Domain-specific Modeling of Power Aware Distributed Real-time Embedded Systems

Gabor Madl, Nikil Dutt
{gabe, dutt}@ics.uci.edu

Center for Embedded Computer Systems
University of California, Irvine, CA 92697, USA[⋆]

Abstract. This paper provides two contributions to the research on applying domain-specific modeling languages to distributed real-time embedded (DRE) systems. First, we present the ALDERIS platform-independent visual language for component-based system development. Second, we demonstrate the use of the ALDERIS language on a helicopter autopilot DRE design. The ALDERIS language is based on the concept of platform-based design, and explicitly captures asynchronous event-driven component interactions as well as the underlying platform for the computation. Unlike most modeling languages, ALDERIS has formally defined semantics providing a way for the formal verification of dense real-time properties and energy consumption.

1 Introduction

Component-based design is an emerging principle for the engineering of complex high-availability distributed real-time embedded (DRE) systems. It has been successfully applied in the domains of hardware design [1], QoS-aware middleware [2], and intellectual property (IP) reuse [3], among others. Components provide an intuitive way to reuse proven designs and implementation, shifting the focus from development by construction to development by *composition*.

Despite recent advances in component-based system design, several key challenges remain that make it hard to develop complex DRE systems with hard QoS-support. Mission-critical system design requires a paradigm shift from conventional methods; the worst case behavior of components have to be considered instead of the average behavior. QoS-support has to be an integral part of the design process providing a way for the rapid evaluation of system designs on a formal basis. To provide a practical modeling language for embedded systems one has to consider how to express *multiple QoS properties* using the same language. Designers have to provide a method that allows to find a balance between various properties such that the system as a whole satisfies all major design constraints.

Domain-specific modeling languages (DSMLs) are languages targeting a well-defined application domain. This approach is rather different from mainstream modeling efforts that focus on creating a language for a wide range of applications, such as *UML*. DSMLs in our approach are defined using *meta-modeling* [4] therefore the designer

[⋆] This research was partially supported by the NSF Grant ACI-0204028 has the option of creating languages that have well defined semantics and are a good fit for a problem domain. Large-scale systems that involve several application domains are modeled as a *composition* of DSMLs. We believe that defining semantics to smaller modeling languages and their composition is more likely to succeed than to define it for a large generic modeling language.

This paper presents the *Analysis Language for Distributed, Embedded and Real-time Systems* (ALDERIS) DSML. ALDERIS is a specification language for power aware distributed real-time embedded systems. The language captures dense (continuous-scale) real-time properties on a distributed platform, and energy savings methods based on frequency- and voltage-scaling. ALDERIS provides a way for the design-time formal verification of system models as well as automated simulation. We also present an equivalent compact XML representation that is the input language of the open-source *Distributed Real-time Embedded Analysis* (DREAM) tool. DREAM implements several analysis and optimization algorithms [5] and also supports formal verification based on the UPPAAL [6] and IF toolsets [7].

The remainder of the paper is organized as follows: Section 2 describes the ALDERIS language its design by meta-modeling, Section 3 describes a case study that demonstrates the use of the ALDERIS modeling language, Section 4 compares the results with related work on the field, and Section 5 presents concluding remarks.

2 The ALDERIS Domain-specific Modeling Language

This section describes the ALDERIS DSML and its role in our model-based analysis framework. The formal semantics of ALDERIS is described in [5]. We formalize an abstract model of computation that can express dense real-time properties and power consumption in a common semantic domain. We propose a platform-based analysis of

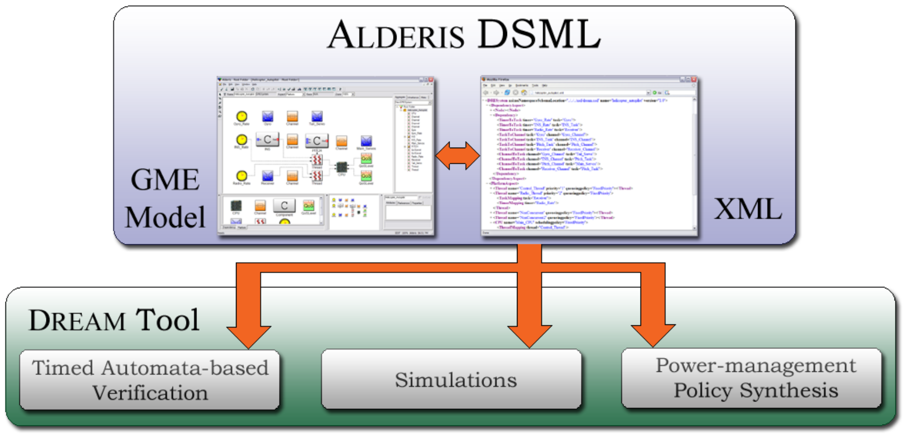

ALDERIS DSML
GME Model
XML
DREAM Tool
Timed Automata-based Verification
Simulations
Power-management Policy Synthesis

