Establishing a Software Architecting Environment

Claudio Riva[1], Petri Selonen[2], Tarja Systä[2], Antti-Pekka Tuovinen[1], Jianli Xu[1], Yaojin Yang[1]

[1]*Software Technology Lab, Nokia Research Center*
P.O. Box 407, FIN-00045, NOKIA GROUP, Finland
{claudio.riva | antti-pekka.tuovinen | jianli.xu | yaojin.yang}@nokia.com

[2]*Institute of Software Systems, Tampere University of Technology*
P.O.Box 553, FIN-33101 Tampere, Finland
{petri.selonen | tarja.systa}@tut.fi

Abstract

We present the work of establishing an integrated environment that facilitates architecture design, reconstruction, and maintenance in the entire life cycle of a software product line. This architecting environment (ART environment) has been used in modeling and analysis of both the designed platform architecture model and the reverse-engineered product implementation architecture models of different releases in a big product line of Nokia mobile terminals. ART environment comprises tools for architecture model validation, architecture model analysis and processing, and reverse architecting. The ART environment fits the current software development process inside Nokia, and is integrated with the design and documentation tools that have already been used by Nokia software architects. UML, after being customized with UML profiles for architecture design, is used as the architecture modeling language in ART environment.

1. Introduction

Software architecture is of vital importance not only to the quality of the resulting software system but also to the control and management of the software development process. Different architecture-centric approaches have been adopted in industrial software development; especially in the development of software product-lines [1][2][3][4]. The growing importance of architectures implies an essential change in the general software development paradigm: software development is becoming architecture-centric, in contrast to the traditional code-centric view.

Although the architecture-centric development is getting more and more popular, in practice software architects are still facing big challenges of applying it effectively. Rather than concentrating on making architectural decisions and designs, they have to struggle with many non-architectural factors. Among many others, there are two important factors that prevent the architecture-centric approaches from playing a more decisive role in product-line software development. First, there is no universal architecture description/modeling language (ADL) [5] that can be used in all the domains. Usually an ADL is either dedicated only to a specific architecture style or is too general to model the architectural details of large and complex systems. Second, there is hardly any adequate architecting tool support for the daily work of software architects. In many companies tools like Microsoft Office are used with some kinds of informal notations (graphical or textual) for the description and documentation of software architecture. With such tool support it is very difficult to analyze, modify and maintain the architecture descriptions.

Many people now believe UML [6] is the way to go in software architecture design, because, as a wide-spectrum standard notation, UML allows the use of the same notation in both architectural and detailed design, and it has better tool support comparing with ADLs. It is relatively easy to proceed from a UML architecture design to system and component design, where UML and OO programming paradigm are usually used. However using UML and UML tools cannot really solve the two problems mentioned earlier. On one hand, UML, as a general purpose OO design notation, lacks appropriate specification concepts at the architecture level [7][8]. On the other hand, current UML-based tools lack support for specifying various architectural rules, for checking the conformance of a design against architectural conventions, for constructing new designs according to given architectural rules, for managing variability supported by a product-line architecture, for creating architectural views from design-level models, and for establishing tracing capabilities between implementation and architectural models. All these are crucial tasks in architecture-centric software development. Without such support, UML-based architecting tools remain little more than specialized graphical editors. Both the research from others [7][8] and our experience from the software development practices at Nokia [9] suggest that there is an urgent need for effective approaches and tools to support architecture modeling with UML.

In order to make UML appropriate for describing software architecture in our domain, we customized the UML metamodel with special UML profiles: architectural profiles. A profile is essentially defined by extending the specification of the UML metamodel using UML's built-in mechanisms. An architectural profile is an UML profile that presents a set of architectural rules and specifies a subset of UML models that can be considered "legal" in a particular context. We have developed the technique and tool to validate a given UML model against the architectural profile [10].

In this paper we describe our work at Nokia in establishing an integrated UML based architecting environment – ART environment. ART environment has been setup as the result of a research project at Nokia Research Center in cooperation with the Institute of Software Systems of Tampere University of Technology. ART environment has been used in the architecting work of one of the main mobile phone product lines in Nokia.

The paper is structured as follows. Section 2 gives the overview of ART environment. From section 3 to section 5 we present its three main toolsets: section 3 describes the model analysis and processing toolset; section 4 introduces the reverse architecting toolset and section 5 presents the model repository and its Web interface. In section 6 we show how ART environment has been used in the design and maintenance of the software architecture of a product line. Section 7 draws the conclusion of the paper and discusses some of the future work.

2. Overview of ART Environment

ART environment consists of three main toolsets: the architecture model analysis and processing toolset, the reverse-architecting toolset, and the model repository and its Web interface.

