A Model-driven Development Environment for Composing and Validating Distributed Real-time and Embedded Systems: A Case Study *

Gabriele A. Trombetti, Aniruddha Gokhale, and Douglas C. Schmidt
{gtrombetti,gokhale,schmidt}@dre.vanderbilt.edu
Department of Electrical Engineering and Computer Science
Vanderbilt University
Nashville, TN 37235, USA

Abstract

Model-driven development (MDD) processes are increasingly being used to develop component middleware and applications for distributed real-time and embedded (DRE) systems in various domains. DRE applications are often mission-critical and have stringent quality of service (QoS) requirements, such as timeliness, predictability and scalability. MDD software techniques are well suited for validating the operation of DRE applications since they offer a higher-level of abstraction than conventional third-generation programming languages. The state-of-the-art in model-driven DRE application development is still maturing, however. For example, conventional MDD development environments for DRE application do not yet provide seamless integration of development capabilities and model checking capabilities.

This paper presents three contributions towards an integrated MDD development and model checking environment for DRE applications. First, we describe how our CoSMIC MDD middleware development toolsuite has been combined with the Cadena model checking toolsuite to provide an integrated environment that accelerates the development and validation of DRE applications. Second, we discuss the technical difficulties encountered integrating these tools in the context of a DRE application case study. Third, we discuss R&D issues associated with implementing MDD algorithms for maintaining lossless and semantics-preserving data transfer across the tools. Our results show that interoperation between significantly different tools for MDD is achievable with the proper choice of communication format, semantics, and the development of a reliable graph diff-merge algorithm.

Keywords: Distributed Real-time and Embedded Systems, Component Middleware, Model-driven Systems, Model checking.

*Submitted to Model-driven Software Development - Volume II of Research and Practice in Software Engineering, edited by Sami Beydeda and Volker Gruhn

1 Introduction

Emerging trends and challenges. Large-scale, distributed real-time and embedded (DRE) software systems form the basis of mission- and safety-critical applications essential to national infrastructure, including air traffic and transportation control, emergency response systems, and electrical power grids. These DRE systems include many interdependent levels, such as network/bus interconnects, many coordinated local and remote endsystems, and multiple layers of software. As a result, developers of DRE systems must address the following challenges:

- As *distributed systems*, DRE systems require capabilities to manage connections and message exchange between (possibly heterogeneous) networked computing devices.
- As *real-time systems*, DRE systems require control over end-to-end quality of service (QoS) properties (such as predictability, low-latency, and reliability) and system resources (such as memory, CPU, and network bandwidth).
- As *embedded systems*, DRE systems have weight, cost, and power constraints that limit their computing and memory resources.

DRE systems have historically been developed and validated using relatively static development and analysis techniques (such as function-oriented design and rate monotonic analysis) to implement, allocate, schedule, and manage their resources and QoS. These static approaches have proven to be acceptable for *closed* DRE systems, such as avionics mission computing and automotive anti-lock braking systems. In closed systems the set of application tasks that will run in the system and the loads they will place on system resources change infrequently and are known in advance.

Static approaches are not well-suited, however, for the next-generation of *open* DRE systems, such as total shipboard computing, multimedia teleconferencing on the Internet, and sensor networks supporting emergency management systems.

These open systems evolve more rapidly and must collaborate with multiple remote sensors, provide on-demand browsing and actuation capabilities for human operators, and respond flexibly to unanticipated situational factors that arise at run-time. Desirable QoS properties of open DRE systems include predictability, controllability, and adaptability of operating characteristics for applications with respect to such features as time, quantity of information, accuracy, confidence, and synchronization.

In large-scale open DRE systems, assuring QoS end-to-end is much harder than in smaller-scale open systems due to the dynamic interplay of the many interconnected parts, which are often constructed from smaller parts. It is possible in theory to develop complex open DRE systems from scratch. In practice, however, contemporary economic and organizational constraints – along with increasingly complex requirements and competitive pressures – motivate the reusability of existing tools and platforms.

