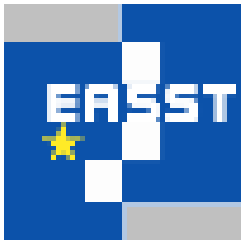
THE EASST NEWSLETTER

Volume 6

June 2003



European Association of Software Science and Technology

Editor:
Dr. Julia Padberg
Universität Bremen On leave from TU Berlin
FB3, Computer Science Department
Postfach 33 04 40
28334 Bremen

E-mail : padberg@cs.tzi.de
URL : http://tfs.cs.tu-berlin.de/~padberg/
Tel : +49-421 218 4202
FAX : +49-421 218 8661

Contents:

Welcome!

Dear EASST Members,

surprise, surprise – another EASST-**Newsletter!**

During the General Assembly I have promised to publish the newsletter more regularly; beginning of June and beginning of December.

So, here it is the June 2003 Newsletter!!

We have chosen to publish in June in order to be able to reflect on ETAPS, FASE or The General Assembly taking place during ETAPS. So, we have the Minutes of the General Assembly in this volume. The General Assembly has elected a new board, and the board has elected our new president:

Tiziana Margaria

We have a short statement by her concerning her ideas and plans for EASST.

And again, there is the TOOL COLUMN by T. Margaria with an contribution by Jörn Guy Süß and Andreas Leicher on *Software Model Engineering and Reuse.* This paper proposes the concept of software model engineering, which pursues model reuse and service separation in order to shift the focus of system development to the design level.

I hope you are enjoying this newsletter and are eagerly waiting for the next one beginning of December. If you have any contribution you would like to have in the forthcoming newsletter please send it to me.

Yours sincerely,

Julia Padberg
(EASST-Secretary)

The New President of EASST
Dr. Tiziana Margaria

Opening Adress

In my vision, EASST will in the next three years provide our members easy access to current and concise technical and non-technical information, facilitate interactions through conferences, through the EASST Newsletters and national activities, with attention to research, applications, and education.

As newly elected EASST President, my goal is to carry out this vision together with the Board Members, the national representatives, and all our members. In particular, it is my objective to represent your interest and emphasize the following directions of activity:

- Guide EASST into an active and lively community. We will explore ways and means to help bridging the gap between theory and practice in software and system technology. This will happen through connections with journals, conferences, and web-based services, by reaching our virtual communities, and informing the members also on thematic meetings and regional activities.
- Promote publications, conferences, meetings, and services for both our research members and our practitioner members, who pay attention to changing technologies and need to continuously expand their technical expertise. We must develop new publications and services in emerging areas, enhancing the Newsletter Columns, and improve our current offerings to serve our practitioner members and student members' needs.
- Develop plans to better serve our members, worldwide. We need more effective and economical methods to deliver information and services to our members. We must bring all our global members to closer collaboration, on a national basis, through our National Representatives, and through thematic initiatives and specific support to their activities.

- Develop EASST's cooperation with other professional societies and initiatives. This includes support of diverse activities in publications, conferences, standards, edu-cation, and international cooperation. It may include strategic partnerships with other entities in order to enhance and bundle member services.

To this aim, the cooperation of all the members is of vital importance. The EASST officials are glad to be available to you for interactions and advice. Do not hesitate to contact me under *tiziana.margaria@cs.uni-dortmund.de* if you have suggestions on how EASST can improve its services to the community.

Dortmund, 31.5.2003

Tiziana Margaria

Software Model Engineering and Reuse with the Evolution and Validation Environment

Jörn Guy Süß *and Andreas Leicher *
*Computation and Information Structures (CIS), TU Berlin

Abstract. *Reuse is an important aspect of software engineering that promises advantages like faster time-to-market, cost reduction, better maintainability etc. The software industry focuses on components and commercials of-the-shelf in order to gain reusable assets. However, reuse on the design level is normally not addressed. If we come to perceive models as assets of the software process, then the design moves from the periphery of software engineering to the center. This implies several advantages, like an improved systems's overview and insight, because of greater abstraction and easier comprehension of the design concepts. This paper proposes the concept of software model engineering, which pursues model reuse and service separation in order to shift the focus of system development to the design level. Models expressed in an open available format - independent of a particular modelling tool - facilitate the exchange and reuse of models. As a result, the user community grows and the quality of model artifacts improves, because of frequent use, correction and peer review. Modelling tools primarily provide a work-place for the definition of models including proprietary services like code generation. The separation of services and modelling tools enables independent reuse of services. Consequently, the efficiency of the modelling process increases as services become globally shared assets. This paper show how these ideas are reflected in the design and functionality of the* Evolution and Validation Environment *EVE. EVE provides an interoperability platform for model exchange. It consists solely of components which adhere to open specifications, such as XMI, UML, and OCL. EVE is designed as a loosely coupled system, which allows users to executed services locally or transparently over the network by combining services in arbitrary ways.*

Keywords: software engineering, models, UML, XMI, Jini, SOAP

1 Introduction

Achieving reuse in software construction has long been a core aspect of software engineering [4], and strategies like high level languages, language structuring and encapsulation have made valuable contributions. Today, reuse is an important aspect of component-based systems [6]. The software industry produces Commercials of-the-Shelf (COTS) as reusable parts [13], which deliver advantages like faster time-to-market, cost reduction, better maintainability, configurability etc. While the reuse qualities of implementation-oriented artifacts improve, one important contributing factor of the engineering process

remains out of scope: Models of software systems still are not assets of the software process [14]. As a consequence, they cannot be used as an abstract basis, exchanged in a market or handled with standardized tools and services and have to be reinvented in a costly and time-consuming way.

If we come to perceive models as assets to the software process, then they move from the periphery towards the center of this process. We believe that *Software Model Engineering* (SME) - the result of that paradigm shift - will deliver an improvement to software development theory and practice. Visual languages like the Unified Modelling Language (UML) provide a common understanding of modeling, which can be used to exchange and discuss model instances. Thus, SME proposes a change from implementation-oriented development to model-oriented development. This results in a higher level of abstraction, which implies several advantages:

Abstraction. A suitable model abstracts from aspects the modeler does not currently reason about or wishes to communicate. Therefore, good design is equivalent to understandable concepts and leads to an improved system's overview and insight.

Models encapsulate constraints, behavior, structure, interactions, as well as non-functional properties on an abstract level. Therefore, developers do not need to deal with certain details of underlying technologies, because they result from the modeled properties.

Comprehension. Formal or semi-formal models, like those formulated in UML, follow a common language definition, which is comparatively easy to understand. Therefore people from different fields can comprehend design concepts in a common way and are able to discuss alternatives and consequences.

Constraint Monitoring. If the modeler states requirements as invariants at the design-level, it is possible to enforce these invariants automatically in subsequent stages of the project and on more detailed levels of the model. The modeler can apply algorithms to the formal part of models to check types for consistency, validate assertions about behavior, or to generate code.

One can think of SME as the discipline of the creation of model-processing services in general. In extension of the UML's credo to provide a general modelling *language* primarily for software engineering projects, SME pursues the creation of general modelling *tools* for the same audience. While the UML is characterized by its capability to express the meaning in a semi-formal model, SME additionally allows to express and execute the mechanics of a semi-formal model-based development process. However, SME suffers from several problems. The authors have experienced the following three as imminent:

First, a large number of different methodologies and notations exists which fragments the model user community. Additionally, model information is usually stored in the proprietary file format of a specific tool and thus tied to a specific vendor, which creates another barrier for reuse. Consequently, it is not possible to exchange models, and the community does not reach a critical mass, that would support reuse of models or model parts.

Second, models often cannot be checked for correctness. It is rare for the proponents of visual languages to give a formal semantic specification, which leaves room for ambiguities. Thus, projects still discover design errors during implementation, which could have been detected and avoided during design using simple checks against the formal structure of the model.