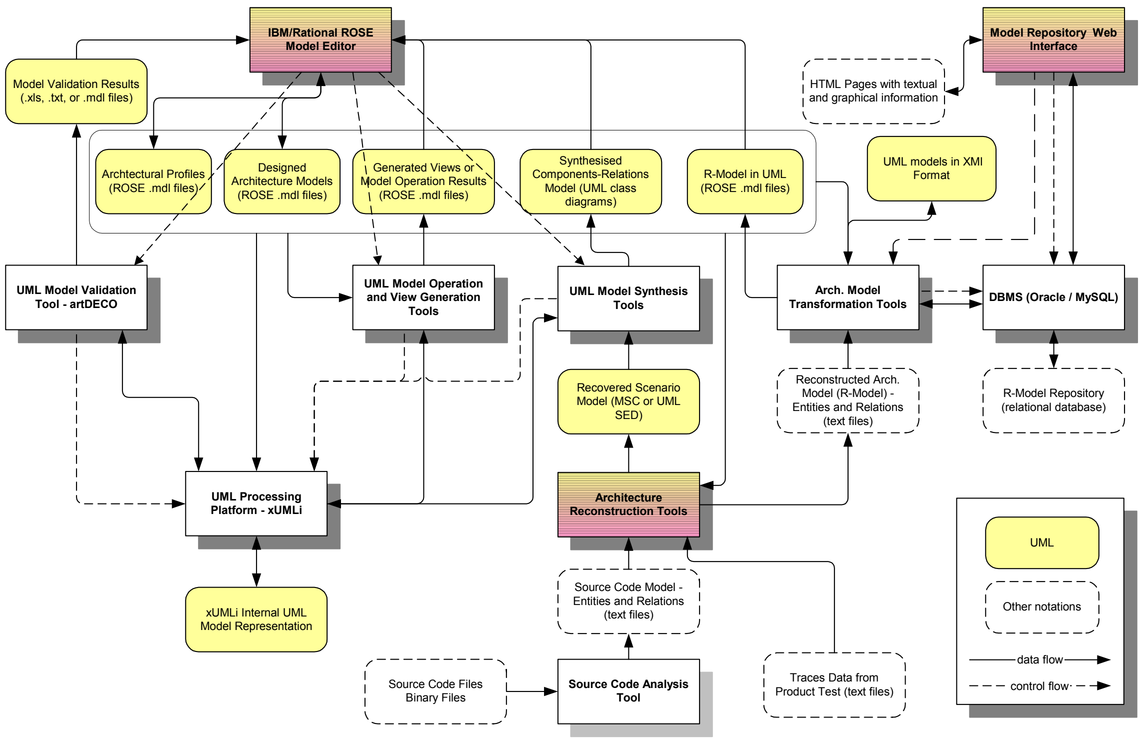
The model analysis and processing toolset works with UML architecture models including not only the models created by architects using a UML CASE-tool (i.e. IBM Rational ROSE[1]) but also the UML models generated by the reverse-architecting tools. It allows the software architects to create the UML architectural profiles and architectural design models, check architectural models against the architectural profiles, generate architectural views at different abstraction levels and from different viewpoints, and finally analyze architectural models and views with adequate UML model operations. The aim of this part is to provide software architects a complete and consistent view of the architecture under design or maintenance, and to help them to make the right architectural decisions.

The reverse architecting toolset is used by the software architects to re-engineer or recover the architectural model from an implementation. A re-engineered or recovered implementation model of the architecture shares the same concepts and same model structure with the designed architectural model, and can provide the same architectural views at the same abstraction levels as the design model does. Hence, the same architectural profiles can be used to check the recovered architectural models to discover any violations in the implementation. The comparison of the recovered model with the designed model can also reveal architecturally significant changes introduced during implementation. For a fast evolving product-line, being able to monitor the changes and keep the evolution under control is extremely important.

The model repository and its Web interface provide an efficient way for managing and retrieving all the architecture model elements. A model repository is necessary when one has to maintain the architectural models of a large number of implementation releases of a product-line. In ART environment we use a relational database to store architectural models. A Web interface to the model repository provides the easy access to the model repository to software architects located in different sites.

Figure 1 gives an overview of all the tools and the data flows among them.

[1]IBM Rational Rose Enterprise Edition. On-line at: http://www-136.ibm.com/developerworks/rational/products/rose



IBM/Rational ROSE Model Editor
Model Repository Web Interface
Model Validation Results (.xls, .txt, or .mdl files)
HTML Pages with textual and graphical information
Archtectural Profiles (ROSE .mdl files)
Designed Architecture Models (ROSE .mdl files)
Generated Views or Model Operation Results (ROSE .mdl files)
Synthesised Components-Relations Model (UML class diagrams)
R-Model in UML (ROSE .mdl files)
UML models in XMI Format
UML Model Validation Tool - artDECO
UML Model Operation and View Generation Tools
UML Model Synthesis Tools
Arch. Model Transformation Tools
DBMS (Oracle / MySQL)
Recovered Scenario Model (MSC or UML SED)
Reconstructed Arch. Model (R-Model) - Entities and Relations (text files)
R-Model Repository (relational database)
UML Processing Platform - xUMLi
Architecture Reconstruction Tools
xUMLi Internal UML Model Representation
Source Code Model - Entities and Relations (text files)
Source Code Files Binary Files
Source Code Analysis Tool
Traces Data from Product Test (text files)
UML
Other notations
data flow
control flow