A key enabler in recent software successes with small- to medium-scale DRE systems (such as avionics mission computing systems) has been *middleware* [1], which is software that provides platform-independent execution semantics and reusable services that coordinate how application components are composed and interoperate. To address the many competing design forces and run-time QoS demands of large-scale DRE systems (such as air traffic and transportation control), however, sustained R&D efforts on comprehensive software methodologies, design-/run-time environments, and hardware/software co-design are required to dependably compose large, complex, interoperable DRE systems from QoS-enabled reusable components. Moreover, the components themselves must be sensitive to the environments in which they are packaged.

Ultimately, what is required is to assemble components that are built independently by different groups at different times to create complete DRE systems that are customized for their requirements and environmental conditions. Over time, these systems become subsystems embedded in still larger open *systems of systems*. Given the complexity of this undertaking, various tools and techniques are needed to configure and reconfigure these systems hierarchically so they can adapt to a wider variety of situations than has historically been possible with earlier generations of smaller-scale, closed DRE systems.

Solution approach → Model-driven development of DRE software. *Model-driven development* (MDD) software processes and tools, such as the Object Management Group's Model Driven Architecture (MDA) [2] or Model Integrated Computing (MIC) [3], are a promising technology infrastructure for addressing the challenges of developing and validating the large-scale open DRE systems described above. MDD is a development paradigm that systematically applies domain-specific modeling languages to engineer computing systems, ranging from small-scale real-time and embedded systems to large-scale distributed enterprise applications. It is *model-driven* because it uses models to direct the course of understanding, design, construction, deployment, operation, maintenance, and modification.

MDD is a key step forward in the long road of converting the art of programming into an engineering process that will ultimately industrialize the production of software [4]. In particular, MDD technologies operationalize the principles of "correct by construction," which involve the use of higher-level specifications early in the design process to express constraints that are successively transformed into running lower-level code that preserves and enforces the semantics of specifications downstream. MDD's "correct by construction" techniques are in contrast to the "construct by correction" techniques commonly used by post-construction tools, such as compilers, source-level debuggers, and script validators.

Due to the sheer magnitude and complexity of the problem space, no single model-driven toolsuite yet offers solutions to all the challenges of large-scale DRE system development. For example:

- Our R&D on model-driven configuration and deployment of component-based DRE systems has resulted in the CoSMIC toolsuite [5, 6], which is an open-source MDD toolsuite with an integrated collection of modeling, analysis, and synthesis tools that address key lifecycle challenges of DRE middleware and applications. The CoSMIC toolsuite supports modeling of DRE system deployment and configuration capabilities, their QoS requirements, and QoS adaptation policies used for DRE application QoS management. The initial set of modeling and synthesis tools in CoSMIC are targeted at the CIAO [7] QoS-enabled component middleware. CoSMIC, however, does not provide tools for analyzing and validating the functional correctness and QoS properties of DRE systems.
- Conversely, various MDD tools exist that perform model checking component-based systems (such as Cadena [8]) or real-time schedulability analysis (such as AIRES [9] and VEST [10]). Model checking is useful for detecting errors early in the development stage, instead of sporadically and at runtime. In particular, if a component has externally identifiable modes (or states), these analysis should be applicable for each mode or set of modes of component functionality. Existing model checking tools, however, do not provide the mechanisms for end-to-end DRE systems composition, assembly, configuration, and deployment.

What is therefore required is an *integrated* MDD tool chain that developers of DRE systems can use to compose, configure, and deploy their applications end-to-end and be able

to validate these configurations and deployments via model checking.

This paper presents three contributions towards providing an integrated model-driven development and model checking environment for DRE applications. First, we describe how CoSMIC has been combined with the Cadena model checking toolsuite using the Open Tool Integration Framework (OTIF) [11] to provide an integrated environment that accelerates the development of DRE applications by addressing key production stages, such as powerful model checking capabilities for tracking errors early in the development stage, reducing total development cost and time-to-market, and increasing the reliability of DRE applications.[1] Second, we discuss the technical difficulties encountered integrating these tools in the context of a case study of a DRE robot assembly application, highlighting how the choice of an effective communication protocol, data interchange format, and development environment for the semantic translators can enable smoother tool integration. Third, we discuss R&D issues associated with implementing algorithms, including coping with export import cycles, storing and transferring supersets and subsets of captured information, merging and preserving information, and addressing future extensibility of the integration.

