to the framework components. The design and implementation of the framework code is done in Eclipse. The final application that consists of the generated code and the framework code is compiled and run in Eclipse. MetaEdit+ needs to produce the source code in way and in a form that Eclipse accepts and is able to load the code in.

In DSM, the generated code is not versioned, but the models are (Kelly & Tolvanen 2008). Looking of the integration solution this means that always when something needs to be fixed or changed in the modeled part of the software, the code is not touched, but instead models or code generators are and the code is generated again. The regenerated code has to replace or overwrite the old versions of generated code in Eclipse. Although no code is maintained in Eclipse, it is a good place to debug and develop the code that is wanted as output from MetaEdit+ generators.

[9]http://www.metacase.com/support/45/manuals/mwb/Mw-8.html#Heading2326
[10]http://www.w3.org/TR/soap/

Not versioning the source code includes those parts of the software that is generated from design models. The framework code is developed and maintained like in traditional software engineering. Eclipse provides environment and tools to store and develop the framework of the software.

Because of poor external interface, Eclipse is the master tool that utilizes MetaEdit+. The process starts from Eclipse and ends to Eclipse. It gathers the pieces of code (generated and framework), links them to each other, compiles the program and is able to, for example bundle the compiled classes and export a deployable web application package. Eclipse can also provide wide set of other tools that are used for development. For example project build tools or integrated source code version control tools. MetaEdit+ provides modeling and code generation environment, and the interface to utilize MetaEdit+ functions from Eclipse.

The integration for MetaEdit+ has the following objectives:

- Eclipse starts MetaEdit+ on the specified model repository with specified username and password.
- Eclipse shows the MetaEdit+ graphs in a view allowing user to browse them and do some basic operations like editing graph's properties.
- Eclipse allows user to call MetaEdit+ operations for graphs such as creating a new graph.
- The generated source code is imported automatically to Eclipse after MetaEdit+ has generated the code.

The following picture shows process where both MetaEdit+ and Eclipse are involved.