Figure 1. ART Environment overview

ROSE is one of the main user interfaces of ART environment and the tools that handle UML models are implemented as ROSE add-ins. The software architects can use ROSE to create, edit or view architectural models and profiles, and then invoke the ART UML model analysis and processing tools from ROSE. The internal model representation is only based on UML standard and the UML model processing platform of ART environment is independent of any UML CASE-tools. Therefore ART environment, in principle, can easily switch from ROSE to another UML CASE-tool as its model editing and viewing user interface. The architecture reconstruction tools and the Web interface of the model repository provide the other two user interfaces. The first one allows the architects to start and control the reverse-architecting process, and view the intermediate results. The latter one allows the architect to query any information about the architectural models in the repository database using a Web browser; the results are shown in a textual format or in different graphical formats.

ART environment is designed to be reconfigurable and modifiable. When the environment is deployed in the development process of different product lines, the tools can be tailored and re-configured for the specific properties of each product line. For example, the rules and patterns for extracting models from source code are different for an OO programming language implementation than for a C language implementation; different architectural profiles require different model validation rules; a product line may decide to use a different UML CASE-tool than we originally considered or even does not use any UML CASE-tool. All these factors should be considered when deploying the environment to a specific product line.

3. Model Analysis and Processing Toolset

3.1. UML Model Validation Tool

An UML model validation tool, artDECO, is used for validating architecture views against given profiles. It is a collection of Python[2] components implemented on top of xUMLi [11], each representing a particular set of conformance and auxiliary operations [10], and VISIOME [12] scripts describing configurations of the conformance operations. The conformance operations together with a set of architecture profiles effectively specify an architecture description language for the selected domain.

An artDECO script uses xUMLi to import the architectural profiles and views from ROSE, and then executes the conformance operations one by one, each

[2] http://www.python.org

validating the views against the profiles focusing on a particular validation aspect. The results are reported to the user, either in the form of an XML error report, or by providing a visual browser that can be used with ROSE. Conceptually, the architectural profiles are interpreted as standard UML profiles, imposed on the views in *a posteriori* fashion.

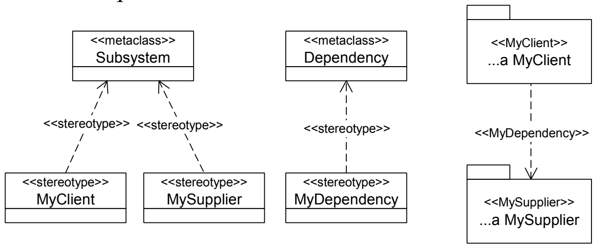


Figure 2 Example of an architectural profile

As an example of using the UML model processing capabilities of xUMLi, consider querying for the valid stereotypes as shown in the left-hand side of Figure 2. In the framework, offering a UML interface supporting COM automation and an interpreter for Object Constraint Language (OCL) [6], the operation is performed on classifiers as follows [13]:

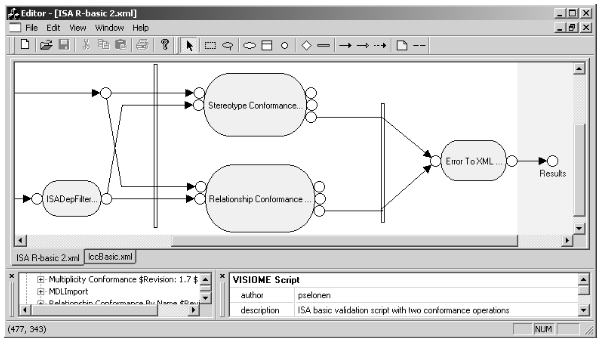
```
st = profile.find(\
 "element.oclAsType(Class).clientDependency->exists(cd |" \
 "cd.stereotype.name->includes('stereotype')" \
 "and cd.supplier.stereotype.name->include("metaclass") )" )

for cls in model.find("element.oclIsKindOf('Classifier')"):
  for cst in cls.stereotype:
    if len(st.select("element.name = '" + cst.name + "'"))==0:
      # handle class with wrong stereotype
```

The first part of the operation queries for all properly defined stereotypes in the stereotype definition profiles, while the latter iterates over all classifiers in the view and checks whether they have a valid stereotype.

An example of architecture profiles is shown in Figure 2. The left-hand side of the figure introduces three new architectural types: MyClient, MySupplier, and MyDependency. The right-hand side of the figure shows a new constraint imposed on the instances of MyClient and MySupplier: there can only be a dependency of type MyDependency from a MyClient to a MySupplier. Further examples of the conformance operations, validation configurations, and architectural profiles are given in [10].

Screenshot in Figure 3 shows a part of an artDECO configuration opened in the VISIOME editor. The script involves four activities (one auxiliary operation for filtering the views, two conformance operations, and an error to XML conversion operation), and two synchronization bars. The script has two inputs, one for architectural profiles and one for architectural views, and an output for the reported incidents.