Fig. 1. Model-based Analysis Framework based on ALDERIS and DREAM

DRE systems consisting of two major aspects: *dependency*, which describes various relations and dependencies between tasks, and *platform*, which specifies the platform that executes the tasks. We capture both these aspects in ALDERIS by specifying the event flow between tasks and their mappings to platform processors.

The ALDERIS language has both visual and textual concrete syntax. Subsection 2.2 describes how we used the meta-modeling to specify the visual syntax of ALDERIS using the *Generic Modeling Environment* (GME) [8] tool by specifying elements and associations between them. Associations can express various relations such as containment, inheritance etc. The textual syntax of ALDERIS is based on XML *schemas* that provides an easy way to exchange the models between various tools. The XML representation has the same abstract syntax as the visual models.

Figure 1 shows an overview of the model-based analysis framework based on ALDERIS and DREAM. ALDERIS models can directly be analyzed using the DREAM tool. The DREAM tool is based on the timed automata [9] model of computation and implements algorithms for (1) real-time verification using the UPPAAL [6] and Verimag IF [7] tools, (2) simulation-based verification of non-preemptive systems based on a discrete event scheduler [5], and (3) power management policy synthesis using the UPPAAL tool [5]. The timed automata models are automatically generated from ALDERIS models as described in [10]. This paper describes the format of the visual and textual ALDERIS models in a simple helicopter autopilot case study. We illustrate the use of the timed automata-based analysis in Section 3. However, the ALDERIS DSML does not assume the timed automata formalism and allows the use of other models of computation such as data-flow or Petri-nets. For the detailed discussion of analysis methods already implemented in DREAM please see [5, 10].

2.1 Syntax

The ALDERIS model of computation is a tuple $M = \{T, C, TR, TH, PR\}$ where T is a set of *tasks*, C is a set of *event channels*, TR is a set of *timers* which are special tasks that publish events at a given rate, TH is a set of *threads* that represent tasks that are scheduled non-preemptively, and PR is a set of *platform processors*. Tasks and timers are assigned to execute on a specific thread and processor. The thread associated with a given task or timer is specified by the map $\mathsf{Thread} : T \cup TR \to TH$, and the processor associated with a given thread is specified by the map $\mathsf{Processor} : TH \to PR$. Timers generate periodic events as specified by the map $\mathsf{Period} : TR \to \mathbb{N}^+$. Tasks are attributed by the properties *priority*, *sub-priority*, *deadline*, *worst case execution time*, *best case execution time* specified by the mappings $\mathsf{p}(t) : T \to \mathbb{N}^+$, $\mathsf{sp}(t) : T \to \mathbb{N}^+$, $\mathsf{deadline}(t) : T \to \mathbb{N}^+$, $\mathsf{wcet}(t) : T \to \mathbb{N}^+$, $\mathsf{bcet}(t) : T \to \mathbb{N}^+$. We write $\mathsf{State}(t,x)$ to denote the state of t at (global) time x_g: $(\forall t \in T)(\forall x \in \mathbb{N})\ \mathsf{State}(t,x) \in \{\mathtt{init}, \mathtt{wait}, \mathtt{run}, \mathtt{pass}\}$.