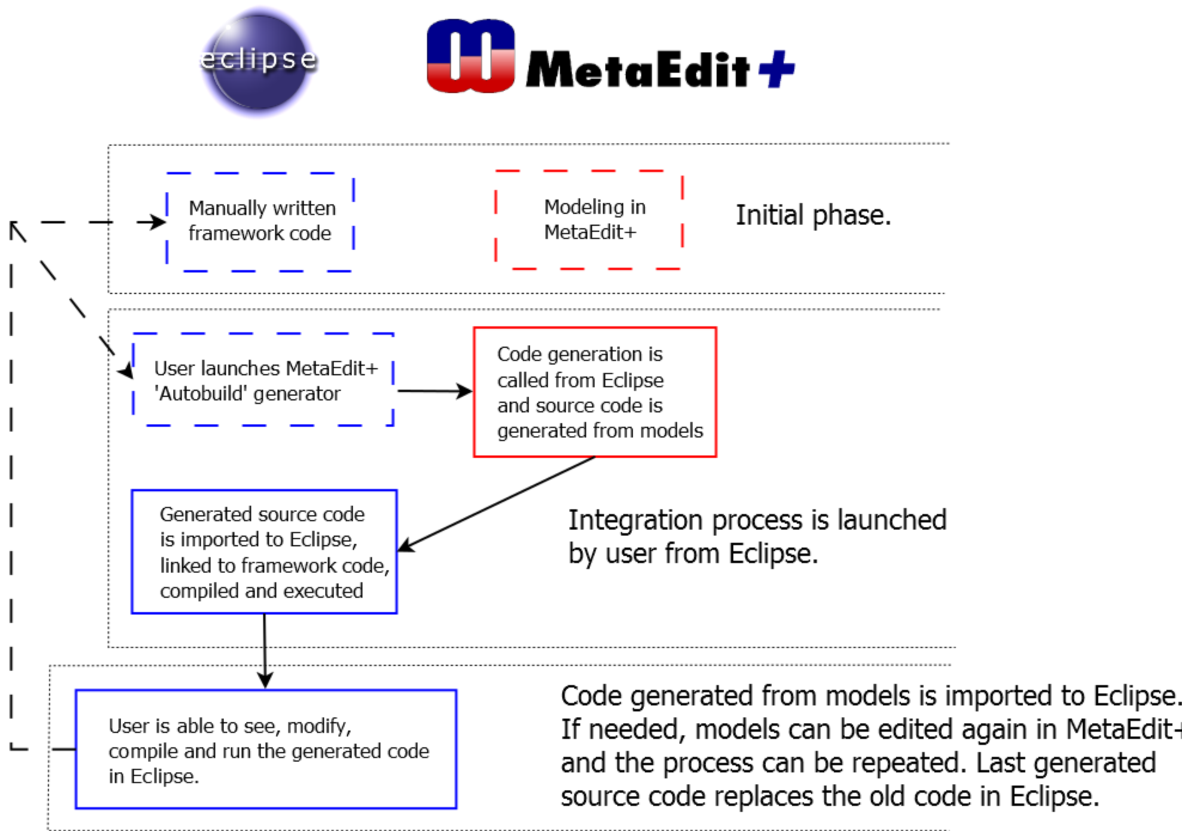
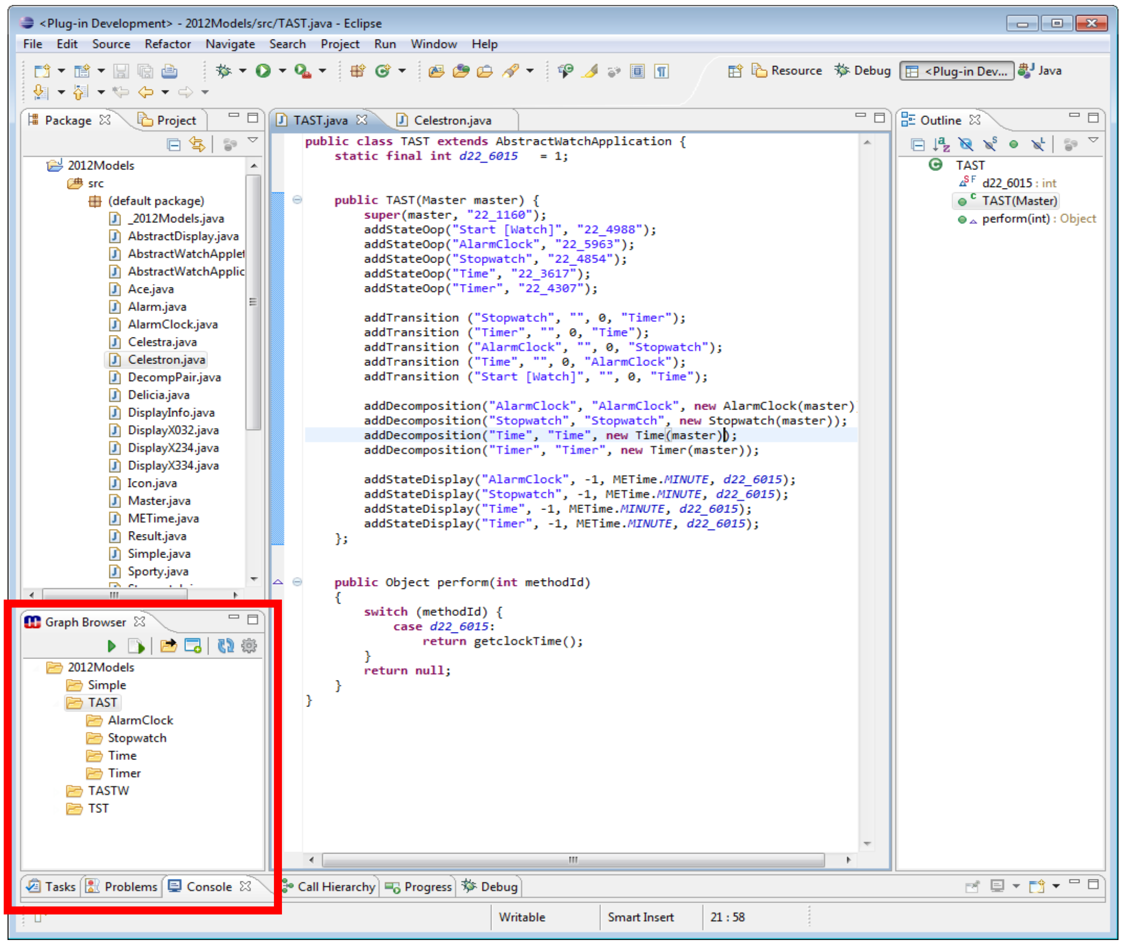