Editor - [ISA R-basic 2.xml]
File Edit View Window Help
Stereotype Conformance...
ISADepFilter...
Relationship Conformance ...
Error To XML ...
Results
ISA R-basic 2.xml
IccBasic.xml
Multiplicity Conformance $Revision: 1.7 $
MDLImport
(477, 343)
NUM

VISIOME Script

author	pselonen
description	ISA basic validation script with two conformance operations

Figure 3 Example of an artDECO configuration

3.2. UML Model Operation Tool

As a part of the environment, a set of UML model processing operations [14] is introduced for analyzing and manipulating the architecture models. A profound example of a model operation is searching for, and filtering of, information in given UML models. More expressive UML model operations usually produce new UML diagrams on the basis of existing UML diagrams, or modify the existing ones. Examples of model operations are transformation operations, projection operations, and set operations.

A transformation operation takes a (set of) UML diagram(s) as its input operand and based on the information implied by this diagram produces a new UML diagram of another type. A set operation takes two UML models as its input operands and produces a new UML model (e.g. merging and slicing of diagrams). A projection operation produces a new UML model based on an existing one, the new model being a projection of the original one (e.g. abstraction or slicing). Other useful operation categories include visualization operations (e.g. highlighting, layout) and user interface operations (e.g. graphical dialogs). A subset of the abovementioned operations have been implemented as xUMLi components and used as a part of the architecture validation process described in this paper.

In particular, the set operations have been applied on both comparing consecutive versions of the reverse engineered architecture models, and comparing reverse engineered architecture models against forward engineered ones. The models, stored in ROSE repositories, are imported into xUMLi platform. The corresponding concepts are then derived using information about e.g. names, metaclasses, stereotypes, container namespaces, and relationship end elements.

The results, namely the common and disjoint parts, are reported both in a textual format and in ROSE using summary diagrams and color highlighting.

As part of the model analysis and processing tools, a set of ROSE scripts (as ROSE add-ins) are developed to automatically integrate the subsystem models into a complete and consistent system model in ROSE. Because there are tens of subsystems and interfaces and hundreds of components, it is difficult to create by hand views that would show inter-subsystem dependencies derived from the component-to-interface dependencies specified in the subsystem units. There is a need for these kinds of views when creating platform architecture descriptions (that show domains larger than a single subsystem) and when checking that architectural rules are followed (who is allowed to use what).

The results created by the scripts are recorded directly in the model, which facilitates the use of the information by other ART tools that can use Rose models as input. The scripts were created as ROSE scripts due to schedule and resource constraints. We plan to make a full xUMLi implementation later to replace the ROSE scripts.

3.3. UML Model Synthesis Tool

Of the model operations described in Section 3.2, transformation operations can be also seen as model synthesis operations. They are particularly useful when used together with the set operations. For instance, to check whether a given sequence diagram is in agreement with an architectural model represented as a class diagram, the sequence diagram can be transformed into another class diagram, and they can be compared using the set operations. The reported differences can then be examined to conclude whether or not the original sequence diagram implied e.g. non-existing operations or relationships between participating classifiers. The transformation simply maps the classifier roles and messages of the sequence diagram to classes, associations and operations of a class diagram. In addition, a collection of heuristics can be applied, for example, to generate interface hierarchies, composition relationships, and multiplicities [15].

3.4. xUMLi – UML Processing Platform

Many UML-based CASE-tools provide APIs allowing external components to access and even modify the model data. However, they fall short in providing stronger support for advanced UML model processing activities and typically have only limited support for UML metamodel. To overcome this problem, xUMLi [11], a CASE-tool independent software platform has been developed for UML model processing. It allows users to build arbitrary UML model operations and to combine them using the VISIOME scripting mechanism to contruct more complicated operations, examples of which are given in Sections 3.1, 3.2 and 3.3.

The platform consists of VISIOME, a UML specialization layer, and a set of standard components. In general terms, xUMLi can be seen as a middleware meant for model processing purposes. The environment is not dependent of any specific CASE-tool, but offers a plug-in interface for components that transfer models between a tool repository and the data model. It is therefore possible to support several different CASE-tools or UML model repositories. Currently xUMLi is integrated with ROSE.

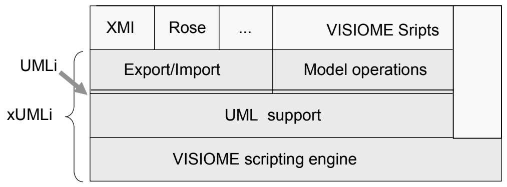


Figure 4. The layered architecture of xUMLi

The platform can be conceptually understood as a layered architecture, as depicted in Figure 4. The lowest layer consists of a scripting engine providing domain-independent, general data model, and an OCL interpreter. The second lowest level specializes the general data model and provides access to UML models according to the UML metamodel. This layer is used through a UML interface (UMLi) which serves as the basic API for xUMLi. Various components have been implemented on top of xUMLi, including export and import components for bridging other UML tools and model processing operations.