2.2 Specifying the ALDERIS DSML using Meta-modeling

The ALDERIS language is expressive enough to capture a wide range of DRE systems [5]. This section demonstrates how the concepts of *model-integrated computing (MIC)* [4] can be utilized to define the ALDERIS DSML. MIC promotes a metamodel-based approach for powerful domain-specific abstractions that capture key concepts and

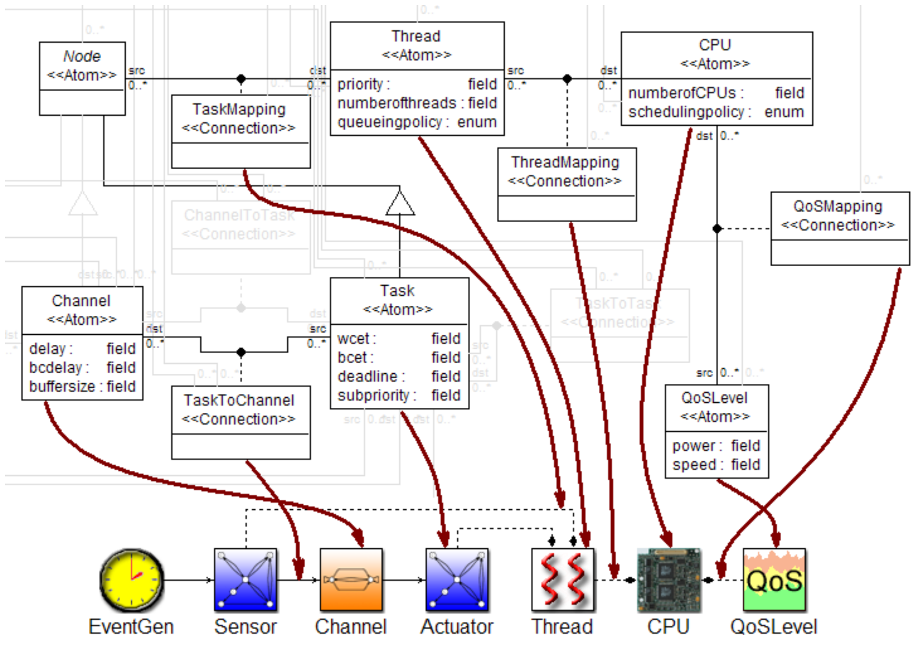


Fig. 2. Specifying the ALDERIS DSML using Meta-modeling

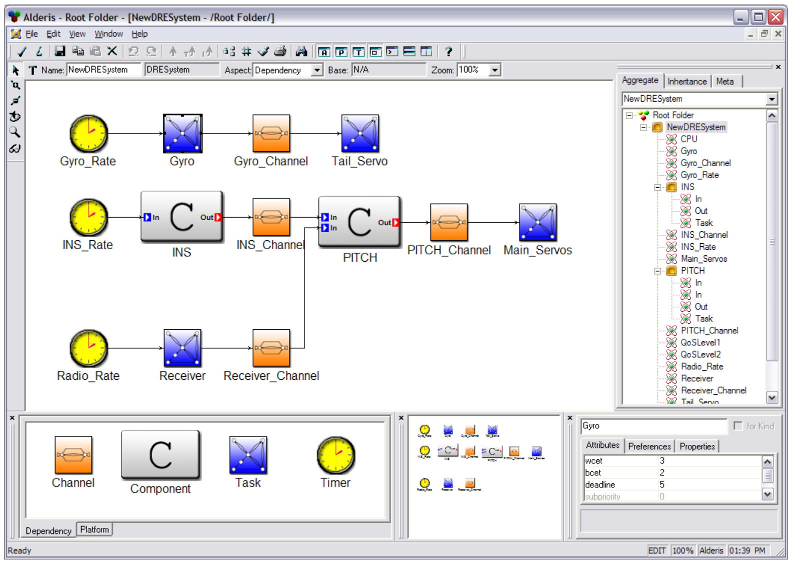
concerns of DRE systems, such as their structure, behavior, and environment, as well as the QoS properties they must satisfy.

GME [8] is an MIC toolsuite that provides a visual interface to simplify the development of domain-specific modeling languages (DSMLs). GME contains a metamodeling environment that supports the definition of paradigms, which are type systems that describe the roles and relationships in particular domains. GME has a flexible object-oriented type system that supports inheritance and instantiation of elements of DSMLs.

Figure 2 illustrates the specification of the ALDERIS language using the GME meta-model, which is a variation of UML class diagrams. The figure shows a part of the ALDERIS meta-model with its corresponding visual representation in GME. The curvy arrows show how individual modeling elements and their relations are defined by different parts of the meta-model. The ALDERIS modeling language is automatically synthesized from the meta-model by the GME tool. The next section describes how the synthesized ALDERIS DSML is used to model power aware DRE systems.

3 Applying ALDERIS to Helicopter Autopilot Design

This section describes a small-scale helicopter autopilot case study to illustrate the use of the ALDERIS modeling language. Please see our technical report [5] for more detailed discussion, performance analysis and large-scale examples, and the underlying analysis methods based on timed automata model checking methods and simulations.


