Metamodel-based Tool Integration

Weiqing Zhang[1], Birger Møller-Pedersen[1], Kai T. Hansen[2]
[1] Department of Informatics, University of Oslo, Norway
[2] ABB Corporate Research Center, Norway

Abstract
Metamodels have been proposed as the basis for tool integration, based on the fact that data and models handled by tools may be defined by means of metamodels, and therefore also their integration. An industrial wind turbine project case study on tool integration has been performed to investigate this. The tool integration scenarios cover both integration of requirements and design Artifacts, and the integration of common data definitions as the basis for data exchange between parts of the system.

1 Introduction

Tool integration is an important and emergent issue in embedded system development. Development within one tool typically depends on data that is made within other tools. Tool integration means that the tool-internal model/data elements distributed across the lifecycle are integrated and handled in a uniform way, like linking a requirement element to whole UML/Simulink models or to the elements of UML/Simulink models work in the same way. The most common tool integration approach is that tools work with each other through their proprietary APIs. The iFEST project (1) proposes an alternative "Adaptor" approach which integrates tools based on tool metamodels and tool element representatives. Tools make either *models* (including implementations in terms of programs) of a system, or *data* (e.g. simulation results, impact analysis). Models are made in modelling languages that are defined by *metamodels*; while data are typically structured in ways that may be described by metamodels.

According to Wasserman(2), tool integration issues can be addressed in five areas: platform, presentation, data, control, and process integration. In this paper we mainly focus on the data integration. The main purpose of this paper is to apply the iFEST adaptor approach to an industrial case study, in order to gain experience and improve the approach itself. The industrial case illustrates two different tool integration scenarios: traceability (between requirements and design elements) and exchange of commonly defined data between parts of a system.

2 iFEST Tool Integration Approach

This chapter describes the iFEST adaptor tool integration approach. The approach has been developed as part of the iFEST project, especially with participants from @pportunity, Atego, Ensta, KTH, Tecnalia, and University of Oslo.

Tools produce and maintain models/data that contain model/data *elements*. Elements are owned, managed, stored and presented in tool-specific ways. A principle of the iFEST tool integration is that it should be independent of how the tools internally handle their model/data elements. For this purpose, we use representatives (called *Artifacts*) to represent the real model/data elements. A tool *Adaptor* provides the bridge from an Artifact to the corresponding tool element, and it also allows external clients access to tool data/model elements and tool functionalities in terms of services. Tool adaptors define their own Artifacts for the corresponding model/data elements and use them as parameters or returned results for adaptor services.

iFEST defines Adaptors for *tool types,* not for specific tools. Tools that handle the models/data defined by the same metamodel are grouped into the same tool type. For example, different UML tools like Papyrus, MagicDraw, and Rational Rhapsody all use UML metamodel, therefore they are classified into the UML tool type. The tool type Artifacts represent the real elements of the models that are produced by tools of a tool type. This *tool type* concept makes the tool integration independent of specific tools.

In this approach, each tool has an adaptor, which is based upon a tool type specific metamodel. Metamodels are used to define the tool element Artifacts. The metaclass Artifact is a core element in these metamodels. In this way, objects of these Artifact metaclasses represent tool models/data elements. The Artifact objects have common properties like URIs of real elements to represent the elements. Specific tool type Artifacts such as Trace Artifact may have relations with other Artifacts, like e.g. a 'source' relation to a Requirement Artifact and a 'target' relation to an UML Artifact.

There are two alternative ways of using metamodels to define tool type Artifacts: define new metamodels for tool types, with metaclasses for all the real model/data elements that have to be handled; or rely on existing and independent metamodels, as e.g. the existing official UML metamodel or an independently made Simulink metamodel, and define these single metaclasses as subclasses of Artifact (with properties such as UID and type of model element).