UMLi has been implemented using the COM component model with automation support, enabling the usage of any language supporting COM automation (e.g. Python, Visual Basic). This kind of a general interface has turned out to be a valuable asset especially for exploratory, research-oriented implementation of various advanced UML processing capabilities. In particular, the Python components can be written on a high conceptual level, without deep understanding of the underlying implementation.

4. Reverse Architecting Toolset

The reverse architecting toolset provides the latest architectural information from the implementation of various products of the family. The goal is to extract the elements and their dependencies according to the profile defined. The reconstruction process takes into account the source code for identifying the low-level software dependencies and the domain information for the structural and behavioral aspects [16].

Simplicity, ease of use and modifiability are the design philosophy of the toolset. The reverse-engineering data is represented with a relational format and stored in textual files with RSF (Rigi Standard Format) [3]. We specify the operations on the dataset with the relational algebra and we use the Python scripting language for programming the reconstruction process. Our reconstruction approach also provides support for combining static and dynamic views of the recovered architecture models [18].

4.1. Source Code Analysis Tools

The source code analysis tools detect the architecturally significant elements from the implementation by searching the source code for particular code patterns. The tools traverse the code files and detect the code patterns defined by regular expressions. The output of the analysis is the *source code model,* a relational dataset containing the low-level system dependencies. The following table shows an example of the relations in the *source code model*:

file	*sendMessage*	msgType
file	*call*	function
file	*defineMsgEvent*	msgType
file	*defineFunction*	function

The various members of the product family are parsed by the Python scripts and the results are merged by concatenating the files. The final result is a super set of all the low-level dependencies. Depending on the implementation language of the system, we can complement the data from the Python scripts with the other dependencies from other source code analyzers such as RedHat SourceNavigator[4] and Columbus[5].

One key aspect is that the meta-model of the tools is not bound to a particular schema but it can be easily adapted to the system under analysis in order to extract those relations that are architecturally relevant.

4.2. Architecture Reconstruction Tools

[3] http://www.rigi.csc.uvic.ca/

[4] http://sources.redhat.com

[5] http://www.frontendart.com

The architecture reconstruction tools consist of three parts: the architectural knowledge base, the static view reconstruction tool, and dynamic view reconstruction tool. The static view reconstruction tool creates the structural views at various abstraction levels from the *source code model*. The dynamic view reconstruction tool builds the high-level behavior views from test or simulation traces. The static and the dynamic view reconstruction tools are integrated in the same reverse-engineering environment and they can share the same knowledge base and same abstraction rules [17].

4.2.1. Architectural knowledge base for reconstruction

The source code model is not sufficient for creating meaningful architectural views but we need to complement it with other domain knowledge (not directly available in the source code): subsystem decomposition, component ownership, mapping between components and source files, mapping between components and run-time objects, and organization of components across sites and projects. This knowledge is essential to the architecture reconstruction tools to reconstruct both the static and dynamic views.

The architecture profile and the platform architecture model define the content of the knowledge base. However, the mappings between the logical entities (e.g. components) to the elements in the *source code model* (e.g. code files and run-time objects) need to be recorded manually by interviewing the experts or from the design documentation. We formalize this additional information in a set of textual tables and we convert them to a relational dataset of the same format like for the source code model. The following table shows an example of the additional architectural relatoins:

component	*compContainFile*	file
package	*pkgContainComp*	component
package	*pkgContainPkg*	package
site	*siteOwnComp*	component

The maintenance of the tables is carried out by the software architects of the product family.

4.2.2. Static view reconstruction tool

The static architectural view reconstruction tool elaborates all the architectural knowledge and creates the static architectural views: *component view*, *development view* and *organizational view*. The main function of the tools is abstraction. The abstraction process consists of a sequence of operations specified in relational algebra. For each architectural view we define the hierarchical structure in terms of subsystems/components and calculate the high-level dependencies by propagating the low-level dependencies to the higher levels.

The output of the whole process, the *RE-model*, is a relational dataset and is the base for the conversion to other formats like DOT[6], SVG[7], SQL, UML. The following table shows an example of the additional relations that are created in the *RE-model*:

component	*compDefineInterface*	interface
component	*compUseInterface*	interface
file	*fileUseInterface*	interface

4.2.3. Dynamic view reconstruction tool

The dynamic architectural view reconstruction tool generates high-level behavior views of the system from the run-time information. A behavior view is presented as set of UML sequence diagrams or MSCs [17].

The extraction of dynamic information is conducted in three steps: first instrumenting the source code, then executing a set of usage scenarios, and finally collecting the traces. The traces collected during test and simulation are converted into a trace relationship in the form of (event, sender, receiver, a label, time stamp). Two special events, 'sc_start' and 'sc_end', delimit the begin and end of a scenario. The initial MSCs of the dynamic view are created from the traces of selected scenarios.