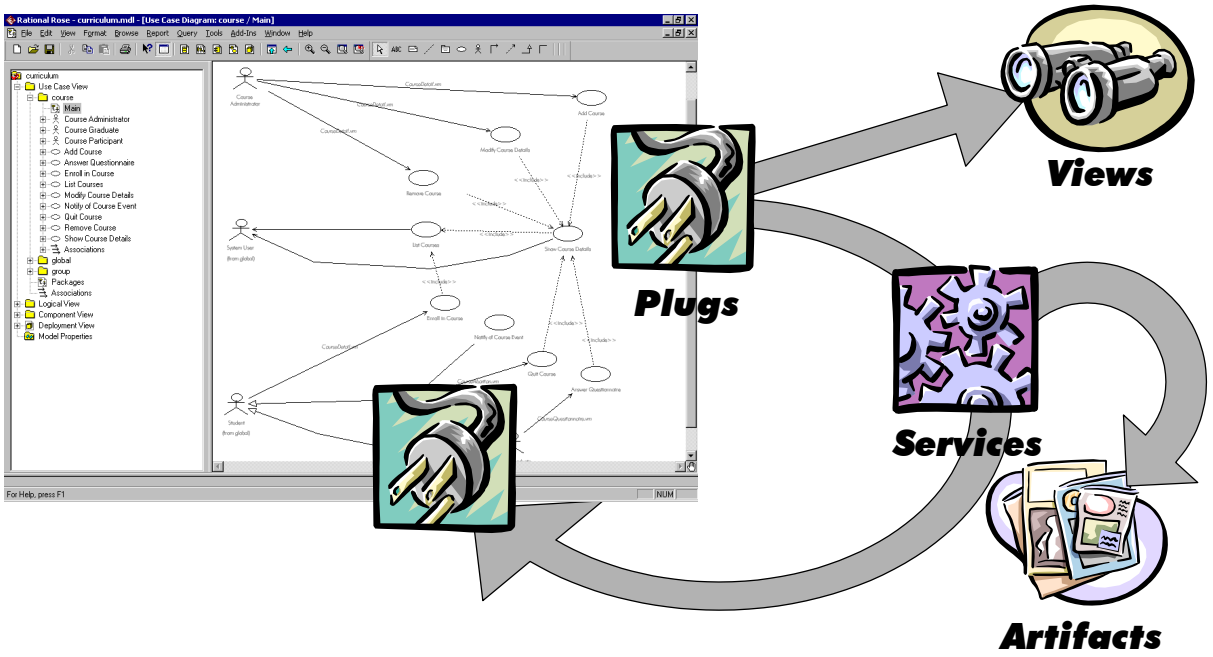


Figure 1:
EVE's Concept: Proprietary plugs give access to the model, external services work on it and may generate artifacts.

Third, models often are not kept synchronized after the system's implementation. Roundtrip engineering still is elusive. Consequently, the implementation is fast becoming inconsistent with the design and the outdated model is devalued.

To address these problems and gain the reuse advantages in SME we propose the *Evolution and Validation Environment* (EVE). EVE provides an interoperability platform for model exchange. It consists solely of components which adhere to open specifications, such as XMI [20], UML [17], and OCL [23]. EVE (figure 1) separates the concerns of modeling tools and services: The modeling tools primarily provide a software-ergonomically well-designed work-place for the definition of those aspects of models, which cannot be created automatically. Services, which are embedded in the infrastructure and shared among the community, encapsulate algorithmic knowledge about the software process as stand-alone components. The most obvious and prominent applications of services would probably be consistency checking, forward and reverse engineering. Plugs enable the separation between the modeling tools and the services. They grant access to the model information inside the proprietary model store of a tool. They are based on the XMI specification and extract or store XMI files, so that services operate on data based on an open, well-defined specification. Views are external components, which visualize three types of information inside EVE: UML-Models expressed as XMI, artifacts expressed in arbitrary form and service properties or status expressed as property files or objects.

EVE is designed as a loosely coupled system. Thus, users can offer or delete services at runtime as well

as combine them in arbitrary ways. Users can call services locally with low overhead, or transparently over the network. EVE provides developers with security and transactional behavior required for network use through an application frame which encapsulates the service's implementation.

EVE shifts the focus in systems' development to models on the design level. On that higher level of abstraction we gain the reuse-advantages in time-to-market, cost reduction, maintainability, configurability, as model exchange increases and roundtrip engineering is supported through services. Separation of concerns between modelling tools and services overcomes the fragmentation of the user community along the lines of tool vendors. We believe that by these measures we can achieve model reuse and clear the path for the development of a model user community, improved availability of services and model quality, and finally increased process efficiency. A modelling community will unify novel and advanced findings of the research community with pragmatic and ambitious approaches of commercial enterprises. Transfer between theory and practice will increase and feedback will be accelerated.

2 Related Work

EVE plays an important role in the materialization of the paradigm of 'Continuous Software Engineering' (CSE) [16, 24]. CSE deals with the evolution and quality of software systems. It pursues integration and extension of object-oriented modeling techniques both spanning the different phases of the software development (analysis, design and implementation) and increasing consistency within the phases (e.g. between classes and state machines of the design phase). CSE implies controlled evolution of software systems. Each modification of a system normally results in a sequence of consistent evolution steps (i.e. a super-step). For each step, consistency has to ideally be guaranteed or at least supported with regard to all phases of software development. Figure 2 sketches the interplay of forward and reverse engineering and the resulting evolution steps. Consequently, mechanization and automation of the engineering process is vital to the success of CSE.

2.1 Dimensions in Software Development

Approaches to integrate models into the software engineering process [15, 9] usually imply a strict *method* that orders the process. SME in contrast is *method-neutral*, but *process-aware*. It proposes an ordered progression through the development space of projects.

Figure 2 suggests a two-dimensional development space with vertical activities and horizontal iterations or super-steps. Pons [8] additionally describes an internal dimension, which contains descendence-traces between artifacts, describing from which artifacts they are derived. Awareness of these *three* dimensions of development is necessary for development teams to set up effective procedures and tools for a model-based software process. This is similar to the awareness of dependencies and tool requirements in traditional source-code-oriented build management.

2.2 Model Driven Architecture

OMG's Model Driven Architecture (MDA) aims to facilitate reuse by creating an ordered spectrum of refinements between Platform Independent Models (PIM) and Platform Specific Models (PSM). The

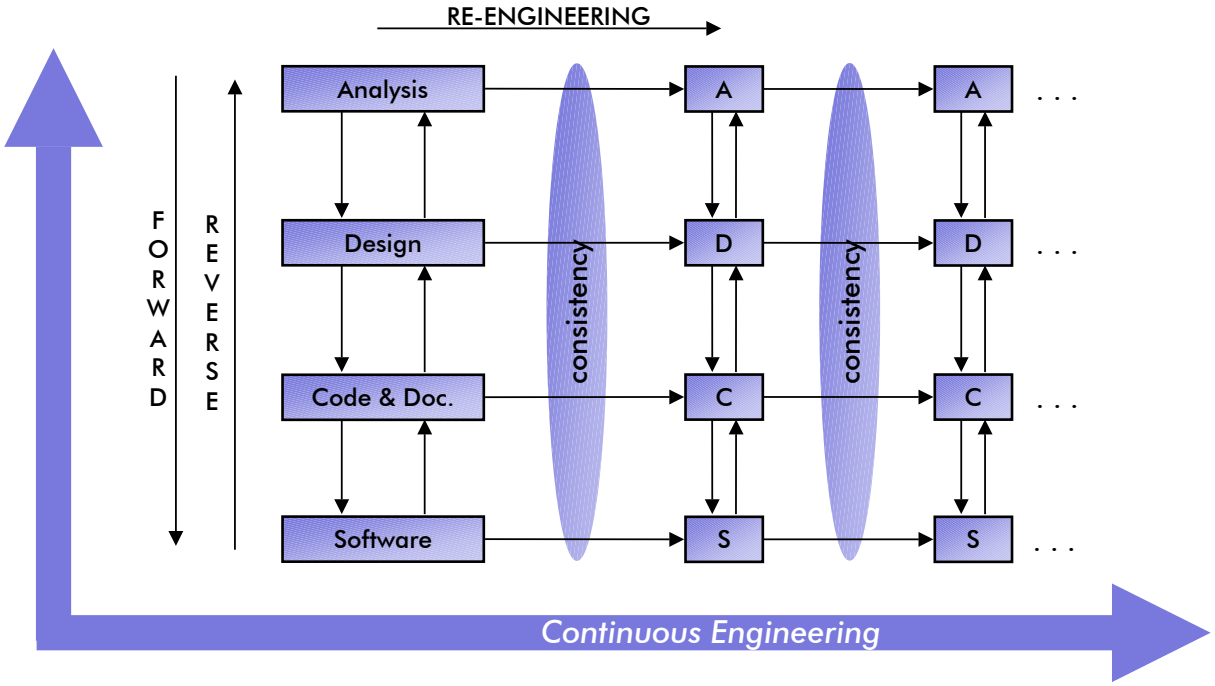


Figure 2: Reference Model '*Continuous Software Engineering*': Consistency of system artefacts is preserved across development phases and along the systems' lifetime of changes induced by continual development.

consistency of refinement steps is guaranteed by UML Profiles. MDA implies a waterfall model of the engineering process and requires specific engineering tools. SME and MDA differ in philosophy: Primarily SME does not rely on a methodology. Specifically SME is

More pragmatic. With EVE we propose a framework that improves and broadens model reuse *in general*, by attaching as many sources of model information and offering as many services to work on that information as possible. SME does not imply that existing tools be radically replaced, rather that they should be well-connected. This lowers the implementation cost and reduces the risk involved in the paradigm-shift.

