An End-to-End Tool Chain for Multi-View Modeling and Analysis of Avionics Mission Computing Software*

Zonghua Gu, Shige Wang, Sharath Kodase and Kang G. Shin

Real-Time Computing Laboratory
Department of Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, MI 48109-2122, USA
{zgu,wangsg,skodase,kgshin}@eecs.umich.edu

Abstract

We present an end-to-end tool-chain for model-based design and analysis of component-based embedded real-time software, with Avionics Mission Computing as an application domain. The tool-chain covers the entire system development life-cycle including modeling, analysis, code generation, and runtime instrumentation. Emphasis is placed on integration of tools developed by multiple institutions via standardized interface format definitions in XML. By capturing all relevant information explicitly in models at the design level, and performing analysis that provides insight into non-functional aspects of the system, we can raise the level of abstraction for the designer, and facilitate rapid system prototyping.

1 Introduction

The Bold Stroke framework [6] is a product-line architecture used at Boeing for developing avionics mission computing software, which is the embedded software aboard a military aircraft for controlling mission-critical functions, such as navigation, target tracking and identification, and weapon firing. It is modeled in UML, manually coded in C++, and runs on top of Real-Time CORBA Event Service. Even though there exist UML models for the software, they mainly serve in a documentation role that the software developer can refer to while performing manual coding. Therefore, the link between model and code is weak and easily broken in the process of system maintenance and evolution, when code is modified or enhanced without the corresponding changes at the model-level, or vice versa. Furthermore, UML has little support for analysis that is relevant for embedded systems, such as real-time properties like schedulability, safety properties like deadlock freedom, etc.

The DARPA MoBIES (Model-Based Integration of Embedded Software) program, started in 2000, has been exploring model-based approaches for embedded software composition and analysis, especially emphasizing non-functional issues such as timing, synchronization, dependability and resource constraints. Within the context of the MoBIES program, researchers from multiple institutions have been working together to produce an end-to-end tool-chain with the Bold Stroke framework as the main application domain. All aspects of an embedded real-time system are captured in domain-specific models, including software components and architecture, timing and resource constraints, processes and threads, execution platforms, etc. Configuration code generation allows automated building of the application executable. Instrumentation of the application running on a target platform is used to collect runtime statistics that are fedback into the models. Analysis tools perform various static analyses based on the models, including system-level dependency analysis, execution-rate assignment to component ports, real-time and schedulability analysis, and automated allocation of components to processors. The MoBIES tool-chain contains a number of novel ideas:

- It uses *meta-modeling* to define domain-specific modeling constructs that allow us to precisely capture domain concepts, as opposed to general purpose UML modeling tools that provides a fixed set of modeling constructs with limited extensibility.
- It covers the entire systems development lifecycle including modeling, analysis, code generation and runtime instrumentation, as opposed to *point so-*

*The work reported in this paper was supported in part by DARPA and ARO under contracts/grants F3615-00-1706 and DAAD19-01-1-0473, respectively.

lutions that targets limited points in the system life-cycle.

- The various tools are integrated on top of the *Open Tool Integration Framework* (OTIF) [2], which enables seamless runtime collaboration of different tools via publish/subscribe communication paradigm on a CORBA backplane and standardized interface file format definitions in XML. It allows easy plug-in of other third-party tools by adding a few lines of code that calls the OTIF API. This represents a significant advantage over closed, proprietary, monolithic tool architectures that offer limited or no extensibility.

This paper is structured as follows. Section 2 describes the ESML modeling language; Section 3 describes several standard interface formats; Section 4 describes the overall workflow of the tool-chain; Section 5 describes the AIRES tool for model-level analysis; Section 6 discusses related work, and the paper concludes with Section 7.

2 Multi-View Modeling

Based on the Model-Integrated Computing (MIC) [8] approach, the Generic Modeling Environment (GME) is a configurable toolset for creating domain-specific modeling and program synthesis environments through a *meta-model* that specifies the modeling paradigm of the application domain. The *meta-model* contains descriptions of the entities, attributes, and relationships that are available in the modeling environment, and defines the family of models that can be created using the resulting modeling environment.

The *Embedded Systems Modeling Language* metamodel (ESML) [4] defines a comprehensive visual modeling language that captures all essential *aspects*, or *views*, of the embedded system, including software architecture, timing and resource constraints, execution threads, execution platform information (processors and network), allocation of components to threads/processors, etc. The model of computation is based on publish/subscribe paradigm on top of CORBA Event Service. Components are composite objects with ports interacting with one another either through event triggers or method invocations. Components are allocated to the (possibly distributed) target execution platform, and execute within the context of *system threads*, which are triggered at harmonically-related execution rates such as 1Hz, 5Hz, and 10Hz, and may span multiple processors. We refer the interested reader to [4] for more details on ESML.

3 Standard Interface Formats

Models described in ESML serve as the central repository of information for all analysis and code generation purposes. Several standardized interface formats such as AIF, CIF and IIF have been defined for use in conjunction with ESML to facilitate integration with third-party tools. They are described with UML-based meta-models, and translators between them can be written using APIs generated from the meta-models.