Paper organization. The remainder of this paper is organized as follows: Section 2 gives an overview of the CoSMIC and Cadena MDD environments; Section 3 presents a case study of a robot assembly DRE application that we used to guide our tool integration strategies; Section 4 describes key R&D challenges associated with integrating CoSMIC and Cadena and explains our solution approaches; Section 5 compares our work with related research; and Section 6 presents concluding remarks.

2 An Overview of the CoSMIC and Cadena MDD Environments

MDD technologies have been used in a variety of contexts. For example, the OMG Model Driven Architecture (MDA) technologies initially focused on enterprise applications [12]. Other MDD techniques, such as Model-Integrated Computing (MIC) [3], focused on smaller scale, tightly coupled embedded systems. More recently, however, MDA and MIC technologies are aligning [13] to add the QoS capabilities necessary to support DRE systems in domains ranging from aerospace [14] to telecommunications [15] and industrial process control [16].

This section provides an overview of our *Component Synthesis using Model Integrated Computing* (CoSMIC) toolsuite that combines the MDD paradigm with QoS-enabled component middleware [5]. We also provide an overview of the Cadena model checking tool developed at Kansas State University [8].

2.1 The CoSMIC Deployment and Configuration Modeling Environment

As shown in Figure 1, CoSMIC consists of an integrated collection of modeling, analysis, and synthesis tools that address key lifecycle challenges of DRE middleware and applications. The CoSMIC toolsuite supports modeling of DRE system deployment and configuration capabilities, their QoS requirements, and QoS adaptation policies used for DRE application QoS management. The initial set of modeling and synthesis tools in CoSMIC are targeted at CIAO [7], which is QoS-enabled component middleware that provides real-time enhancements to the CORBA Component Model (CCM) [17]. CIAO abstracts component QoS requirements into metadata that can be specified in a component assembly after a component has been implemented [18]. Decoupling the specification of QoS requirements from component implementations greatly simplifies the conversion and validation of an application model with multiple QoS requirements into CCM deployment of DRE applications.

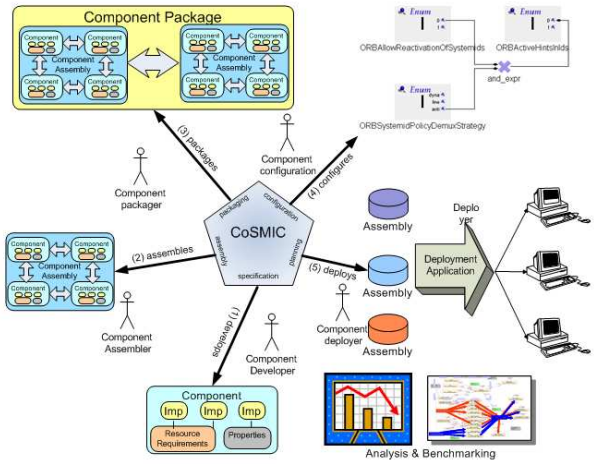

Figure 1: **CoSMIC Model Driven Middleware Development Toolsuite**

CoSMIC provides principled methods needed to specify, develop, compose, integrate, and validate the application and middleware software used by DRE systems. These methods must both enforce the physical constraints of DRE systems (such as footprint and resource constraints) and satisfy the system's stringent functional and systemic QoS requirements. Achieving these goals requires an integrated MDD

[1] The CoSMIC toolsuite and the associated Cadena and OTIF integration translators are available at `www.dre.vanderbilt.edu/cosmic`.

toolchain that allows developers to specify application and middleware requirements at higher levels of abstraction than that provided by low-level mechanisms, such as conventional third-generation programming languages, operating systems, and middleware platforms.