Alderis - Root Folder - [NewDRESystem - /Root Folder/]

Name: NewDRESystem DRESystem Aspect: Dependency Base: N/A Zoom: 100%

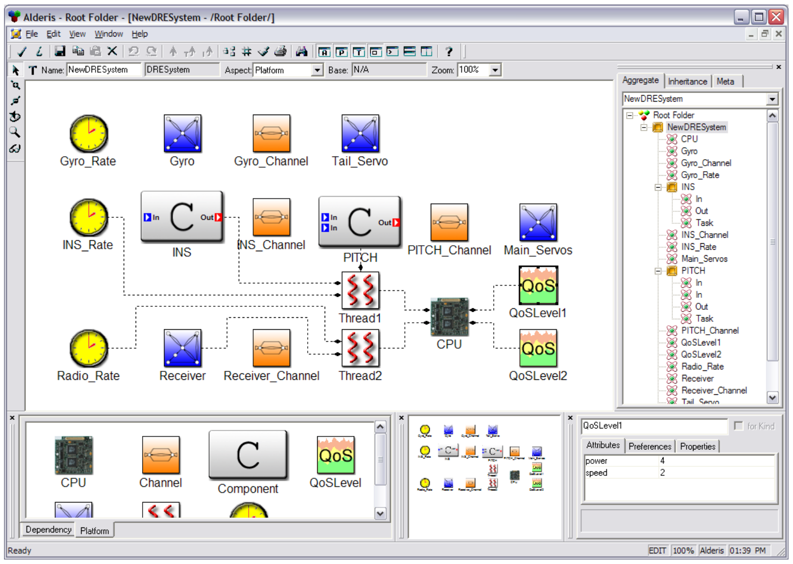
wcet	3
bcet	2
deadline	5
subpriority	0

Fig. 3. The Dependency Aspect of the Autopilot Design

Helicopter controllers are well-known real-time mission-critical systems, since helicopters inherently have unstable flight modes that have to be avoided, otherwise the safety of the helicopter can be at risk. Although energy consumption is not a traditional problem domain for autopilot design, the wider adoption of *unmanned aerial vehicles (UAVs)* will require cheaper and smaller DRE systems where power consumption is an essential design constraint. Traditional engineering practices used in airplane and helicopter design involve extensive testing and validation that is too costly for UAV design. ALDERIS provides a simple modeling language to experiment, evaluate, and formally verify power aware DRE systems.

Figure 3 shows the dependency aspect of the autopilot application. Dependencies are captured using generic (not synchronous) dataflow semantics, following the *publisher/subscriber* communication pattern [11]. The event channels serve two major purposes in the design: (1) they can model delays in the communication between tasks and components. The event channel captures delays as intervals similarly to the tasks' execution intervals. (2) Event channels provide simple FIFO buffering between tasks and components alleviating the need to synchronize communication between the event sources (publishers) and the event sinks (subscribers).

The dependency aspect focuses mainly on the software components and their interactions. The autopilot consists of 3 major components and a few tasks that represent simple sensors and actuators. There are 3 timers in the system (`Gyro_Rate`, `INS_Rate`, and `Radio_Rate`) that drive the computations with different rates. The top part of Figure 3 shows the tail rotor controller. The `Gyro` component reads the gyroscope sensor values and is connected directly to the tail servo that controls the tail rotor speed. This


Alderis - Root Folder - [NewDRESystem - /Root Folder/]

Name: NewDRESystem DRESystem Aspect: Platform Base: N/A Zoom: 100%

```
Root Folder
  NewDRESystem
    CPU
    Gyro
    Gyro_Channel
    Gyro_Rate
    INS
      In
      Out
      Task
    INS_Channel
    INS_Rate
    Main_Servos
    PITCH
      In
      In
      Out
      Task
    PITCH_Channel
    QoSLevel1
    QoSLevel2
    Radio_Rate
    Receiver
    Receiver_Channel
    Tail_Servo
```

QoSLevel1

power	4
speed	2

Ready EDIT 100% Alderis 01:39 PM

Fig. 4. The Platform Aspect of the Autopilot Design

setup provides fast response times resulting in stable tail movement, and is therefore commonly used in helicopters.