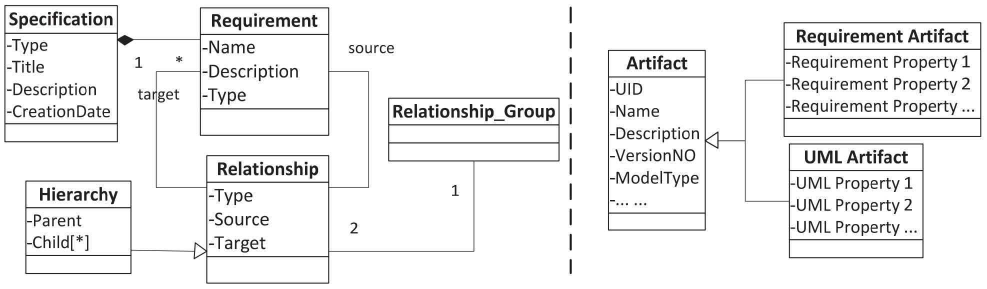

Figure 1 Definition of Tool Type Artifacts by means of a Metamodel

As illustrated in Figure 1, the left side is an independently defined metamodel with the detailed data/model structure for a requirement tool type. In this metamodel, all the metaclasses are subclasses of Artifact. As an alternative, the right side figure uses single metaclasses, e.g. Requirement_Artifact and UML_Artifact, to represent each metaclass of the tool types. The first approach clearly defines the structure of data/model; while the second alternative is more flexible in that it is easy to make changes to the metamodels, such as adding or modifying metaclasses. Metamodel tool integration approaches like Fujaba (3) only handles the tools that are fully controlled by the integration platform, while from the above discussion we find that iFEST also integrates tools that are independent of the iFEST project.

3 Industrial Case Study Description

3.1 Case Study Description

The industrial case addresses two main tool data integration scenarios for a wind turbine system development: traceability between requirements and models, and transformation of a common data definition into models (in two different languages) of two parts of a system that have to exchange data.

As shown in Figure 2, sensors collect environmental data periodically and transmit the data to an IEC61131(4) module and a Simulink module in real time. The IEC61131 module uses these sensor data, and produces outputs for the Simulink module and a remote financial system. Meanwhile, the Simulink module receives the sensor data and the real time data from the IEC61131 module, and then calculates with these data and generates commands for controlling the performance of the wind turbine.

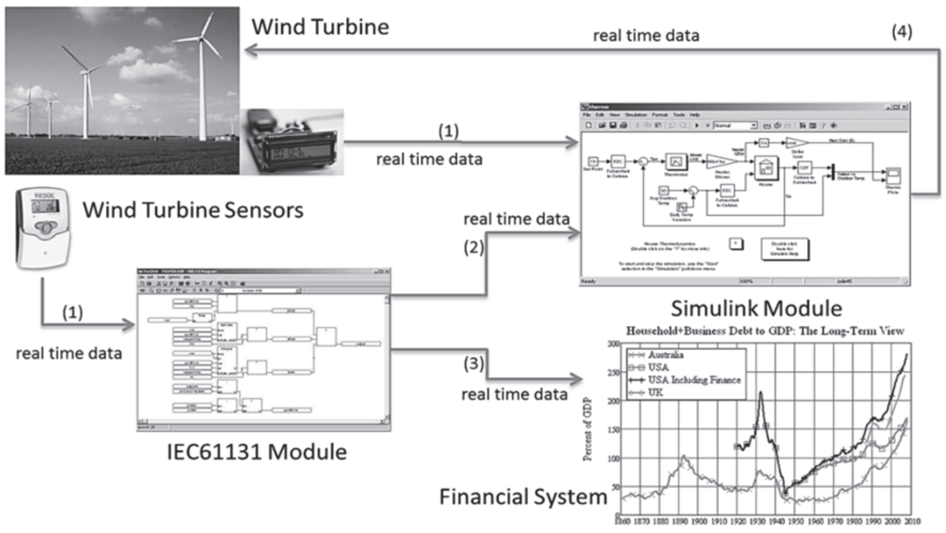

Figure 2 Industrial Case Study Description

During the development process, engineers address requirements in requirement tool first, then design the IEC61131 models and Simulink models, and then build up the traceability accordingly. The Simulink module uses the real time data such as temperature values from the IEC61131 module. These values may be in different format, such as in Fahrenheit or Centigrade degree. Engineers expect that these shared data can be easily understood by both systems.

3.2 Use of Artifacts for Traceability

Traceability is the ability to logically relate two or more Artifacts of the product lifecycle, e.g. matching requirement items with models or part of models, source code implementations, test cases, and verification and validation activities results.

The left part of Figure 3 shows how trace links between requirement elements and UML elements that are handled by means of Artifacts and Adaptors. Each tool specific adaptor is based upon its tool type Artifact definition, such as UML Artifact, Requirement Artifact and Trace Artifact metaclasses. Adaptors generate tool type Artifact instances for the purpose of tool integration. The UML model element that shall be traced can be e.g. a Class, or a Use Case, and that is represented by the UML Artifact instance. A trace link contains a source Artifact and a target Artifact. The UML Artifact instances and the Requirement Artifact instances represent the real elements within the UML tool and the requirement tool, respectively, by means of URIs. The trace tool traces the data from tools of UML/Requirement tool types and is independent of specific tools.