eclipse
MetaEdit+
Manually written framework code
Modeling in MetaEdit+
Initial phase.
User launches MetaEdit+ 'Autobuild' generator
Code generation is called from Eclipse and source code is generated from models
Generated source code is imported to Eclipse, linked to framework code, compiled and executed
Integration process is launched by user from Eclipse.
User is able to see, modify, compile and run the generated code in Eclipse.
Code generated from models is imported to Eclipse. If needed, models can be edited again in MetaEdit+ and the process can be repeated. Last generated source code replaces the old code in Eclipse.

Figure 2: Workflow of the integrated process

The most critical part of the integration is to produce working Java code and to import it to Eclipse. In addition to the source code, Eclipse needs also resources that define its project properties, for example, the used programming language and version, the used version of Java compiler and dependencies for compiling and running the generated source code.

4.3 Implementation

The integration is implemented as an Eclipse plug-in which makes use of MetaEdit+ API server. Some basic part of MetaEdit+ model information and operations are shown and available in Eclipse plug-in which is the main user interface when using the integrated tools together. The way of implementing it that way, Eclipse as the primary UI, was an outcome of investigating the possibilities and limitations of both of tools. Eclipse supports plug-in development but it doesn’t offer any external interface for other tools to utilize it. MetaEdit+ does not support any plug-in development, but it does offer an external interface that other tools may call.



```java
public class TAST extends AbstractWatchApplication {
    static final int d22_6015   = 1;


    public TAST(Master master) {
        super(master, "22_1160");
        addStateOop("Start [Watch]", "22_4988");
        addStateOop("AlarmClock", "22_5963");
        addStateOop("Stopwatch", "22_4854");
        addStateOop("Time", "22_3617");
        addStateOop("Timer", "22_4307");

        addTransition ("Stopwatch", "", 0, "Timer");
        addTransition ("Timer", "", 0, "Time");
        addTransition ("AlarmClock", "", 0, "Stopwatch");
        addTransition ("Time", "", 0, "AlarmClock");
        addTransition ("Start [Watch]", "", 0, "Time");

        addDecomposition("AlarmClock", "AlarmClock", new AlarmClock(master)
        addDecomposition("Stopwatch", "Stopwatch", new Stopwatch(master));
        addDecomposition("Time", "Time", new Time(master));
        addDecomposition("Timer", "Timer", new Timer(master));

        addStateDisplay("AlarmClock", -1, METime.MINUTE, d22_6015);
        addStateDisplay("Stopwatch", -1, METime.MINUTE, d22_6015);
        addStateDisplay("Time", -1, METime.MINUTE, d22_6015);
        addStateDisplay("Timer", -1, METime.MINUTE, d22_6015);
    };


    public Object perform(int methodId)
    {
        switch (methodId) {
            case d22_6015:
                return getclockTime();
        }
        return null;
    }
}
```

Figure 1: Graph tree initialized in Eclipse plug-in for MetaEdit+.