The `INS` component represents the internal navigation system of the helicopter, and is based on several sensors such as the inertial measurement unit, compass, and/or GPS devices. The `INS` component implements computationally more expensive functionalities than the `Gyro` component. Instructions for the autopilot may be transmitted over the radio that is handled by the receiver. The received message together with the `INS` data is fed into the `Pitch` component that controls the cyclic and collective pitch of the main rotor. The control signal is sent to the servos/actuators "steering" the aircraft as necessary.

Figure 4 shows the platform aspect of the autopilot case study. There is one main CPU in the system that schedules the `Control_Thread` and `Radio_Thread` preemp-

Task	`WCET`	`BCET`	`DL`	`P`
`Gyro`	3	2	5	-
`Tail_Servo`	1	1	2	-
`INS_Task`	4	2	5	`high`
`Pitch_Task`	2	1	7	`high`
`Main_Servos`	2	2	3	-
`Receiver`	3	3	20	`low`

Channel	`WCDelay`	`BCDelay`
`Gyro_Channel`	2	1
`INS_Channel`	3	2
`Pitch_Channel`	3	1
`Receiver_Channel`	4	2

Table 1. Timing Information for the Autopilot Case Study

```
<?xml version="1.0" encoding="UTF-8"?>
<DRESystem xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="alderis.xsd" name="helicopter_autopilot" version="1.0">
     <DependencyAspect>
       <Node>
         <Task name="Gyro" deadline="2" subpriority="1" wcet="1"/>
         ...<Channel name="Gyro_Channel" buffersize="2"/>
         ...<Timer name="Gyro_Rate" period="5"/>
       <Hierarchy>
         <Component name="INS">
           <TaskContainment task="INS_Task"/>
         </Component>
         ...</Hierarchy>
       <Dependency>
         <TimerToTask timer="Gyro_Rate" task="Gyro"/>
         ...<TaskToChannel task="Gyro" channel="Gyro_Channel"/>
         ...<ChannelToTask channel="Gyro_Channel" task="Tail_Servo"/>
         ...</Dependency>
     </DependencyAspect>
     <PlatformAspect>
       <Thread name="Control_Thread" priority="1" queueingpolicy="FixedPriority">
         <TimerMapping timer="INS_Rate"/>
         <ComponentMapping component="INS"/>
         ...</Thread>
       <CPU name="Main_CPU" schedulingpolicy="FixedPriority">
         <QoSLevel speed="2" power="4"/>
         ...<ThreadMapping thread="Control_Thread"/>
         ...</CPU>
       <CPU name="NonConcurrentManager" schedulingpolicy="NonConcurrent">
         <ThreadMapping thread="NonConcurrent"/></CPU>
     </PlatformAspect>
</DRESystem>
```

Fig. 5. Partial ALDERIS XML Representation for the Helicopter Autopilot

tively based on their priorities. The assignment of tasks and timers to the two threads is shown with dashed lines. The event channels and some of the simple tasks representing sensors and actuators are not mapped to the main CPU as these components are scheduled non-concurrently. The reason for this in the case of sensors/actuators is that they have their own hardware and their execution depends solely on their own states. The event channels model delays and the buffering of the network layer which does not require further scheduling. Priorities for the threads and sub-priorities for the tasks are represented in the model as simple attributes that can be updated using the visual GUI. The possible execution speeds and their corresponding power levels can be specified as QoS-level attributes of the `Main_CPU`. In the helicopter autopilot case study we assume that `Main_CPU` has a full speed and a half-speed mode, and that tasks that execute in the full speed mode consume 4 times as much energy.

Figure 4 shows how we modeled these two QoS-levels by introducing two `QoSLevel` atoms and associating them with the CPU. Any number of QoS-levels can be modeled this way, however ALDERIS cannot capture voltage scaling on a continuous scale. This method provides a way to capture dynamic voltage scaling as well as frequency scaling by specifying the relation between execution speed and its corresponding power level for each QoS-level. Algorithmic methods can utilize this information to obtain a power management policy that respects real-time guarantees. Please see [5] for the thorough discussion of the timed automata-based method for power management policy synthe-