The *Analysis Interface Format* (AIF) is essentially a subset of the ESML language that contains the dependency and real-time information needed by the analysis tools. Utility tools developed by Vanderbilt University can be used to transform ESML models into AIF files, as well as feedback analysis results from AIF files into ESML model. Besides ESML/GME, another modeling tool developed by Honeywell has been integrated into the MoBIES tool-chain by extracting AIF files from its models.

The *Configuration Interface Format* (CIF) is used to express the component interconnection topology and generate C++ header file used in initializing the application structure at system startup. It is essentially a configuration script that guides generic configuration methods in the creation of components, event ports, facets and receptacles, establishing the relationships between them, and configuring them according to various QoS attributes. CIF can be obtained in a forward direction from the ESML model by a translator developed by Vanderbilt University, or from reverse engineering an existing legacy application by instrumenting the system.

The *Instrumentation Interface Format* (IIF) defines a standardized format for collection of runtime execution traces used for off-line analysis. Instrumentation currently supports gathering and outputting static/configuration information (e.g., threads, components, publish ports, subscribe ports, and events) as well as dynamic information, which includes event timestamps (i.e., supplied event timestamps and consumed event timestamps), component mode timestamps, thread preemption timestamps, frame timestamps, and remote method timestamps. A utility tool developed by Southwest Research Institute is used to extract information from IIF files and update AIF files with aggregate timing information such as worst-case execution time (WCET) for component methods. WindView from WindRiver Systems is used to visualize component execution timeline in the form of Gantt Charts.

4 Workflow of MoBIES Tool-Chain

As shown in Figure 1, the MoBIES workflow has the following steps:

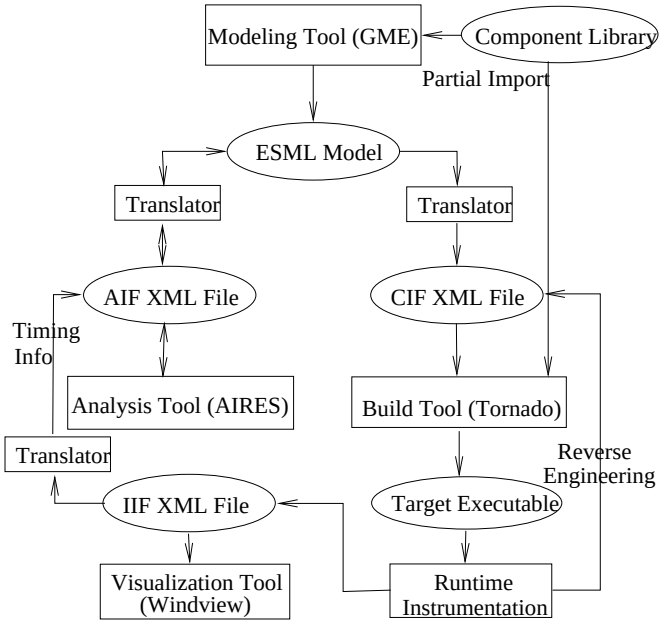

Figure 1: The end-to-end MoBIES tool-chain for avionics mission computing. Tools within the tool-chain inter-operate via standardized interface file formats in XML such as AIF, CIF and IIF. Translators, such as *ESML2AIF*, *ESML2CIF* and *IIF2AIF*, are command-line utility programs that transform between file formats.

1. The *input translation* step imports existing UML models in Rational Rose into GME as component types in ESML. The designer then manually constructs models of system architecture in ESML by instantiating and inter-connecting the components, and enhancing the models with attributes specific to embedded systems such as timing and resource information.

2. The *analysis translation* step extracts information from the ESML models for analysis purposes in the form of Analysis Interface Format (AIF) files. The analysis tool called *AIRES* (Automatic Integration of Reusable Embedded Software), developed at University of Michigan, performs various types of static analysis tasks on the AIF models and updates them with analysis results, which can be imported back into the ESML models, as shown by the bi-directional arrow between ESML and AIF models.

3. The *configuration translation* step generates system configuration file in the form of Configuration Interface Format (CIF) files. Together with the component library, the target application can then be built using the Tornado environment from WindRiver [10].

4. Once we have a running system on the target platform, we can instrument the system and collect runtime execution trace in the form of Instrumentation Interface Format (IIF) files. We can import IIF files into the AIF model in order to add certain timing annotations, or use the WindView tool from WindRiver [10] to visualize the execution timeline. Instrumentation can also be used to reverse-engineer an existing application to create CIF models.