Less complex. Services written for EVE do not need to employ UML Profiles, Meta-models and the associated theory. They can just as well be quickly written 'hacks' for ad-hoc processes and exploration. This reduces training cost and makes the approach more appealing to a general audience.

Not confined to UML. Although EVE is primarily based on the UML and on related specifications and technologies it supports models based on other arbitrary meta-models. Consequently, developer's that are familiar with EVE can build new model-based integration suites with low extra effort and cost.

2.3 Interactive Modelling Tools

Because in SME mechanical services that provide the iteration dimension (i.e. the system's evolution) are moved out of the CASE tools, the tools can take on a new character. Besides their role as a drawing board for static structures, they can become interactive specification aids for the dynamic properties of models. They can help the modeler by generalizing scenarios into behavior or by playing through the state space of the model.

Life-Sequence-Charts (LSCs) and the Play-In/Play-Out approach proposed by Harel [12] allow the modeler to implicitly define the intended behavior of the system by training a mock-up of the interfaces of the system. This approach is so successful, that it can be used like a light-weight scripting language.

Stevens [22] proposes that modellers view the design of a system as a game between two players, Refuter, who tries to demonstrate that the system does not satisfy its recorded requirements, and Verifier, who tries to demonstrate that it does. Losing the game discovers a design (or specification) error. These playful tools could make it easier for a designer to solve problems in that they cast the computer in the role of an intellectual sparring partner.

2.4 Meta-model Standards

Although the concept of SME scopes the term 'model' very widely, EVE as a tool and collection of services for SME is built primarily for the large community of UML practitioners, which has emerged historically: In the late 80's industry convergence ended the 'object wars' and led from many similar modelling languages with minor differences in their expressive power to the UML as one broadly-supported modelling language intended for general purpose use. UML consists of a collection of diagrammatic representation techniques and a set of axiomatic model elements (use case, class, constraint, component

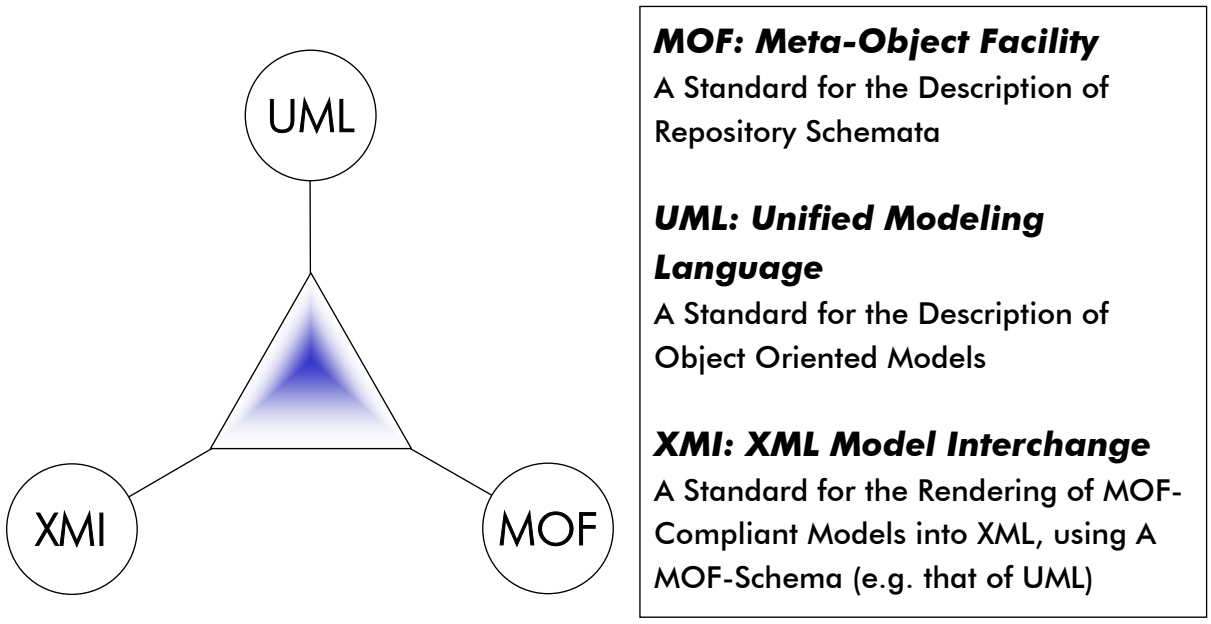


Figure 3: XMI Instance as ternary compliance binding of three standards: XMI describes how to render a model expressed in terms of a MOF-compliant metammodel such as UML as an XML document.

etc.), which are held together by a static object-oriented meta-model and declarative constraints, with semantics explained by precise English text. Since its introduction in 1999, the UML has arguably reached the first of its primary design goals, which is '*to provide users with a ready-to-use, expressive visual modeling language to develop and exchange meaningful models.*' [18]. However, the UML did not at first provide a standardized interchange format, which led to a plethora of proprietary formats.

This lack was overcome through the appearance of XML, and the subsequent definition of the eXtensible Metainformation Interchange standard (XMI). Now UML models could be encoded for transfer and storage. From a software engineering point of view XMI appears as a file format for models expressed in the Unified Modelling Language (UML). A closer look at the documents however reveals that another standard is involved: The OMGs Meta Object Facility (MOF) [19]. MOF operates on a level above the concept of UML (i.e. the UML meta-model), rendering the XMI file a ternary binding between UML, XML and MOF.

This explains why XMI is a specification in its own right and not an appendix of the UML specification: It is not a *data* format, but describes a *procedure* to create a data format capable of storing model instances which have been formulated in accordance with a MOF compatible meta-model (cmp. figure 3). This also allows for the expression of model data based on meta-models other than UML, e.g. the Common Warehouse Metadata (CWM) standard, the Java Modelling Language (JML) or even the content model of XML Documents.

MOF was created as a facility in the context of the Common Object Request Broker Architecture

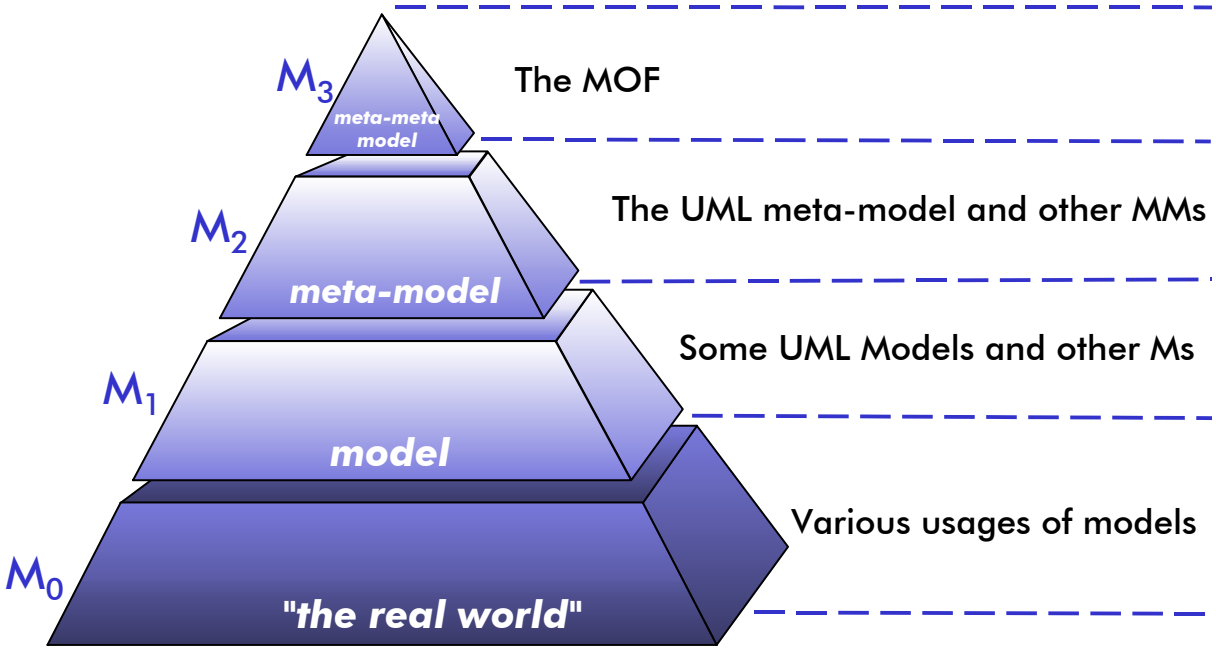


Figure 4: OMG Model Stack: The OMG defines modelling on different levels. With increasing abstraction fewer concepts exist in the modelling languages. On the M2 level only major OO concepts like packages, classes, relations and the OCL as constraint language remain as a means of description.