Eclipse plug-in was developed with Java. It uses the Eclipse internal API to integrate itself as an Eclipse plug-in. MetaEdit+ API uses the Web Service standard listening SOAP calls from clients. From developer's point of view, in Java SOAP calls are proxy classes that look like normal Java classes offering operations that are available in MetaEdit+ API server. The proxy classes are basically interfaces to the real operations in API server. The Java proxy class files are generated with standard Java tools from Web Service Definition Language (WSDL) file that is generated from MetaEdit+ API server.

The integration for MetaEdit+ and Eclipse can be divided into four parts: MetaEdit+ initialization, graph representation in Eclipse, graph manipulation, and importing the generated source code from MetaEdit+ to Eclipse.

The initialization opens new MetaEdit+ instance, logs in, opens one or more projects and starts the API server automatically. When an API connection is established, plug-in shows user the graphs in Eclipse and allows to navigate them, make basic use for the graphs, run generators and import the generated source code to Eclipse.

MetaEdit+ is launched from Eclipse with following parameters that user has defined in Eclipse plug-in: username and password that are used for logging in, names of the projects to be opened, port number to be used with the SOAP connection and a command to start the API server in MetaEdit+. If user tries to launch a new MetaEdit+ instance when one with open SOAP connection exists, the existing connection is used and no launching is done. After MetaEdit+ is launched, the graphs of the projects that were opened are shown in Eclipse.

User can explore MetaEdit+ project’s graphs and their hierarchy in Eclipse visually. Working from Eclipse, user is able to open, edit and create new graphs to MetaEdit+ repository using the MetaEdit+ API functions. The changes are updated to a model tree view shown in Eclipse.

Every graph has a name and type name property. A graph might have a subgraph, parent graph or both. This information is used when forming the graph tree. The graph data needs to be manipulated before showing it for user in Eclipse. MetaEdit+ does not organize the graphs as hierarchical tree where one graph is child to another. Instead they are organized as a net. In order to show the graphs in a tree view for Eclipse users, the graph tree is built at runtime.

Showing the graphs in a tree view in Eclipse, user is allowed to do some basic operations for the graphs. First, graphs can be opened from Eclipse in MetaEdit+ graph editor. Although the editor is provided by MetaEdit+, the editing process starts from Eclipse and returns back to it when user is finished. The same convention is used when user wants to edit graph’s properties. The graph property editor is a separate editor from graph editor. The property editor can be opened from Eclipse and when returning back to Eclipse, all the changes made in Eclipse are updated to graph view. The third operation that is applied from Eclipse is creating a new graph. Like the previous commands a dialog for creating a new graph in MetaEdit+ is opened from Eclipse. After successful return from the MetaEdit+ wizard to Eclipse, the tree view is updated with the latest changes.

Maybe the largest part of integration between MetaEdit+ and Eclipse is the generation and importing process. MetaEdit+ generates output, the source code, which can be edited, debugged and ran in Eclipse. First, the generation is started from Eclipse. Eclipse writes its workspace location and name as key-value-pairs in a settings file that is located at MetaEdit+ working directory. The working directory location is read from the launch parameter settings which user has given beforehand in Eclipse plug-in. When generating code for Eclipse, MetaEdit+ read the code output location from the shared settings file.

Eclipse calls MetaEdit+ to start the generator for the top level graph's generator named 'Autobuild'. The generator's name 'Autobuild' is common convention for MetaEdit+ generator that provides fully automatic generation process the produces compiled and running Java program.

The existing settings file with proper values tells MetaEdit+ to generate Eclipse project property files in addition to the Java source files. This integration makes use of previously designed, fully working MetaEdit+ Java generator which have no additional files among the source code in its output. Using MetaEdit+ and Eclipse together, Eclipse project property files are necessary for importing the source code in Eclipse and handling it as a common Java project. The required changes made to the existing MetaEdit+ generator considered mainly the Eclipse project files plus the generation output location. During the generation, the generator reads and writes values to the shared settings file. Later in the same process, the settings file tells Eclipse the name of the Java project which it should import and the Java main class in that particular project.