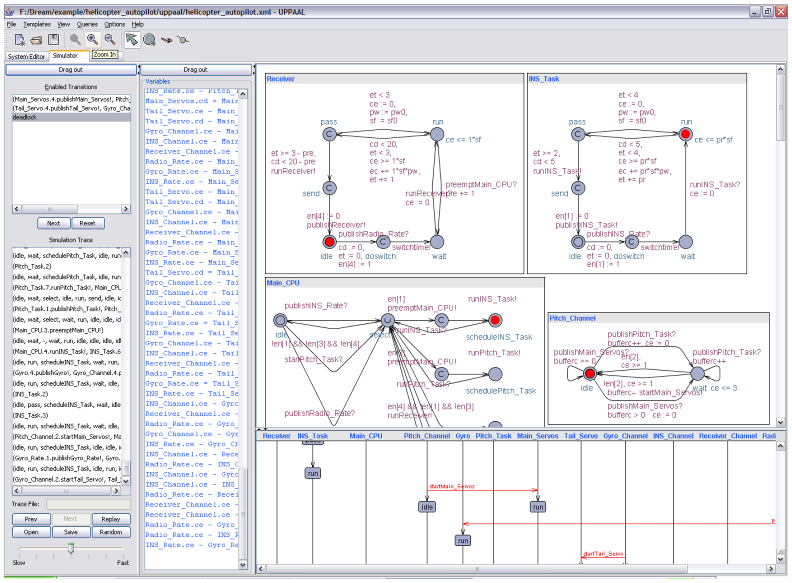


Fig. 6. Unschedulable Execution Trace Detected using Timed Automata Models

sis, and the timed automata models generated from ALDERIS. Figure 5 shows the XML representation of the helicopter autopilot case study shown in Figure 3 and Figure 4. The XML format is specified using *schemas* and provides a simple method to exchange ALDERIS models between tools.

Table 1 shows the parameters assigned to the case study. We have analyzed the helicopter autopilot case study using these parameters using the DREAM tool, and found that it is unschedulable because the `INS` component misses its deadline. At first this seems strange as the `INS` component is deployed on the high-priority `Control_Thread`, and depends only on the `INS_Rate` timer, therefore it should not wait for low-priority tasks.

A great advantage of the model checking method is that whenever a property is violated a counter-example can be automatically obtained. Figure 6 illustrates the execution trace of the counter-example generated by the UPPAAL model checker. The reason behind the missed deadline is that the `Pitch` component may already been executing when the `INS` component becomes enabled. Since non-preemptive scheduling is used between tasks deployed on the same thread the `INS` component has to wait for the `Pitch` component to finish its execution. To compensate for this hidden dependency between the `INS` and `Pitch` component we have increased the deadline of the `INS` component to 7 units which turns the system schedulable. Moreover, our analysis shows that if we specify 10 units deadline for the `INS` and `Pitch` components the processor can save 24% energy by switching to the half speed mode during the execution of these tasks.

The performance of the timed automata-based verification scales exponentially with respect to the number of tasks. Verifying the example shown in Figure 3 and 4 takes around 2 seconds on a 1.6GHz Pentium 4-M processor with 768 MB memory running the Windows XP OS. Please see [5, 10] for the detailed discussion on the verification method.

4 Related Work

The SAE AADL is an international standard avionics architecture description language. AADL is a successor of the *Honeywell MetaH* toolset [12], a commercially available domain-specific architecture description language (ADL) for developing reliable, real-time multiprocessor avionics system architectures. AADL, however, does not consider energy savings as an objective. In contrast, ALDERIS targets power aware DRE systems.

Ptolemy II [13] is a complex modeling framework that composes heterogeneous models of computation to simulate and evaluate embedded systems. Although the MoCs and their composition is formally defined the focus in Ptolemy II is simulation, not verification. In contrast, ALDERIS and the DREAM tool provide a way for formal verification of dense timed systems using several model checkers.

The SYSWEAVER [14] toolset is a component-based framework that supports the reusability of components across systems with different requirements. It supports code generation, as well as automated analysis based on Matlab/Simulink and real-time rate-monotonic analysis tools, such as the TIMEWIZ model-checker. In contrast, ALDERIS focuses on dense time formal verification using the asynchronous event-driven paradigm.

The CADENA [15] framework is an integrated environment for building and analyzing CORBA Component Model (CCM) based systems. Its main functionality includes CCM code generation in Java, dependency analysis, and model-checking. The emphasis of verification in Cadena is on software logical properties. In contrast, ALDERIS represents time and power levels explicitly and allows dense time verification.

The Component Synthesis using Model Integrated Computing (COSMIC) [16] toolkit is an integrated collection of DSMLs that support the development, configuration, deployment, and evaluation of DRE systems based on CIAO, which is an implementation of the CORBA Component Model that is integrated with Real-time CORBA. The major focus of COSMIC is software development, and does not support formal verification.