(CORBA) before the end of the 'object wars', when a large number of different meta-models and methods still existed. It seemed sensible to create a common repository, that could accommodate different 'modelling languages', to enhance the communication among software developers. Thus, the MOF standard defines interfaces and methods enabling expression, storage and externalization of models based on arbitrary meta-models. MOF itself expresses meta-models using class diagrams and primitive types. So if for example the Entity/Relationship (E/R) meta-model was described in MOF, then E/R tools could read models from and write models to that store, or externalize them in an E/R-based XMI data stream following the specifications of the XMI standard.

3 Concepts of the EVE Framework

SME shifts the focus of system development from the level of infrastructure (i.e. code) to the more abstract level of design. To prepare this shift the EVE framework pursues model reuse and service separation.

Model Reuse. Today, qualities of design reuse do not impact the actual software engineering process. However, Biggerstaff et al. [3] note that design reuse is more effective and beneficial than code

reuse. EVE fosters design reuse based on open standards to facilitate a user community to grow, as more model data in standardized format becomes available. Accordingly, the quality of model artifacts and the efficiency of the overall modeling process increases, because frequent use, correction and peer review helps to remove insufficiencies and problems.

Service Separation. The separation of services and modelling tools enables the independent re-use of the framework's components (i.e. services, plugs, views). Consequently, frequency of use is increased and applicability improved as compared to a monolithic implementation. Intensive reuse again increases the efficiency of the modeling process as it frees resources in the user communities, which can be invested elsewhere.

The realization of these objectives promises the same advantages that are the focus of Component Based Software Engineering (CBSE): faster time-to-market, cost reduction, better maintainability, configurability etc. Moreover, model instances and services developed in the scientific community come into the scope of industrial use, as practitioners can locate a domain experts in science and then apply their experimental domain services. Thus EVE helps to promote good modeling methods and services into practice and yields valuable feedback for the research community.

To allow the modelling community to integrate semantically and exchange knowledge on domains and methods in an adequately formalized, semantically unambiguous way, we chose UML Profiles as a fixed syntax or form. Profiles are defined applications of the UML extension mechanism, which specialize elements of the UML into stereotypical elements of an application domain. Although the syntax could be arbitrarily defined, in practice success is determined by ergonomic aspects like clarity of denotation, ease of use and familiarity with the subject matter. UML Profiles fulfill these criteria, because they are part of a well-known standard. Like the UML metamodel, Profiles consist of three parts: An implicit meta-model contains the base classes and applicable tagged values of the stereotypes. A precise English language description defines the semantics. Constraints attributed to the stereotypes lay down well-formedness rules for the domain formulated in the Object Constraint Language (OCL).

That ternary character of Profiles simplifies typical software engineering tasks: Generation of code from platform specific models is facilitated, because the stereotypes and associated tagged values will generally reify language constructs. The precise English description serves as an effective documentation of the domain. The OCL well-formedness rules can be used in a generic model checker to validate conformance of a model instance with the domain. Thus, a Profile is like a 'module' or 'template' to work on a specific domain. We envisage that a large part of contributions to the modelling community will therefore be Profiles.

We believe that Profiles are effective because they provide a clear structure to the presentation of domain information, in essence a good template or document type. Also, Profiles do not necessarily require graphical editing, so they can be represented as an XML document and manipulated generically and independently of the presentation. Their structure is documented as part of the UML standard. Finally, a Profile can be embedded into a UML model so that the model instance and its domain description form an inseparable compound.

Technically, savings of SME are gained through the application of rules, which enable automation of repetitive software engineering tasks. From a computational point of view any rule-based language would

suffice for this purpose. In EVE OCL was selected. It is applied for all constraints within Profiles and as a query language to select elements for model evolution and forward engineering. OCL was created in the context of financial institutions to allow domain experts to specify constraints on object-oriented models in a user-friendly way. It does not use any special characters in denotation and is based on simple first-order logic. Because of these properties the language was later included as part of the UML specification and reused on the level of the meta-model. Through extensive use in the OMG's MDA OCL will gain further weight. As a result, the availability of reusable components for OCL like syntax checkers, parsers and compilers will grow, improving the quality of EVE and lowering implementation cost.

4 Implementing EVE

The implementation of the aforementioned concepts implies decisions ranging from the selection of components over the basic terminology and architecture patterns to fine-grained modelling.

4.1 Selecting Components for the Framework

The largest part of EVE is made up of existing components. For EVE, the rating of fitness of candidate components was calculated based on factors derived from its usage scenarios in a software development environment and on the Internet.

Internal parts of the framework will run on Local Area Networks (LANs) and are used by workgroups. Services offered there might be experimental or of low quality. Heterogeneity with regard to equipment and methods must be handled. These circumstances translate into the following requirements:

Simplicity. Framework components need to be user friendly and therefore simple to operate. Under UNIX for instance, parts of the system would be UNIX commands. They would perform their functionality on local files as a filters or run permanently as a background process to provide network services.

Distribution. It should be possible to distribute services arbitrarily within a LAN. The framework should handle distribution of services across the network transparently. Hence, services need to handle discovery and connectivity themselves.

Transactionality. Combination of services allows to set up complex functionality. Such dependencies involve transactional behavior. If one service calls one or more other services, it must be possible to roll back faulted operations without hand-coding synchronization details. Hence, the framework has to provide primitives of distributed transactions.

Manageability. In a dynamic distributed environment central administration is almost impossible. Administrators, if available, usually provide support for the network, storage and operating systems (OSs) only. As a result the framework must maintain itself. It should run out-of the box and either heal itself from faults or provide plain instructions for repair.

Cross-Platform Executable. It should be possible to execute services on every workstation regardless of the OS and processor architecture. This implies, that a virtual machine or script interpreter must be used as the basis of the whole system.

Also, EVE must be able to offer services of an organization to interested parties on the Internet.

Web-Integration. Technologies used to implement the external Web presentation may vary greatly. As a result, EVE must either be integrateable into most of these technologies as part of the HTTP service or provide an HTTP service itself.

Demilitarized Zone. Usually access to an organization's Web resources is channeled through a firewall and centralized on certain servers in an intensely supervised zone. So EVE must cooperate with a firewall and provide facilities to publicize and withdraw EVE services on the Internet.

Business-Integration. EVE may be used in a commercial context, where business relevant information about calls must be forwarded to the surrounding infrastructure and performance may be of concern. Additionally, existing security infrastructure has to be supported by EVE.

In order to fulfill these requirements, we decided to use the Java programming language to realize platform independence. Distribution is accomplished through the use of Jini, which forms a self-organizing and self-healing network with low administration overhead. Jini also facilitates transactional behavior between services. We decided to realize EVE's Web-integration with JSP Tag Libraries and Servlets. Formal requirements are approached via the Java Metadata Interface and the Metadata Repository framework, which is able to handle MOF based model instances.

4.2 Architecture Overview

The objectives introduced above led to a system design based on the separation of concerns between models and services we already mentioned in the introduction. Thus, first class abstractions are model instances, which are rendered in XMI descriptions. Most modelling tools support an XMI dialect that more or less complies with the XMI Specification [20]. So before such data enters EVE, it has to be cleansed in order to reach full compliance. After this has been achieved, the consistent use of standards reduces the dependency on specific design tools, enables the exchange of artifacts and realizes interoperability between heterogeneous tools.

Figure 1 shows the basic concepts of the EVE framework: Models residing in arbitrary (UML) modeling tools are extracted through proprietary interfaces and transformed into correct XMI representations. These model instances can be displayed through individual views or processed further. The framework consists of the following basic components:

Plugs. Model instances are handled and stored by modelling tools in different proprietary formats. In order to use these model instances, they have to be translated into proper XMI [20]. Plugs provide this functionality. They extract model instances from tools into the framework and merge incoming model instances with local data.

Services. Services define functionality that can be applied on model instances. A service that is supplied with a model instance, applies its functionality on this instance and then offers the result and feedback. Typical examples of services are validation, feedback and model transformation, based on the canon of OMG standards as UML and OCL. But specialized services may any other desired formalisms like temporal logics or automata as long as these interface with the MOF repository.

Views. Views are components for immediate visual presentation of work items, feedback, and configuration information in EVE. A View can be a HTML Page, a RDF[11] feedback or error report or a (full fledged) GUI. Views are also used to monitor the status of the EVE platform, as users must be able to determine which services are available.

Artifacts. The term artifacts refers to all data that does *not* represent a (UML) model instance. In implementation terms this means all data that is not in XMI format. Services may consume or produce arbitrary artifacts such as source code, reports, etc. However artifacts cannot be passed around in the primary framework. If services intend to cooperate on the content of the artifact, then the artifact must either be *modelled*, i.e. expressed as a model instance and embedded into the model that is passed around, or the services must define a secondary proprietary communication channel.