Figure 2 illustrates how CoSMIC tools can be applied in the context of DRE middleware and applications to:

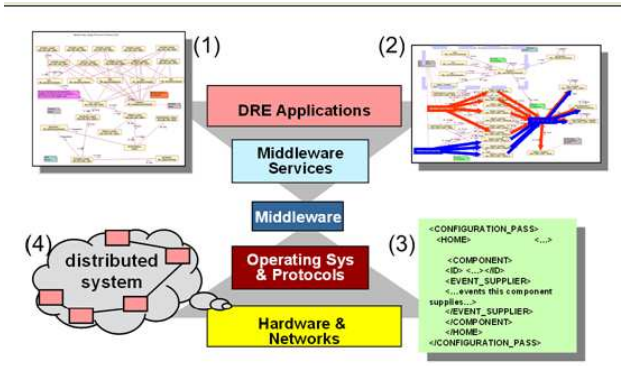

Figure 2: **MDD Process using CoSMIC**

• **Model** different functional and systemic properties of DRE systems via separate middleware- and platform-independent models [3]. Domain-specific aspect model weavers [19] can integrate these different modeling aspects into composite models that can be further refined by incorporating middleware and platform-specific properties.

• **Analyze** different—but interdependent—characteristics and requirements of DRE system behavior (such as scalability, predictability, safety, schedulability, and security) specified via models. Model *interpreters* [20] translate the information specified by models into the input format expected by model checking [8] and analysis tools [21]. These tools can check whether the requested behavior and properties are feasible given the specified application and resource constraints. Tool-specific model analyzers [22, 23] can also analyze the models and predict [24] expected end-to-end QoS of the constrained models.

• **Synthesize** platform-specific code and metadata that is customized for a particular QoS-enabled component middleware and DRE application properties, such as end-to-end timing deadlines, recovery strategies to handle various run-time failures in real-time, and authentication and authorization strategies modeled at a higher level of abstraction [25, 26].

• **Provision** middleware and applications by assembling and deploying the selected components end-to-end using the configuration metadata synthesized by MDD tools. In the case of legacy components developed without consideration of QoS, the provisioning process may involve invasive changes to existing components to provide the hooks that will adapt to the metadata. The changes can be implemented in a relatively unobtrusive manner using program transformation systems, such as DMS [27].

• **Assure** run-time QoS properties are delivered to applications in DRE systems, *e.g.*, via modeling dynamic adaptation and resource management strategies that use hybrid control-theoretic [28] techniques.

Our initial focus in the CoSMIC project has been the deployment and configuration of DRE systems. In particular, CoSMIC is tailored to comply with the OMG's Deployment and Configuration (DnC) specification [29] and provides the following capabilities:

- *Specification and implementation*, which enables application functionality specification, partitioning, and implementation as components.
- *Packaging*, which allows bundling a suite of software binary modules and metadata representing application components.
- *Installation*, which involves populating a repository with the packages required by the application.
- *Configuration*, which allows configuration of the packages with the appropriate parameters to satisfy the functional and systemic requirements of application without constraining to any physical resources.
- *Planning*, which makes appropriate deployment decisions including identifying the entities, such as CPUs, of the target environment where the packages will be deployed.
- *Preparation*, which moves the binaries to the identified entities of the target environment.
- *Launching*, which triggers the installed binaries and bringing the application to a ready state.
- *Adaptation*, which enables run-time reconfiguration and resource management to maintain end-to-end QoS.

The CoSMIC toolsuite also provides the capability to interwork with model checking tools, such as Cadena [8] (described in Section 2.2), and aspect model weavers, such as C-SAW [30]. The integration of CoSMIC with Cadena is the focus of Section 4.

The CoSMIC toolsuite provides a number of modeling languages that are developed using the Generic Modeling Environment (GME) [20]. GME is a metamodeling environment that defines the modeling paradigms[2] for each stage of the CoSMIC tool chain. The CoSMIC tools leverage GME to produce domain-specific modeling languages and generative tools

[2] A *modeling paradigm* defines the syntax and semantics of a modeling language [3].

for DRE applications. CoSMIC ensures that the rules of construction – and the models constructed according to these rules – can evolve together over time. Each CoSMIC tool synthesizes metadata in XML for use in the underlying middleware.