The dynamic view reconstruction tool creates the MSCs that match the same abstraction levels of the static views. Vertical and horizontal abstraction techniques [18], such as message and participant compression, hierarchical grouping, are used to cope with the complexity of the generated MSCs. So the tool can present human readable and analyzable MSCs to the architects. The output MSCs are input to the UML model synthesis tool for creating other type of models.

5. Architecture Model Repository

The development teams need an easy access to the latest architectural information. We regularly publish the models in the intranet through a web interface and other visualization formats. This section describes the central component repository and the various presentation formats.

5.1. Model Repository and Its Web Interface

The role of the model repository is to store the relevant architectural information in a relational database. The ART environment supports the MySQL[8] database and Oracle[9] database. The queries are defined using SQL. The following table shows an example of the tables that are stored in the database:

components:	general information of components
compInterface:	Interfaces and interface types provided by the components
fileComponent:	allocation of files to logical components
compDependency:	High-level dependencies among components
fileUseInterface:	File-level dependencies to the interfaces of the components
package:	Structure of the package hierarchy
packageDependency:	High-level dependencies among packages

The web interface is an alternative representation of the architectural model that provides a distributed access to the information and more detailed textual descriptions, which are weakly supported by the UML modeling tools.

The web interface provides a standard interface for querying the model and dynamically generating the architectural information pages in HTML. The user can query the model according to various criteria: by package, by type, by component or simply grepping the whole model.

For each element in the database (e.g. component, package, interface), the web interface can generate a page that contains a short summary (e.g. name, owner, type, clients and suppliers of the component) and list of hyperlinks to more detailed information (e.g. the list of files that implement a particular interface). The summary page contains also a link to various diagrams that can be dynamically generated. All the elements in the pages are hyperlinked, so the user can follow naturally the chains of dependencies.

The process of generating the web pages is based on a set of CGI scripts that retrieve the data from the repository and generate the textural information and the diagrams. The diagrams are created directly in SVG or by DOT.

5.2. Architecture Model Transformation Tools

The transformation tools convert the RE-model in RSF format to other formats like DOT, SVG, SQL, and UML. The conversion to UML serves as an interface between the reverse architecting tools and the model analysis and processing tools (see Figure 1). The conversion to DOT and SVG is called by Web interface tools for generating embedded diagrams in HTML pages. The conversion to SQL is used for

[6] http://www.research.att.com/sw/tools/graphviz
[7] http://www.w3c.org

[8] http://www.mysql.org
[9] http://www.oracle.com

accessing the model repository. Each conversion is conducted by a dedicated tool.

The UML conversion is achieved by a process called UML encoding [19]. The process consists of two steps: concept conversion and model generation. The whole process of UML encoding is fully supported by a tool, which is implemented in TCL[10] and seamlessly integrated with the reverse-architecting tools.

The SQL conversion facilitates a data connection between the model repository and the reverse-architecting tools. It converts the data format between repository schema and RSF. Data can also be retrieved from the repository based on a certain configuration of the data filter. A tool implemented in Python is used in ART environment for doing the SQL conversion.

6. Application of ART environment

ART environment has been used in the architecture design and maintenance of one of the largest product line of Nokia mobile terminals based on the same OS and middle-ware platform. The creation of ART environment was largely motivated by the challenges in the architecture design and maintenance tasks from the software development of this product line.

In the early stages of the product line development, the architecture group (a group of product architects headed by a chief architect) created a platform architecture based on the architecture designs of the first several products. The group also defined a reference architecture for the product line. The reference architecture is a kind of meta-model that defines the unique architectural style of the product line, component types, the right way of inter-component communications, the necessary views of product architecture descriptions, etc. The platform architecture was described with a set of Excel sheets and Visio UML diagrams, and it was called "the big picture" of the product line by the development teams. The reference architecture was described in an MS-Word document with some UML diagrams.

The reference architecture and "the big picture" guided well the design and development of new products in the beginning. But when the scale of the product line grew, the development organization became global, more unanticipated new features were added, bigger time-to-market pressure came from the hard competition, and so on, it was unrealistic to have all necessary changes (even the architecture significant ones) go through "the big picture". After this situation lasted for some time no product architect could clearly answer what was the architecture of an implementation product and how different it was from the original architecture design, and if the principles defined in the reference architecture still hold in the implementations.

We tackled those problems in the following steps using the tools from ART environment:

1. Creation of the architectural profiles of the product line based the reference architecture document. The concepts, such as types of components and types of dependencies between components, were used to map the implementation elements to architecture components by the reverse architecting toolset in step 4.
2. Transformation of "the big picture" to a complete UML model (called F-model, means the forward engineered model) in ROSE with the model transformation tool.
3. Reconfiguration of the UML model validation tool (artDECO) according to the specific validation rules imposed by the profiles, and tested the validation tool with the F-model.
4. Recovery of the implementation architecture of one product release. The model was saved in the database and all information about the model could be retrieved via the Web interface of the database. The model was also transformed to a ROSE UML model (called R-model, means reconstructed model).
5. Validation of the R-model against the architectural profiles using artDECO tool. The tool generated detailed error reports about all the violations been found. Compared the R-model with the F-model using the model analysis and processing tool, the tool also generated a report on all the differences found.
6. Discussion about the results from the above steps in the architecture group and checked the error reports with the corresponding development teams.