Essentially, the architecture of EVE follows the *Pipe-and-Filter* architectural style [21, 5]. EVE handles the communication of model instances (pipe) between the three kinds of components (filter): Plugs, Services, Views. The basic pipe-and-filter architecture has been extended to handle loosely-coupled distributed communication by message passing between the different components of the framework.

4.3 Design of EVE

In the design of EVE we strive to reuse as many well-known and well-tested elements of software engineering as possible. Reuse on the level of models started with the construction of the system from *design patterns*. The resulting *interface design* was subsequently extended with the fixed types of the implementation platform. Then, these interfaces were assigned to *components* and finally placed in a general *package structure* to allow distributed development and detect access violations in contributed code. The reminder of this section exemplifies this approach.

4.3.1 Pattern Selection

Christopher Alexander proposes that complex composites can actually be constructed from a selection out of a set of simple patterns, which collectively resolve the forces that occur in the problem context [1]. To factor the design of EVE, patterns from the classic software engineering pattern book by Gamma et al. [7] have been applied. The well-arranged terminology and documentation of patterns should facilitate the understanding of EVEs internals. Here we just give an overview of the major patterns used in the design of EVE:

Model-View-Controller EVE views display UML models/model elements, artifacts or service properties and status. They observe this data and update themselves on change. They may hold one or

more references to processes which change the data (primarily services). As such, EVE views are a superclass of specialized views in three different model/view/controller designs: Model handling, Artifact handling, Service handling

Command EVE services can be interpreted as general purpose manipulators on models and artifacts and may have state and side-effects on other models. These properties closely resemble those of the *Command* design-pattern.

Factory As services can be chained and nested, several instances of one service class might share resources. These savings are realized in EVE through the application of the *factory* pattern.

4.3.2 Interface Definitions

As it should be easy to implement an EVE service, the requirements on the service functionality should be kept to a minimum and the description of these requirements should be precise. Since services are central to the concept of EVE, the interface design of `IEveService` is exemplified in figure 5.

The `IEveService` interface provides the combined functionality to exchange model data and to execute the respective service. It additionally provides a name, which is required to identify the service. The name is independent of the class to allow services to choose their names dynamically. This provides a way to represent such varying aspects as service parameters, provider and status informally inside the name of the service, allowing developer's to invent their own naming convention as necessary. The interface consists of the `ICommand` interface, which fulfills the *Command* pattern and the `IEveShunt` interface, which defines an exchange facility for model data between the framework and the EVE service.

`IEveShunt` provides three channels for input, output and error/feedback handling, respectively. The input and output consist of serialized model information. The error channel provides error reports and feedback. This design facilitates loosely coupled communication.

4.3.3 Component Delineation

The interface design of EVE has been subsequently consolidated into those components, which make up our default implementations (figure 6). Since the behavioral assumptions on the interfaces have been kept to a minimum (most operations are stateless with regard to the state of the entity abstracted by the interface), EVE system implementations could in theory be composed of component implementations from different implementers or vendors. Components can be updated and replaced without the negative impact of inheritance breaking encapsulation.

Frame The implementation frame contains environment connectivity, a command-line parser and a coordinator for sequential operation (chaining) of services. It instantiates the service(s) through dynamic class loading. Thus, it is possible to run a service like a command-line filter. Additionally transactional behavior is supported to allow the vertical composition of services.

The distribution includes an example implementation and template of a service for demonstration and testing purposes. This service generates WSDL files from model elements stereotyped as ≪`Interface`≫.

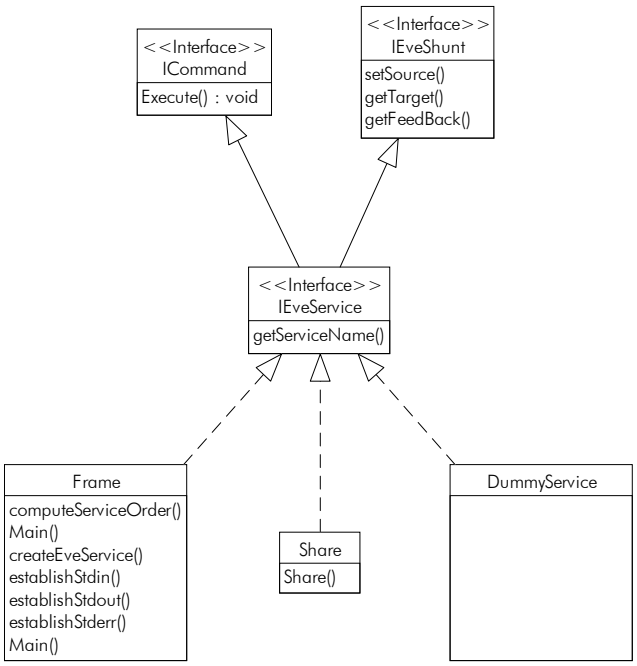


Figure 5: First-order relations of *IEveService*: The diagram only contains relevant compartment members and omits visibility modifiers.

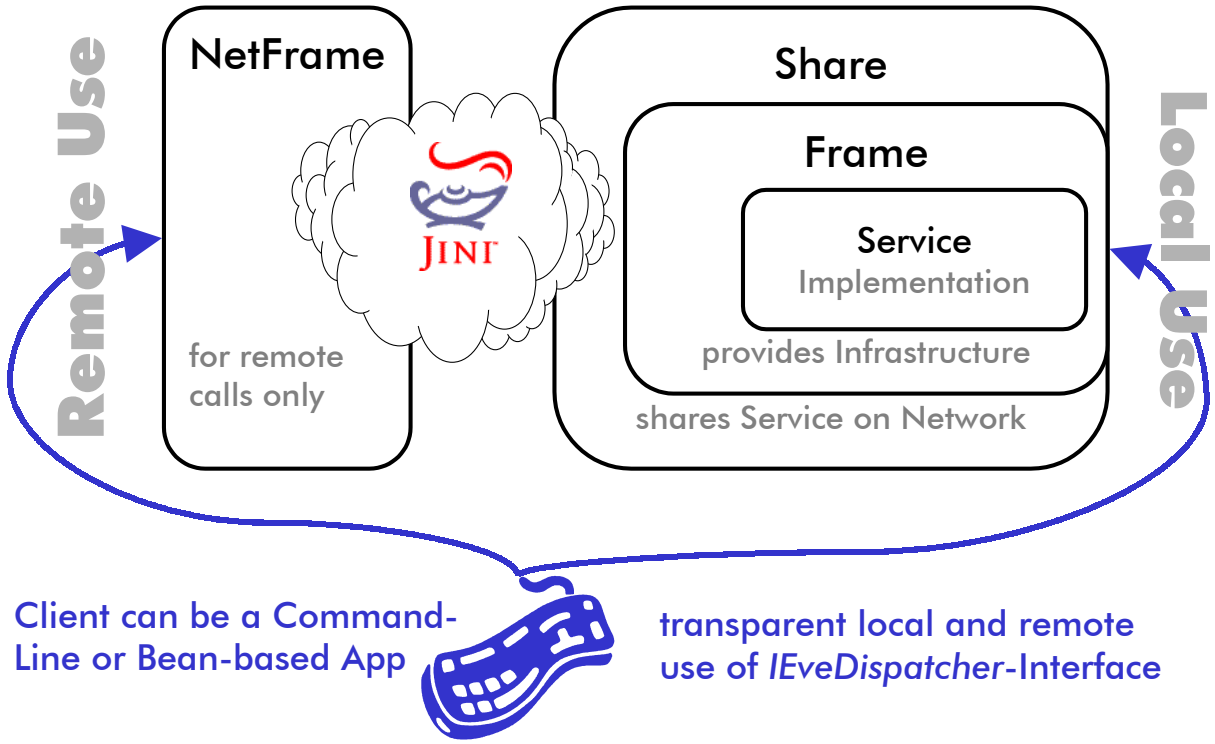


Figure 6: EVE Components

Share Services, which are located on the LAN, are supported via a `Share` component based on Jini[2]. The `Share` component encapsulates the `Frame` component and shares the component on a Jini network as a service. A `Share` registers with a Jini lookup service and can be accessed by `NetFrame` instances, as well as any other Jini client.

NetFrame The `NetFrame` component executes an EVE service remotely. It locates the EVE service requested (using service name resolving) and executes the service via the service interface.

The overall structure has been scoped through packages as shown in table 1.

5 Example Application

To show how EVE actually performs in an environment, we present a walk-through in a realistic scenario employing a standard EVE service.