After the generation the control returns to Eclipse which reads the values from the INI file and destroys it. The Java project is opened automatically in Eclipse, compiled and the main class is run if its name is provided. Following picture illustrates typical generation and import process in integration.

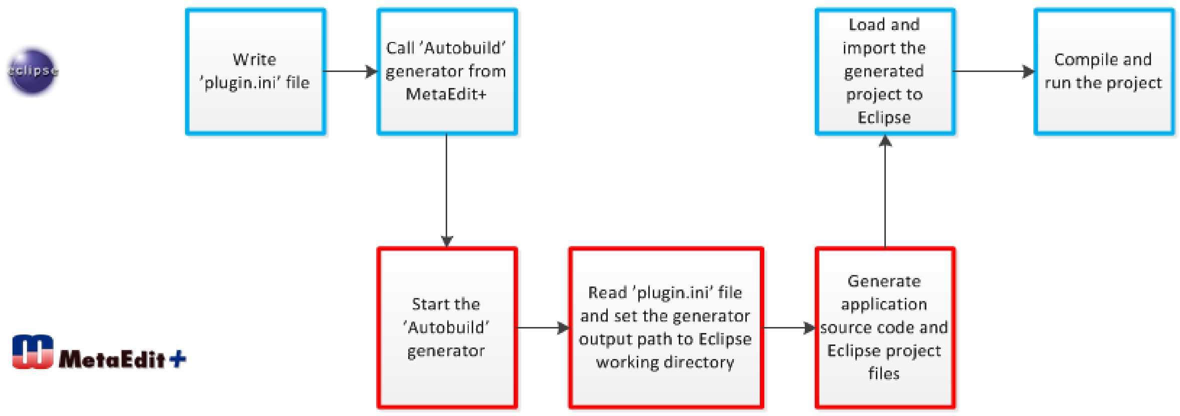

eclipse
Write 'plugin.ini' file
Call 'Autobuild' generator from MetaEdit+
Load and import the generated project to Eclipse
Compile and run the project
MetaEdit+
Start the 'Autobuild' generator
Read 'plugin.ini' file and set the generator output path to Eclipse working directory
Generate application source code and Eclipse project files

Figure 2: Source code generation and import in the integrated process.

4.4 Summary of implemented tool integration

Both of the tools have graphical user interfaces. MetaEdit+ model information is presented in Eclipse plug-in because it enables user to get the basic information of models when using Eclipse. To avoid unnecessary duplication, MetaEdit+ model editors are called and used from Eclipse, instead of creating new windows to Eclipse side. Both of the tools maintain their own look and feel, so the user is always aware which tool he is using. Data produced by tools is not duplicated. Mostly one form of data belongs to one tool. For example, MetaEdit+ stores, maintains, and makes use of the graph data. Eclipse makes use of the source code, both the generated and traditionally developed framework code. The shared settings file is shared between both of the tools. That's because the data is used by both of the tools.

The information shared between the tools consists of the models information contained in MetaEdit+ and the generated source code that imported to Eclipse. None of these is replicated: the graph information is fetched from MetaEdit+ to Eclipse and it's only kept in memory at runtime. All persistent data related to the models is being held by MetaEdit+. The generated source code is imported to Eclipse and is not treated by MetaEdit+ any more.

From the Wasserman's data manipulation view, no manipulation is needed to transform the source code and project properties from MetaEdit+ to Eclipse. Nor is there any redundancy of shared data: Eclipse has always the latest generated source code.

Looking at the process which the integration has automated for user there can be seen couple of major improvements compared to the process with no integration. Suggesting that the user wants to have the generated source code on his hand and to be able to manipulate it, Eclipse makes a huge difference to any text editor. It helps in several ways to read, create, and test the written code that can be adapted later to MetaEdit+ generators. Eclipse as integrated code editor is a benefit for (meta)modeler that wants to see and manipulate his code.