The process and tools were fine tuned by repeating all the steps on the same implementation release. When we checked the R-model of one implementation release against the architectural profiles, the number of violations found from the R-model was surprisingly high. We used the model operation tool as well to compare the F-model and a R-model. The results revealed many changes in the architecture implementation, for example, there are many unanticipated interactions between some components that have no direct designed relations among them.

Another interesting experiment was using the UML model synthesis tool to generate a component-

[10] http://www.tcl.tk/

relationship model (class diagrams) from the system testing traces (sequence diagrams) of a set of scenarios. We compared the generated model with the R-model using the model operation tool, and discovered that a few dependencies in this model are not in the R-model. This fact signals the limitation of static analysis based reverse-architecting techniques. For instance, when pointers and dynamically loadable classes are used in the implementation, the actual dependencies imposed by them can hardly be found by static analysis of the source code.

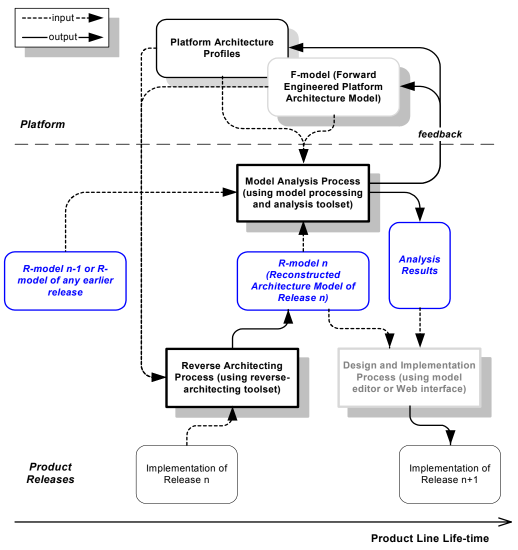

input
output
Platform Architecture Profiles
F-model (Forward Engineered Platform Architecture Model)
Platform
feedback
Model Analysis Process (using model processing and analysis toolset)
R-model n-1 or R-model of any earlier release
R-model n (Reconstructed Architecture Model of Release n)
Analysis Results
Reverse Architecting Process (using reverse-architecting toolset)
Design and Implementation Process (using model editor or Web interface)
Product Releases
Implementation of Release n
Implementation of Release n+1
Product Line Life-time

Figure 5 ART Environment in the product-line process

ART environment has proved to be necessary and practical to support the product line development and guarantee the right evolution of it. Now we are integrating ART environment with the system development process of the product-line. Figure 5 shows how the tools from ART environment are used to support the evolution of the product-line architecture in a simplified process.

7. Conclusion and Future Work

In this paper we have presented an integrated architecting environment - ART environment. The environment, using UML as the architecture modeling language, facilitates the software architecture design, architecture model analysis and processing, architecture model reconstruction and maintenance, during the entire life cycle of a software product-line. By establishing the ART environment, we attempt to tackle the two important problems in software architecture work mentioned earlier in the paper: language and tool support. UML, customized with domain specific architectural profiles, is used as the architecture description/modeling language. This allows us: 1) to use the same notation, UML, which has already been widely used as a program design language, in both architectural and detailed design; 2) to use the best available UML CASE-tools in architecture modeling, at least the model editing, UML syntax checking, and model management functions of those tools; and 3) most importantly, to give UML more precise semantics for architecture modeling by using architectural profiles, so that the profile based architecture model validation tool can be used to check the architecture design.

ART environment provides strong and efficient tool support to software architecture design and maintenance in the context of large product-line development. ART environment has been used intensively in the architecture design and maintenance task of a main product-line of Nokia mobile terminal products, and has also been partly applied in another product-line. The results achieved so far are significant to the further development of the product-lines. We have already started the large-scale deployment of the environment in Nokia mobile terminal software development.

Our experience gained during the establishment, deployment, and application of the environment demonstrates that the methods used for architecture-centric software development and maintenance are heavily influenced by the particular context they are applied to. The architectural profiles, for instance, are domain-dependent, which also makes the model validation rules, model manipulation methods, and reverse architecting methods domain-dependent. Therefore, it is necessary that the tools belonging to the environment are reconfigurable and modifiable so that they could be conveniently adapted to a new domain. Further, to be able to support software development and maintenance tasks in different domains, it should be easy to integrate new model manipulation and validation tools to the environment.