5.1 Evolution Service

In object-oriented modeling, modelling methods often encompass rules, which describe modification steps. An example of this would be the UseCase-to-Interface Method by Jansson [10]. It declares that for

model	contains an abstraction for the data model: Interfaces for models, model elements, model queries and feedback.
core	encompasses everything required for local use of the framework. Interfaces for services, service factories, the I/O facility and a naming scheme for service lookup. The components of the core package are Java Beans-compliant.
net	holds all functionality required to provide and use services on the Jini network, including a Jini bootstrap utility.
web	provides a JSP tag library for administration of services and controlled WSDL/SOAP access to the services across the company firewall.
service	is the package for standard and example service implementations like the WSDL generator, generic UML Profile Checker and generic UML Mapper.
transform	consists of tools for UML model maintenance, mostly based on XSLT: An `XMI Cleaner` to remove product specific dialects and a `Model Browser`. Most of these components are also wrapped as a JSP tag library.

Table 1: EVE Package Structure

each association between an actor and a use case, an interface must exist. If we do not want to generate all these interfaces by hand, a rule can be defined, which describes how the interface elements are to be generated from the associations between actors and use cases in the model. That algorithm, which helps to develop or evolve a model can be encapsulated in a specific evolution service. The service can be developed from a fixed filter into an interpreter which accepts the algorithm as a parameter. The resulting generalized evolution service could add arbitrary model elements based on the algorithm supplied. Further capabilities to remove model elements and to recurse over a number of mapping steps complete the EVE Evolution Service (ES) as a generic machinery for algorithmic evolution of models.

5.2 Example Step

The following example describes a single evolution step of ES in the framework. It requires the execution of operations shown in the sequence diagram in figure 7. A developer initiates the service execution in the ArgoUML modelling tool. In this example EVE's plug is directly integrated into ArgoUML and the service is called as a distributed application in a JINI network:

1. A model instance is extracted from the modelling tool via a *Plug*.

2. An object stream of type `IEveModel`, which is an encapsulated XMI stream containing the model instance is obtained. The next step executes a service either locally, on a LAN or on the Internet.

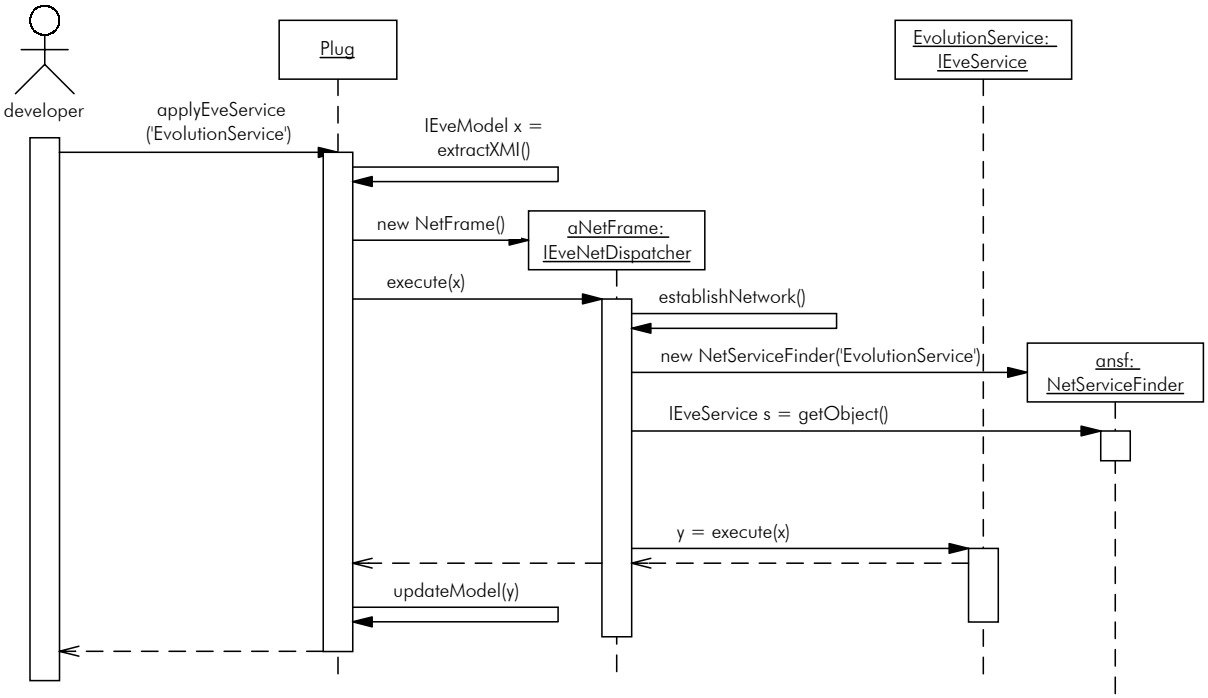


Figure 7: Example: ES Application

In this example the service is invoked on a Jini network: A network connection has to be established and ES has to be looked up via a service listener. In the EVE framework `NetFrame` components cover these complex tasks and forward the service's invocation to the server implementation (details omitted for clarity).

3. ES returns a modified model instance encapsulated in an `IEveModel`. Additionally, it provides feedback concerning the service execution (not shown).

4. Finally the model instance is reintegrated into the model inside ArgoUML. If ES only deletes model elements this is relatively easy to accomplish. Integrating new model elements is challenging because XMI only contains information about model structure but not about diagrams. Hence the plug has to provide some display heuristics for this case.

5. Alternatively the modified model instance as well as the feedback can be viewed or printed in a View.

6 Conclusion

Today, proprietary modelling tools dominate the market. Developers are bound to tools used in their company. Services like code generation or reverse engineering are only available in a subset of these tools. Currently the lack of open model exchange is a common drawback in the application of these tools: Models cannot be used as an abstract basis of system development, cannot be exchanged in a market or handled with standardized tools and services. Existing services in different tools cannot be shared, as they are not portable. So model instances and services must be reinvented for each proprietary platform in a costly and time-consuming way. Thus, current system development is focused on the level of infrastructure, because tool developers benefit from defined open structures of programming languages and libraries.

The EVE framework pursues the objectives of model reuse and separation of modeling tools and modeling services in order to shift the focus of system development to the more abstract level of design. This paper introduced the EVE framework which facilitates reaching these objectives. Primarily, the framework is based on open standards, such as XMI, UML, and OCL. All models instances which comply with MOF are exchangeable and modifiable within the framework.

The realization of these objectives promises the same advantages that are the focus of CBSE, like faster time-to-market, cost reduction, better maintainability, configurability etc. Moreover, model instances and services developed in the scientific community come into the scope of industrial use, as practitioners can locate a domain expert in science and then try out her domain services.

Irrespective of concerns, about or stance towards intellectual property rights on software artifacts, such an environment delivers a value to any model user community. In areas where exchange beyond tight boundaries of a working group is seen as undesirable, it creates a local sphere of collaboration. Alternatively, if exchange is seen as desirable per se, as it is in the open source community, the environment provides a vehicle for such global exchange. When professional services like software consultancy and expertise are the focus, per-use payment of expert services can be implemented easily with the framework by combining it with any generic e-business suite.

The separation of services and modeling tools enables the independent re-use of service functionality. Consequently, frequency of use is increased and applicability improved as compared to a monolithic implementation. EVE is able to integrate services that comply to a basic interface (`IEveService`). EVE is designed to support command-line as well as distributed service application. Additionally, services can be executed in a transactional chain within a network.

The framework described has been implemented in a project at the TU Berlin. First services of the framework are OCL-based validation and browsing services, which are integrated into existing tools.

References

[1] Christopher Alexander. *Notes on the Synthesis of Form*. Harvard University Press, 1964.

[2] Ken Arnold, Bryan O'Sullivan, Robert W. Scheifler, Jim Waldo, and Ann Wollrath. *The Jini Specification*. Addison-Wesley, 1999.

[3] T. J. Biggerstaff and C. Richter. Reusability Framework, Assessment, and Directions. In T. J. Biggerstaff and C. Richter, editors, *Software Reusability*, volume I — Concepts and Models, chapter 1, pages 1–17. acm press, 1989.

[4] Jr. Frederic P. Brooks. *The Mythical Man Month*. Addison-Wesley, anniversary edition, 1995. ISBN: 0-201-83595-9.

[5] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture: A System of Pattern*. John Wiley & Sons, 1996.

[6] Ivica Crnkovic, Brahim Hnich, Torsten Jonsson, and Zeynep Kiziltan. Specification, implementation, and deployment of components: Clarifying common terminology and exploring component-based relationships. *Communications of the ACM*, 45(10):35 – 40, October 2002.