The Virginia Embedded Systems Toolkit (VEST) [17] is a framework designed for the reliable and configurable composition and analysis of component-based embedded systems from COTS libraries. VEST applies key checks and analysis but - unlike ALDERIS and the DREAM tool - does not support formal proof of correctness.

5 Concluding Remarks

This paper presents the ALDERIS domain-specific modeling language for component-based power aware distributed real-time embedded systems with both visual and XML textual syntaxes. ALDERIS explicitly captures component interactions as well as the platform for computations providing an abstract framework for formal verification and analysis. The ALDERIS meta-model is available for download at http://alderis.ics.uci.edu. Models developed using ALDERIS can be verified and analyzed using the open-source DREAM tool available for download at http://dre.sourceforge.net.

References

1. Alberto Sangiovanni-Vincentelli: Defining Platform-based Design. EEDesign of EETimes (2002)
2. Schmidt, D.C.: Model-driven engineering. IEEE Computer **39**(2) (2006)
3. Daniel D. Gajski and Allen C.-H. Wu and Viraphol Chaiyakul and Shojiro Mori and Tom Nukiyama and Pierre Bricaud: Essential Issues for IP Reuse. In: Asia and South Pacific Design Automation Conference (ASP-DAC 2000). (2000) 37 – 46
4. Sztipanovits, J., Karsai, G.: Model-Integrated Computing. IEEE Computer (1997) 110–112
5. Madl, G., Dutt, N.: Tutorial for the Open-source DREAM Tool. In: CECS Technical Report. (2006)
6. Pettersson, P., Larsen., K.G.: UPPAAL2k. Bulletin of the European Association for Theoretical Computer Science **70** (2000) 40–44
7. Bozga, M., Graf, S., Ober, I., Ober, I., Sifakis, J.: The IF Toolset. Formal Methods for the Design of Real-Time Systems, LNCS 3185 (2004) 237–267
8. Ledeczi, A., Bakay, A., Maroti, M., Volgyesi, P., Nordstrom, G., Sprinkle, J.: Composing Domain-Specific Design Environments. Computer (2001) 44–51
9. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science **126**(2) (1994) 183–235
10. Madl, G., Abdelwahed, S., Schmidt, D.C.: Verifying Distributed Real-time Properties of Embedded Systems via Graph Transformations and Model Checking (accepted). The International Journal of Time-Critical Computing (2006)
11. Schmidt, D.C., Stal, M., Rohnert, H., Buschmann, F.: Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects, Volume 2. Wiley & Sons, New York (2000)
12. Vestal, S.: Formal Verification of the MetaH Executive Using Linear Hybrid Automata. In: RTAS '00: Proceedings of the Sixth IEEE Real Time Technology and Applications Symposium (RTAS 2000), Washington, DC, USA, IEEE Computer Society (2000) 134
13. Lee, E.A., Hylands, C., Janneck, J., II, J.D., Liu, J., Liu, X., Neuendorffer, S., Stewart, S.S.M., Vissers, K., Whitaker, P.: Overview of the ptolemy project. Technical Report UCB/ERL M01/11, EECS Department, University of California, Berkeley (2001)
14. de Niz, D., Bhatia, G., Rajkumar, R.: Model-Based Development of Embedded Systems: The SysWeaver Approach. In: Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06). (2006) 231–242
15. Hatcliff, J., Deng, X., Dwyer, M.B., Jung, G., Ranganath, V.P.: Cadena: An Integrated Development, Analysis, and Verification Environment for Component-based Systems. In: Proceedings of International Conference on Software Engineering. (2003)
16. Gokhale, A., Balasubramanian, K., Balasubramanian, J., Krishna, A.S., Edwards, G.T., Deng, G., Turkay, E., Parsons, J., Schmidt, D.C.: Model Driven Middleware: A New Paradigm for Deploying and Provisioning Distributed Real-time and Embedded Applications. The Journal of Science of Computer Programming: Special Issue on Model Driven Architecture (2005 (to appear))
17. Stankovic, J., Zhu, R., Poornalingham, R., Lu, C., Yu, Z., Humphrey, M., Ellis, B.: VEST: An Aspect-based Composition Tool for Real-time Systems. In: Proceedings of the IEEE Real-time Applications Symposium. (2003)