Making the user to use Eclipse as primary code editor was a choice that was made after the generation target language; Java. As it can’t be changed there is still possibility to use the same design template and to implement similar functionality to some other development environment that supports other programming languages. This way the models in MetaEdit+ are kept and the only parts that need additions are the code generators to support new programming language and development environment. Actually, after this implementation work, similar solution was made to Microsoft’s Visual Studio, and new code generators were designed to generate C# and Visual Studio solution similar to Eclipse project.

5 Related work

Heiko Kern (2008) has implemented integration for MetaEdit+ and Eclipse. The solution is a mapping system that transforms metamodels and models from MetaEdit+ to Eclipse EMF framework to EMF metamodels and models. For sharing data between tools, XML transformation is used for mapping the metamodels and MetaEdit+ SOAP API is used to request the model data and build EMF models in Eclipse. In 2009, Kern has implemented similar integration from Microsoft Visio and Eclipse EMF. Both of these works include Eclipse plug-in developed for the (meta)model transformation. Compared to our work, both of these works stay in the same abstraction levels and do utilize the models for real development work. More integration solution for Eclipse is described in (Kern 2006).

Gautier describes case of integrating existing tools to EAST environment in his research (Gautier et. al., 1995). The integration includes a common User Interface (UI) for the tools provided by the platform. The integrated tools also share common data by using a shared repository. Integrating tools to an environment means in this case that none of the tools can take aggressive role to other tools. The framework has its own management mechanism where the tools need fit in.

Existing tools can be integrated also with a common platform. ModelBus environment (Sriplakich et. al., 2006) uses CORBA remote procedure call mechanism to integrate software engineering tools. Similar to our work, tools are separate from each other, heterogeneous and both using their own memory space using the same platform for communication.

Integration for Petri net analysis tool INA and Eclipse is implemend by Adilson et al (2005). Their work includes modeling environment EZPetri inside Eclipse for graphical modeling of Petri net. The integration for the tools utilizes the analysis methods of INA. EZPetri produces a file containing the commands that is being executed in INA. The results are shown to EZPetri user.

6 Conclusion

It this paper, the software engineering tools and models in software engineering have been looked from the tool integration perspective. The tool integration is reviewed from the literature, giving both technical and usage centric arguments of integration aspects. Tool integration can be seen as integration of two separate tools that includes data and control dimensions of integration. On the other hand, integration is always bound to the process and activities that tools automate and to its users and organization it used by. Those aspects should also be included in tool integration design and evaluation.

Integration can be implemented in different ways. Many cases describe a platform or framework that enables tools to integrate to a bigger system. Nowadays Eclipse can be seen as this kind of platform too. It allows to build extensions that integrate new software engineering activities to existing environment. Another approach is to take two existing tools and build integration for them.

A real case integration for MetaEdit+ and Eclipse was described. The integration enables developers to use domain-specific modeling environment and integrated development environment utilizing both of them for effective use of software development. The integration contains information and workflow integration that allow user to make use of the MetaEdit+ graphs in Eclipse and to import automatically generated source from MetaEdit+ to Eclipse.

References

Adilson Arcoverde, Jr., Gabriel Alves, Jr., and Ricardo Lima. 2005. *Petri nets tools integration through Eclipse*, In Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange (eclipse '05). ACM, New York, NY, USA, 90-94.

Balance, R., A., Graham, S., L., Van deVanter, M., L., *The Pan Language-Based Editing System For Integrated Development Environments*", Proceedings of SIGSOFT'88: Fourth Symposiums on Software Development Environments, December 1990.