[7] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns, Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[8] Roxana Giandini, Claudia Pons, and Gabriela Alejandra Pérez. Use case refinements in the object oriented software development process. In Alfredo Viola, Mariana Lasarte, Daniel Perovich, Martin Solari, and Andres Vignaga, editors, *XXVIII Conferencia Latinoamericana de Informatica (CLEI'02), InfoUYclei 2002*, November 2002. ISBN 9974-7704-1-6.

[9] Aniruddah Gokhale, Douglas C. Schmidt, Balachandran Natarajan, and Nanbor Wang. Applying model-integrated computing to component middleware and enterprise applications. *Communications of the ACM*, 45(10):65 – 70, October 2002.

[10] Par Jansson. *Use Case Analysis with Rational Rose*. Rational Software Corp, Santa Clara CA, 1995.

[11] Ora Lassila and Ralph R. Swick. Resource description framework (rdf) model and syntax specification. `http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/`, February 1999. W3C Recommendation 22 February 1999.

[12] Rami Marelly, David Harel, and Hillel Kugler. Multiple instances and symbolic variables in executable sequence charts. In *Proc. 17th Ann. ACM Conf. on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA'02)*, 2002.

[13] M.D. McIlroy. Mass produced software components. In P.Naur and B. Randel, editors, *NATO Conference on Software Engineering*. NATO Science Commitee, Oktober 1968.

[14] Stephen J. Mellor. Make models be assets. *Communications of the ACM*, 45(11), November 2002.

[15] J. Miller and J. Mukerji. Model driven architecture(mda). Technical Report ormsc/2001-07-01, Object Management Group(OMG), Architecture Board ORMSC, July 2001.

[16] H. Müller and H. Weber, editors. *Continuous Engineering of Industrial-Scale Software Systems*, March 1998. Proc. Dagstuhl Seminar.

[17] Object Management Group (OMG). *Unified Modeling Language Specification, Version 1.3*, March 2000. http://cgi.omg.org/docs/formal/00-03-01.pdf.

[18] Object Management Group (OMG). *Unified Modeling Language Specification, Version 1.4*, September 2001. http://cgi.omg.org/docs/formal/01-09-67.pdf.

[19] Object Management Group (OMG). *Meta Object Facility(MOF) Specification*, April 2002. Version 1.4, http://www.omg.org/.

[20] Object Management Group (OMG). *XML Metadata Interchange(XMI) Specification*, January 2002. Version 1.2,http://cgi.omg.org/docs/formal/02-01-01.pdf.

[21] Mary Shaw and David Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. PH, April 1996. ISBN 0131829572.

[22] Perdita Stevens. Playing games with uml tools. http://www.dcs.ed.ac.uk/home/pxs/talksEtc.html, November 2002. Invited Talk, First International Symposium on Formal Methods for Components and Objects (FMCO2002).

[23] Jos B. Warmer and Anneke G. Kleppe. *The object constraint language: precise modeling with UML*. Addison-Wesley, 1999. ISBN 0-201-37940-6.

[24] Herbert Weber. Continuous engineering of information and communication infrastructures (extended abstract). In Jean-Pierre Finance, editor, *Proceedings of Fundamental Approaches to Software Engineering, FASE'99, Amsterdam*, volume 1577 of *LNCS*, pages 22–29, March 22-28 1999. ISBN 3-540-65718-5.

Minutes of the EASST General Assembly

April 10, 2003, Warsaw
Version May 22, 2003

Prof. H. Ehrig, Prof. H. Weber

The EASST General Assembly was announced in the EASST Newsletter and took place on April 10 from 12:45 to 13:30 at the ETAPS conference site in Warsaw. The General Assembly was attended by about 25 EASST members and chaired by the EASST President Herbert Weber with the following topics:

TOP 1 Report of EASST President

EASST is a registered non for profit association under German law. As such an association it enjoys a number of preferences, especially, tax preferences. EASST may receive donations that are exempted from taxation - even more, donors may deduct the amount donated from their taxable income. To get there was a rather long procedure but it was worth it, because EASST may now develop activities and seek sponsorship that are at least to some extent of advantage to sponsors.
In order to achieve this, it was necessary to change the charter of EASST to include the promotion of education as one of its goals. A respective decision of the General Assembly has been taken.
A number of other facts may be of interest:
The EASST association has now 167 registered members.
EASST has sponsored "Best Paper Awards" during the last three ETAPS conferences. The award money was donated by the Fraunhofer Institute for Software and Systems Engineering.
Some of the EASST members have engaged themselves in the preparation of an Integrated Project under the 6.FP of the European Union on "E-Learning".

TOP 2 EASST Newsletter

There have been 5 volumes of the EASST-newsletter, the first in June 2000, then in February 2001, in December 2001, in August 2002 and the last in March 2003.

The schedule of the EASST-newsletter is going to be more fixed. There will be two volumes a year. The first in beginning of June, the second in the beginning of December. There will be a call for contributions a month earlier. This call is send to all EASST-members via email. This way we hope to resume discussions that occurred during FASE and to stimulate the EASST-members to participate in those discussions.
The Newsletter has established columns for vital topics of the EASST community. These columns are edited by the column editor. He or She can initiate discussions, present specific opinions on that matter or present interesting papers in that area.
There are by now the following columns:

- The Education column by H. Weber
- The Tool Column by T. Margaria
- The Software Architecture Column by M. Wermelinger

TOP 3 EASST Future Perspectives

The further development of EASST should follow a combined action of continuity in the line established by the current Board and start of new initiatives, primarily directed at outreach and visibility.

Future activities that continue the line of interests and initiatives followed up to now concern the establishment of EASST as a reference point for the discussion and promotion of technical issues about the development and use of well-founded techniques and methodologies in hot areas of software and system engineering. Current examples are
- system-level design and modeling, and the establishment of a culture of quality assurance for these systems
- increased attention to the support of software integration concerns, that involve e.g. legacy systems or underspecified systems
- improving techniques aimed at the support for system maintenance and system evolution.

Following the successful line adopted so far, EASST will continue to support international conferences (e.g., FASE, ICGT), strengthen the dialogue with other societies with a potential for cooperation, and exploit synergies in order to reach these goals.
In particular, EASST will sponsor new, young events with a driving force (workshops, meetings, ...) and events more closely related to the potential target of industry professionals.

These events and initiatives should be supported in a low-budget way, e.g. through "in cooperation with" status, best paper awards, and qualification certificates. This will increase the visibility to EASST, strengthen its influence, and help establishing a community of similar minds, bridging between academia and industry and between theoreticians and practitioners.
This goal is also supported by EASST´s education-related initiatives (see Top. 5), and by the new, informal ways of communication (like e.g. the specific columns that enrich the EASST Newsletters), to address specific topics and a specific public: EASST is not only for scientists and experts, it also addresses a wider public of practitioners, which must be adequately reached and prepared.

TOP 4 Election of new EASST Board

The General Assembly has been asked by the chairman to nominate at least 8 candidates for the election of the new EASST Board. In return the assembly has asked the present EASST Board members whether they are willing to continue. H. Weber and H. Ehrig pointed out that they want to step down from their positions as EASST President and EASST Vice President respectively, but that they are willing to serve in the EASST Board for another period of 1-2 years, if this is wanted by the other EASST Board candidates. T. Margaria has been asked by the old EASST Board to become candidate for the new EASST Board, and especially also for the new EASST President. She has accepted under the condition that H. Weber and H. Ehrig are willing to continue as Board members at least for a period of 1-2 years. The other old EASST Board members M.-C. Gaudel, J. Padberg and W. Wermelinger have also accepted to be candidates for the EASST Board under these conditions. Furthermore E. Astesiano is again nominated as candidate for the EASST Board. Although he was not able to be present at the assembly, it is assumed that he would accept to continue. Finally P. Pepper, another old EASST Board member, had declared not to be candidate any more. For this reason T. Margaria proposed S. Graf as new candidate after making sure that she would accept. The General Assembly was asked again for further candidates but no additional candidate was nominated. After this nomination procedure the General Assembly has elected with no objections and one abstention the following new EASST Board members (in alphabetical order):

Egidio Astesiano (Genova)
Hartmut Ehrig (Berlin)
Marie-Claude Gaudel (Paris)
Susanne Graf (Grenoble)
Tiziana Margaria (Dortmund)
Julia Padberg (Berlin)
Herbert Weber (Berlin
Michael Wermelinger (Lisbon).

All the new members of the EASST Board have accepted the election either during or after the meeting.

In the subsequent Board meeting the new EASST Board has elected the following Board members for the following positions:

EASST President: Tiziana Margaria
EASST Vice President: Hartmut Ehrig
EASST Secretary: Julia Padberg
EASST Treasurer: Herbert Weber.

TOP 5 Modification of EASST Statute

H. Weber proposes to modify the EASST statute in the following way in order to add the goal of education:

§ 2, Zweck des Vereins, Abs. 1 hat bisher folgenden Wortlaut:
1. Der Zweck des Vereins ist die Förderung von Forschung, Entwicklung und Anwendung zum systematischen und rigorosen Engineering von Software und softwareintensiven Systemen.