3.3 Use of Models in Exchanging Common Data

This section illustrates how models and model transformation help in defining common data to be exchanged between different parts of a system. As shown in the right part of Figure 3, the case is based on a common input and output data type definition that enables the exchange of data between the Simulink and IEC 61131 parts of the system. A common data type Temperature is defined in an UML tool. This shall then be available in both the Simulink and IEC61131 modules, and the parts of the system made by these two languages should be able to exchange Temperature values in different formats. Given (predefined) metamodels for Simulink and IEC61131, the transformations produce two model elements according to the two metamodels, and the adaptors will then be able to produce the corresponding real model elements.

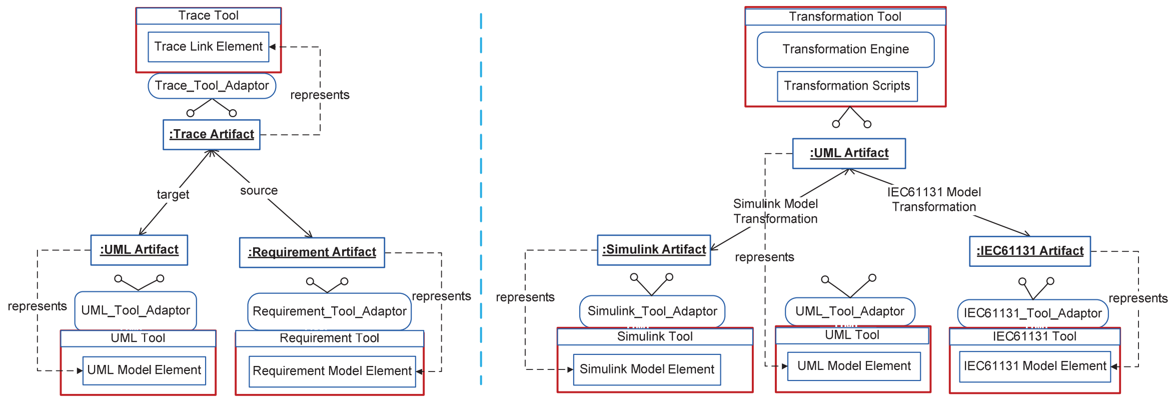

Figure 3 Apply iFEST Approach by means of Artifacts and Adaptors

We have implemented a Java prototype to demonstrate the idea of both how to use Artifact concepts in traceability and exchange common data scenarios. The trace contains links to the source Artifact and target Artifact, instead of the real tool elements. The IEC61131 and Simulink models are the instances according to the IEC61131 and Simulink metamodels, e.g. the graphical Block representations are instances of the Simulink Block metaclass. The Simulink Artifacts and IEC61131 Artifacts are used to represent the graphical models or part of the graphical models from the Simulink or IEC61131 tools. The applications are implemented differently in the two languages that are defined by the two different metamodels. The UML model that defines the common representation of the exchange data type is transformed into the two parts to get 'code' according to their respective language metamodels.

4 Conclusion

A tool integration approach based upon metamodels has been applied to an industrial case study, and it has been demonstrated that it works for as diverse data integration scenarios as traceability between Artifacts, where very little has to be known about the real traced model elements, and transformation from commonly defined data in one language into corresponding data definitions in two other languages, involving the full metamodels of these languages in order to handle the real data definition Artifacts.

5 Acknowledgements

This work was supported by the ARTEMIS Tool Integration project iFEST.

6 Reference

1. iFEST. iFEST - industrial Framework for Embedded Systems Tools. ARTEMIS-2009-1-100203. 2010.
2. Wasserman AI. Tool Integration in Software Engineering Environments. Proceedings of the international workshop on environments on Software engineering environments 1990.
3. Sven Burmester HG, Jörg Niere, Matthias Tichy, Robert Wagner, Lothar Wendehals, Albert Zündorf. Tool integration at the meta-model level: the Fujaba approach. International Journal on Software Tools for Technology Transfer (STTT) - Special section on tool integration applications and frameworks. 2004;6(3).
4. PLCopen. Introduction into IEC 61131-3 Programming Languages. 2011.