Biermann, E., Ehrig, K., Ermel, C., Hurrelmann, J., *Generation of Simulation Views for Domain Specific Modeling Languages Based on the Eclipse Modeling Framework,* Automated Software Engineering, *2009,* 24th IEEE/ACM International Conference on , vol., no., pp.625-629, 16-20 Nov. 2009

Broy, M., Feilkas, M., Herrmannsdoerfer, M., Merenda, S., Ratiu, D., *Seamless Model-Based Development: From Isolated Tools to Integrated Model Engineering Environments,* Proceedings of the IEEE , vol.98, no.4, pp.526-545, April 2010

Brown, Alan N., Penedo, Maria H., "An annotated bibliography on Integration in Software Engineering Environments", Technical Report CMU/SEI-92-SR-008, Carnegie Mellon University (CMU), Software Engineering Institute (SEI), May 1992.

Damm, C.H.; Hansen, K.M.; Thomsen, M.; Tyrsted, M., *Tool integration: experiences and issues in using XMI and component technology*, Technology of Object-Oriented Languages, 2000. TOOLS 33. Proceedings. 33rd International Conference, pp.94-107, 2000

Dubois, H.; Lakhal, F.; Gerard, S., *The Papyrus Tool as an Eclipse UML2-modeling Environment for Requirements*, Managing Requirements Knowledge (MARK), 2009 Second International Workshop, pp.85-88, 1-1 Sept. 2009

Gautier, B., Loftus, C., Sherratt, E., and Thomas, L., *Tool integration: experiences and directions*, In Proceedings of the 17th international conference on Software engineering (ICSE '95). ACM, New York, NY, USA, 1995

Gruhn, V., *Process-Centered Software Engineering Environments, A Brief History and Future Challenges*, Annals of Software Engineering, Vol. 14 , pp. 363 – 382, 2002

Huw, O., "Adding control integration to PCTE", Software Development Environments and CASE Technology, vol. 509, pp. 69-80, 1991. Springer Berlin / Heidelberg

Kelly S., *Improving the integration of a domain-specific modelling tool*, Workshop on Tool Integration in System Development (TIS 2003 at ESEC/FSE 2003, Helsinki), pages 57-60.

Kelly S., Tolvanen J-P., "Domain-Specific Modeling: Enabling Full Code Generation", John Wiley & Sons, 2008.

Kern H., *Model Interchange between ARIS and Eclipse EMF*, In: Proceedings of the 7th OOPSLA Workshop on Domain-Specific Modeling DSM'07, Nr. TR-38 Finland: (2007), p. 105--114.

Kern H., *The Interchange of MetaModels between MetaEdit+ and Eclipse EMF using M3-Level-Based bridges* In: 8th OOPSLA Workshop on Domain-Specific Modeling at OOPSLA 2008University of Alabama at Birmingham (2008) , p. 14-19.

Kern H., Kühne S., *Integration of Microsoft Visio and Eclipse Modeling Framework Using M3-Level-Based Bridges*, In: 2nd ECMDA Workshop on Model-Driven Tool \& Process Integration, 24 June 2009, at Fifth European Conference on Model-Driven Architecture Foundations and Applications 2009 Enschede, Netherlands: (2009) .

Lefering, M., *A framework for incremental integration tools*, Computing and Information, 1993. Proceedings ICCI '93, Fifth International Conference, pp.398-402, 27-29 May 1993

Lewis, G.R., *Some simple models of CASE tool integration*, Information Technology, 1990, 'Next Decade in Information Technology', Proceedings of the 5th Jerusalem Conference on (Cat. No.90TH0326-9), pp. 247-250, 22-25 Oct 1990

Long, Frederick W. and Morris, Edwin J., *An Overview of PCTE: A Basis for a Portable Common Tool Environment,* (1993). Software Engineering Institute. Paper 218

Losavio, F., Ortega, D., Perez, M., *Modeling EAI [Enterprise Application Integration]*, Computer Science Society, 2002. SCCC 2002. Proceedings. 22nd International Conference of the Chilean, pp. 195- 203, 2002

Maalej, W., *Task-First or Context-First? Tool Integration Revisited*, Automated Software Engineering, 2009. ASE '09. 24th IEEE/ACM International Conference, pp.344-355, 16-20 Nov. 2009

Nandita Mandal, Ewa Deelman, Gaurang Mehta, Mei-Hui Su, and Karan Vahi. 2007. *Integrating existing scientific workflow systems: the Kepler/Pegasus example,* In Proceedings of the 2nd workshop on Workflows in support of large-scale science (WORKS '07). ACM, New York, NY, USA, 21-28.