§ 2 Abs. 1 lautet nunmehr wie folgt:
1. Der Zweck des Vereins ist die Förderung der Bildung, insbesondere die Vermittlung von Informationen über die Forschung, Entwicklung und Anwendung zum systematischen und rigorosen Engineering von Software und softwareintensiven Systemen durch die Erstellung und Verbreitung von regelmäßig erscheinenden virtuellen Informationsblättern, die Veröffentlichung und Rezension von Fachliteratur, die Organisation und Durchführung von Ringvorlesungen und Vorträgen.

This proposal is accepted unanimously by the General Assembly.

FASE 2004

Seventh International Conference on Fundamental Approaches to Software Engineering
http://ctp.di.fct.unl.pt/~mw/conf/fase04

Barcelona, Spain, March 29–31, 2004

An ETAPS main conference supported by EASST

INVITED SPEAKER

Gruia-Catalin Roman
Univ. of Washington at St. Louis, USA

IMPORTANT DATES

Submission October 17, 2003
Notification December 12, 2003
Final version January 9, 2004

Authors who anticipate possible last minute delays in the submission of a full paper are encouraged to submit a title and abstract well in advance of the deadline.

For any matter concerning FASE (except for submitting papers and abstracts) please contact the PC chairs at fase04@di.fct.unl.pt.

PROGRAMME COMMITTEE

Ralph-Johan Back, Åbo Akademi, Finland
Jean Bézivin, Univ. Nantes, France
Maura Cerioli, Univ. di Genova, Italy
Matthew Dwyer, Kansas State Univ., USA
Reiko Heckel, Univ. Paderborn, Germany
Constance Heitmeyer, Naval Research Laboratory, USA
Heinrich Hußmann, Dresden, Germany
Dan Craigen, ORA, Canada
Serge Demeyer, Univ. Antwerp, Belgium
John Fitzgerald, Univ. Newcastle upon Tyne, UK
David Garlan, Carnegie Mellon Univ., USA
Antónia Lopes, Univ. de Lisboa, Portugal
Jeff Magee, Imperial College, UK
Tiziana Margaria, Univ. Dortmund, Germany (co-chair)
Tom Mens, Vrije Univ. Brussel, Belgium
Mauro Pezzè, Univ. Milan, Italy
Gian Pietro Picco, Politecnico di Milano, Italy
Ernesto Pimentel, Univ. de Málaga, Spain
Gabriele Taentzer, Technische Univ. Berlin, Germany
Michel Wermelinger, Univ. Nova de Lisboa, Portugal (co-chair)

Large scale Information and Communication Infrastructures are of growing concern to industry and public organizations. They are expected to exist indefinitely long, are supposed to be flexibly adjustable to new requirements and are hence demanded to encompass evolvable software systems. Quality is increasingly important in classic as well as new application domains. This poses new challenges to software engineering research and practice: new software structuring and scaling concepts are needed for heterogeneous software federations that consist of numerous autonomously developed, communicating and inter-operating systems; new software development processes are needed to enable the continuous improvement and extension of heterogeneous software federations. New quality assurance methods are needed to guarantee acceptable standards of increasingly complex software applications. Different component paradigms are under discussion now, a large number of specification and modeling language are proposed and an increasing number of software development tools and environments are made available to cope with the problems. At the same time research on new theories, concepts and techniques is under

way that aims at the development of their precise and (mathematically) formal foundation.

Contributions are encouraged that aim at both pragmatic concepts and their formal foundation that can lead to new engineering practices and a higher level of reliability, robustness, and evolvability of heterogeneous software federations. Especially sought are submissions on:

- Component-based software architectures: design methods and strategies, design patterns, quality assurance
- Systematic (in particular coordinative) approaches towards change management in large scale systems, reverse engineering, and improvement of legacy systems
- Rigorous approaches to the design, test, and maintenance of reactive, mobile, and distributed software systems
- Integration platforms and middleware systems for large scale heterogeneous software federations
- Requirements engineering: techniques for acquiring, modeling, specifying and analyzing software components
- Analysis, Verification, and Testing: algorithms, techniques, and processes concerned with assuring, developing, or assessing software with respect to requirements or goals
- Integration of (lightweight) formal concepts and current best practices in industrial software development
- Experience reports on best practices with component models and specifications, development tools, modeling environments, and software development kits

Submission Details

FASE accepts two types of contributions. Only the programme committee chairs are not allowed to submit to FASE.

Research papers

Prospective authors are invited to submit full papers in English presenting original research. Submitted papers must be unpublished and not submitted for publication elsewhere. In particular, simultaneous submission of the same contribution to multiple ETAPS conferences is forbidden.

Papers must be submitted in PostScript or PDF format. Submissions in the format of any specific text processing system such as Latex, MS-Word, Adobe-Framemaker, or any other proprietory format cannot be accepted. Papers are to be submitted electronically, through an on-line system to be made available closer to the submission deadline.

The proceedings will be published in the Springer-Verlag Lecture Notes in Computer Science series. Final papers will be no more than 15 pages long in the format specified by Springer-Verlag at

http://www.springer.de/comp/lncs/authors.html. It is recommended that submissions adhere to that format and length. Submissions that are clearly too long may be rejected immediately. E-mail addresses and fax numbers of the authors should be included on the title page.

Tool demonstration papers

Demonstrations of novel and state-of-the-art tools are also invited. A submission should have a clear connection to FASE, possibly complementing a paper submitted separately.

Tool demonstrations are an integrated part of the ETAPS programme. Selected demonstrations will be presented in ordinary conference sessions, using state-of-the-art projection. The time allowed will be approximately the same as that for the presentation of a research paper. The demonstration will be accompanied by the publication of a short paper (up to 4 pages) in the proceedings of FASE, describing the main features of the tool.

Submissions should take the form of a self-contained tool description of no more than 4 pages in the same format as reserach papers. The tool description should be accompanied by an appendix (not intended for publication, and not included in the page limit) indicating which features of the tool would be demonstrated — preferably with some sample screen snapshots — followed by a detailed specification of the hardware, software, and licensing requirements for installing and using the tool.

N.B. Tool demonstrations should not be confused with research contributions to the TACAS conference, which emphasizes principles of tool design, implementation, and use, rather than focusing on specific domains of application.

CALL FOR PARTICIPATION

The 12th International FME Symposium

Pisa, Italy – September 8-14, 2003

http://fme03.isti.cnr.it - fme03@isti.cnr.it

General Chair
Stefania Gnesi, ISTI-CNR, Pisa, Italy
gnesi@iei.pi.cnr.it

Program Co-Chairs
Keijiro Araki, Kyushu Univ., Japan
araki@csce.kyushu-u.ac.jp
Dino Mandrioli, Politecnico of Milan, Italy
mandrioli@elet.polimi.it

Organizing Committee Chair
Alessandro Fantechi, Univ. of Florence, Italy
fantechi@dsi.unifi.it

Publicity Chair
Vinicio Lami, ISTI-CNR, Pisa, Italy
lami@iei.pi.cnr.it

Tool Exhibition Chair
Tiziana Margaria, Univ. Dortmund and Metaframe, Germany
tiziana@ls5.cs.uni-dortmund.de

Tutorial Chair
Mieke Massink, ISTI-CNR, Pisa, Italy
massink@cnuce.cnr.it

Workshop Chair
Tommaso Bolognesi, ISTI-CNR, Pisa, Italy
bolognesi@iei.pi.cnr.it

FM 2003 is the twelfth in a series of symposia organized by **Formal Methods Europe**, an independent association whose aim is to stimulate the use of, and research on, formal methods for software development. These symposia have been notably successful in bringing together a community of users, researchers, and developers of precise mathematical methods for software development as well as industrial users.

Formal methods have been controversial throughout their history, and the realization of their full potential remains, in the eyes of many practitioners, merely a promise. Have they been successful in industry? If so, under which conditions? Has any progress been made in dispelling the skepticism that surrounds them? Are they worth the effort? Which aspects of formal methods have become so well established in the industrial practices to loose the "formal method" label in the meanwhile?

FM 2003 aims to answer these questions, by contributions not only from the Formal Methods community but also from outsiders and even from skeptical people who are most welcome to explain, document, and motivate the source of their reluctance.

Satellite events

FM 2003, will host 7 Workshops, 8 Tutorials and 1 Day dedicated to the Industry besides the 3 days of the FME Symposium. Tool demonstrations will also take place during the symposium, with the opportunity of holding presentations for each tool.

For full details on the Symposium organization see the web site **http://fme03.isti.cnr.it**, or send your query to fme03@isti.cnr.it**.**