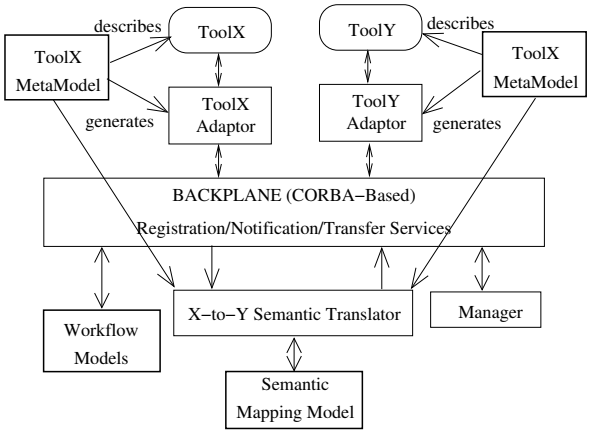

Figure 2: The MoBIES Open Tool Integration Platform (MOTIF).

5 The AIRES Tool

We provide a brief overview of the AIRES tool, and refer the interested reader to [3] for more details. AIRES extracts system-level dependency information from AIF file, including event- and invocation-dependencies, and constructs port- and component-level dependency graphs. Various analysis tasks are supported based on these graphs, such as checking for anomalies like dependency cycles, visual display of dependency graphs, as well as forward/backward slicing to isolate relevant components. It then assigns execution rates to component ports by traversing dependency graphs from timers, and uses real-time scheduling theory to analyze the resulting system of real-time task-set. Allocation of components to execution platforms is assisted by several heuristic algorithms such as *first-fit*, *best-fit* and *graph min-cut*.

We provide some details on using *model-checking* for formal verification, since this functionality was added after the publication of [3]. Model-checking is a popular technique for software verification, which explores

the system state space exhaustively and proves or disproves property specifications, typically written in temporal logic. It can perform deeper semantic analysis, and uncover subtle bugs that are harder to detect with conventional simulation or static analysis techniques. We transform parts of ESML models into a variant of process algebra called *Finite State Processes* [5](FSP), and use an existing tool *Labelled Transition System Analyzer*(LTSA) to analyze the resulting specification. Since FSP is an untimed formalism, we can only check for software *logical* properties such as deadlock freedom, not properties related to system timing behavior.

In order to automate the transformation from ESML models to FSP, we provide a set of reusable component building blocks that can be instantiated to form a complete system. Documents provided by Boeing describe various types of components in natural language. We use FSP to provide an unambiguous, formal description for each component *type* based on the natural language descriptions, and instantiate each component *instance* to form a system architecture. An example of a component type description from Boeing's documentation is:

> The ModalComponent is used to alter the flow of events. The component can be enabled and disabled via the facet method ChangeMode(). When it is enabled, it will update and generate an event when it receives an event. When it is disabled, it will not update or generate an event.

The corresponding FSP specification is shown below, assuming the component is initially disabled:

```
ModalComponent = Disabled,

Disabled = (enable->Enabled | disable->Disabled |
inEvt->Disabled),

Enabled = (enable->Enabled | disable->Disabled |
inEvt->issueCall->receiveReply->outEvt->Enabled1).
```

6 Conclusions

We have described a model-based approach for analysis and rapid prototyping of object-oriented real-time software, with the avionics mission computing software as the main application domain. The MoBIES toolchain covers the entire systems development life-cycle including modeling, analysis, code generation and runtime instrumentation. We believe the MoBIES toolchain represents a significant improvement over the current software development practice, which relies heavily on time-consuming and expensive testing on the target platform, as it provides insight into non-functional aspects of models at design-level, and helps the engineer make high-level design decisions that have a large impact on the embedded software.

Acknowledgements We would like to thank engineers at Boeing as well as researchers at Vanderbilt University and Southwest Research Institute for fruitful collaborations on the MoBIES tool-chain.

References

[1] Dionisio de Niz and Raj Rajkumar. Geodesic - a reusable component framework for embedded real-time systems. Technical report, Carnegie Mellon University, 2002.

[2] S. Neema G. Karsai, A. Lang. Tool integration patterns. In *Proceedings of Workshop on Tool Integration in System Development, European Software Engineering Conference*, pages 33–38, 2003.

[3] Zonghua Gu, Sharath Kodase, Shige Wang, and Kang G. Shin. A model-based approach to system-level dependency and real-time analysis of embedded software. In *Proceedings of Real-Time Applications Symposium*, 2003.

[4] Gabor Karsai, Sandeep Neema, Arpad Bakay, Akos Ledeczi, Feng Shi, and Andy Gokhale. A model-based front-end to tao/ace. In *Proceedings of the 2nd Workshop on TAO*, 2002.

[5] Jeff Magee and Jeff Kramer. *Concurrency: State Models and Java Programs*. Wiley, 1st edition, 2000.

[6] David Sharp. Object-oriented real-time computing for reusable avionics software. In *Proceedings of Fourth International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 185–192, 2001.

[7] John Stankovic. Vest: A toolset for constructing and analyzing component based operating systems for embedded and real-time systems. Technical report, University of Virginia, 2000.

[8] Janos Sztipanovits and Gabor Karsai. Model-integrated computing. *IEEE Computer*, 30(4):110–111, April 1997.

[9] TimeSys website. http://www.timesys.com.

[10] WindRiver website. http://www.wr.com.