The kernel of ART environment is designed to support those properties mentioned above, for example, the xUMLi platform is based only on the UML metamodel and is UML CASE-tool independent, and the reverse-architecting toolset is highly modifiable for different implementation techniques. Due to the fact that ROSE is one of the main software design CASE-tools at Nokia, the UML model analysis and processing toolset has specific interfaces to ROSE. But they can be tailored to support other CASE-tools with moderate efforts, because most of the model analysis and manipulation operations are implemented solely on top of the xUMLi platform.

Our work has been mainly focused on modeling and reverse-architecting static architectural views. We will shift our research focus on behavior model/views of architecture design. The topics we are going to investigate include: 1) how to define and use behavioral profiles to regulate the system behavior; 2) techniques and tool support of creating a behavior model from the specification, or recover the behavior model from the test traces (i.e., to further develop the model synthesis tool of ART environment); and 3) how to combine the dynamic and static analysis, and keep the consistency between dynamic and static views of the architecture.

Acknowledgements

The authors wish to thank Nokia Technology Platform for supporting ART project, Jan van der Ven (currently at the University of Groningen, Netherlands) for his work on implementing the set operations, Jani Airaksinen for implementing the artDECO tool, and the TUT PRACTISE/UML group for work on xUMLi.

References

[1] J. Bosch, *Design and Use of Software Architectures: Adopting and evolving a product-line approach*, Addison-Wesley, 2000.

[2] Clements, P. and Northrop, L., *Software Product Lines: Practices and Patterns*, Addison-Wesley, 2001.

[3] J. Kuusela, "Architectural Evolution, Nokia Mobile Phones Case". In *Proceedings of the 1st Working IFIP Conference on Software Architecture (WICSA'1)*, February 1999, San Antonio, Texas, USA.

[4] M. Jazayeri, A. Ran, F. van der Linden (eds.), *Software Architecture for Product Families Principles and Practice,* Addison Wesley, 2000.

[5] N. Medvidovic and R. N. Taylor, "A Classification and Comparison Framework for Software Architecture Description Languages", *IEEE Trans. Softw. Eng. 26*, 1 (Jan. 2000), 70-93.

[6] The Object Management Group, homepage. AT http://www.omg.org/uml, 2003.

[7] C. Hofmeister, R. Nord, D. Soni, "Describing Software Architecture with UML", In *Proceedings of the TC2 First Working IFIP Conference on Software Architecture (WICSA)*, San Anotonio, Texas, Feb. 1999.

[8] N. Medvidovic, D.S. Rosenblum, D.F. Redmiles and J.E. Robbins, "Modeling Software Architecture in the Unified Modeling Language", *ACM Transactions on Software Engineering and Methodology*, Vol.11, No.1, January 2002.

[9] C. Riva, J. Xu and A. Maccari, "Architecting and Reverse Architecting in UML", *Workshop on Describing Software Architecture with UML (as part of ICSE 2001),* Toronto, 2001.

[10] P. Selonen, and J. Xu, "Validating UML Models Against Architectural Profiles", In *Proceedings of ESEC 2003*, Helsinki, Finland, September 2003, pp. 58-67.

[11] J. Airaksinen, K. Koskimies, J. Koskinen, J. Peltonen, P. Selonen, M. Siikarla, and T. Systä, "xUMLi: Towards a Tool-independent UML Processing Platform", In: K. Østerbye (Ed.), *Proceedings of the Nordic Workshop on Software Development Tools and Techniques, 10th NWPER Workshop*, IT University of Copenhagen, Copenhagen, Denmark, August, pp. 1-15.

[12] J. Peltonen, "Visual scripting for UML-based tools", In *Proceedings of ICSSEA 2000 (vol. 3),* Paris, France, December 2001.

[13] M. Siikarla, J. Peltonen, and P. Selonen, "Combining OCL and Programming Languages for UML Model Processing", In *Electric Notes in Theoretical Computer Science (ENTCS) dedicated to the UML 2003 workshops*, Elsevier publishing, San Francisco, CA, USA.

[14] J. Koskinen, J. Peltonen, P. Selonen, T. Systä, and K. Koskimies, "Towards tool assisted UML development environments", In *7th Symposium on Programming Language and Software Tools*, Szeged, Hungary, June 2001.

[15] P. Selonen, K. Koskimies, and M. Sakkinen, "Transformations Between UML Diagrams", In *Journal of Database Management, 14(3)*, Idea Group, 2003, pp. 37-55.

[16] C. Riva, "Reverse Architecting: an Industrial Experience Report", *Proceedings of the 7th Working Conference on Reverse Engineering (WCRE2000)*, Brisbane, Australia, 23-25 November 2000.

[17] ITU-T. Recommendations Z.120. ITU – Telecommunication Standardization Sector, Geneva, Switzerland, May 1996.

[18] Riva, C. and Rodriguez, J.V., "Combining Static and Dynamic Views for Architecture Reconstruction", In *Proceedings of the 6th European Conference on Software Maintenance and Reengineering (CSMR'02)*, 2002.

[19] Y. Yang and J. Xu, "Encoding Informal Architectural Descriptions with UML: an Experience Report", *In Proceedings of UML 2003,* San Francisco, CA, USA.