Ossher H., Harrison W., and Tarr P., 2000. *Software engineering tools and environments: a roadmap,* In Proceedings of the Conference on The Future of Software Engineering (ICSE '00). ACM, New York, NY, USA, 261-277.3

Pohjonen R., Kelly S., *Domain-Specific Modeling,* Dr. Dobb's Journal, August 2002

Sriplakich P., Blanc X., and Gervais. M.-P., *Supporting transparent model update in distributed CASE tool integration*, In Proceedings of the 2006 ACM symposium on Applied computing (SAC '06). ACM, New York, NY, USA, 1759-1766.

Rader, J., Morris, E.J., Brown, A.W., *An investigation into the state-of-the-practice of CASE tool integration*, Software Engineering Environments Conference, 1993, pp.209-221, 7-9 Jul 1993

Rangarajan, M., Penix, J., Alexander, P., Wilsey, P.A., *Gravity: An object-oriented framework for hardware/software tool integration,* in Proceeding of the Conference Simulation Symposium, 1997 30th Annual, pp.24-30, 7-9 Apr 1997

Selic, B., *The pragmatics of model-driven development, Software, IEEE* , vol.20, no.5, pp. 19- 25, Sept.-Oct. 2003

Smith, M.J.A., *Abstraction and Model Checking in the PEPA Plug-In for Eclipse*, Quantitative Evaluation of Systems (QEST), 2010 Seventh International Conference, pp.155-156, 15-18 Sept. 2010

Soley R., “Model driven architecture”, Object Management Group, November 2000

Sommerville I., “Software Engineering”, Addison-Wesley, 2008, 8th edition.

Sum, S.; Ibold, C., *Information technology support for concurrent and simultaneous engineering-tool integration in a meta framework*, TENCON '94. IEEE Region 10's Ninth Annual International Conference. Theme: Frontiers of Computer Technology. Proceedings of 1994, pp.1068-1073 vol.2, 22-26 Aug 1994

Thomas, I., Nejmeh, B.A., *Definitions of tool integration for environments,* Software, IEEE, vol.9, no.2, pp.29-35, March 1992

Wu, Y., Hernandez, F., Ortega, F., Clarke, P.J., France R., *Measuring the effort for creating and using Domain-Specific models*, In Proceedings of the 10th Workshop on Domain-Specific Modeling (DSM '10), ACM, New York, NY, USA 2010.

Wasserman, A.I. *Tool Integration in Software Engineering Environments*, Proceedings of the International Workshop on Environments, Springer-Verlag, Berlin, 1990, pp. 137-149.

Wasserman, A.I. *Toward a discipline of software engineering,* Software, IEEE, vol.13, no.6, pp.23-31, Nov 1996

Wicks, M.N., Dewar, R.G., *A new research agenda for tool integration*, Journal of Systems and Software, Volume 80, Issue 9, September 2007, pp.1569-1585.

Ye J., Ni Y., He Y., Wang R., Jin C., Hao G., *The design and implementation of a Visual Workflow Modeling tool based on Eclipse plug-ins*, Image Analysis and Signal Processing (IASP), 2011 International Conference, pp.572-577, Oct. 2011

Bao Y., Horowitz, E., *Integrating through user interface: a flexible integration framework for third-party software*, Computer Software and Applications Conference, 1996. COMPSAC '96., Proceedings of 20th International, pp.336-342, 21-23 Aug 1996

Zarrella P.F., 1990. *CASE Tool Integration and Standardization*