The *Platform Independent Component Modeling Language* (PICML) is the core modeling paradigm provided by CoSMIC. PICML allows modeling the packaging of components into assemblies that can then be configured and deployed appropriately. Deployment and configuration are concerns that crosscut entire assemblies and entire systems. These crosscutting concerns are captured by the different artifacts of PICML. During the configuration and deployment process, multiple concerns captured in the format of metadata in the component development process are woven together by PICML, as shown in Figure 3.

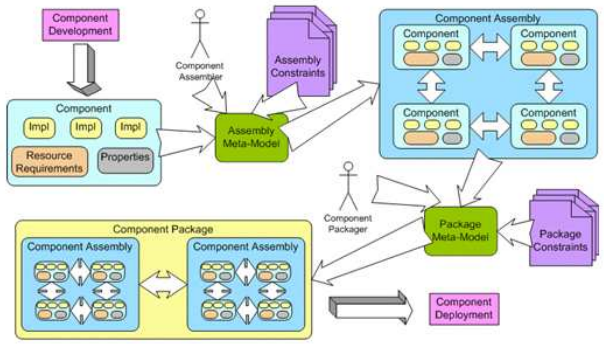

Figure 3: **Platform Independent Component Modeling Language Architecture**

PICML allows the specification of all the above concerns of component-based deployment and configuration by allowing users to model them as elements in a GME paradigm. Additional constraints are defined via the OCL-based constraint definition facilities of GME to ensure that the models built using PICML are semantically valid. PICML's constraints check that the "static semantics" (*i.e.*, the semantics that are required to be present at design time) are not violated. For example, at design-time, PICML can enforce the CCM constraint that only ports with the same interface or event type can be connected together.

After the static semantics are validated, PICML weaves together the separate crosscutting aspects via a "model interpreter," which is responsible for ensuring the "dynamic semantics" of models built using PICML. Dynamic semantics of a modeling language can range from performing analysis of models to synthesizing run-time code for the component. PICML contains multiple model interpreters, each performing a particular function.

The model interpreters in the initial prototype of PICML target the deployment and configuration of DRE components for CIAO [7]. We chose CIAO as our initial focus since it is designed to meet the QoS requirements of DRE systems. As other component middleware platforms (such as J2EE and .Net) mature and become suitable for DRE systems, we will enhance CoSMIC so it supports platform-independent models (PIMs) and then include the necessary patterns and policies to map the PIMs to platform-specific models (PSMs) for various component middleware platforms.

2.2 Cadena Model Checking Environment

The modeling tools in the CoSMIC toolsuite described in Section 2.1 perform various forms of static type-checking based on GME metamodels [31] and constraint checking based on the OMG's Object Constraint Language (OCL) [32]. CoSMIC does not, however, contain sophisticated model checking capabilities, nor static model checking capabilities, such as forward splice, backward splice, chopping and cycle detection. These types of analyses, respectively, detect the components that are affected (forward) or affect (backward) a particular signal or port, highlight all the ports and components in the path of a signal, and detect signal feedbacks that can bring instability to a DRE system. To augment CoSMIC with these model checking capabilities, we integrated it with Cadena [8] shown in Figure 4, which is an open-source MDD environment for modeling and model checking CCM-based DRE systems. Cadena provides analysis capabilities that enable developers to navigate dependencies among components and detect signal loops among components that can cause instability in DRE systems.

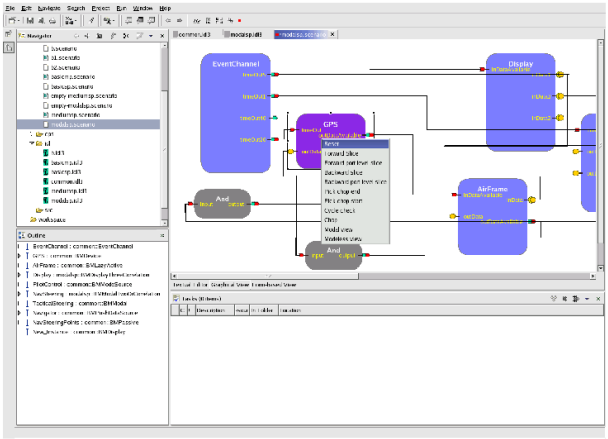

Figure 4: **Cadena MDD Toolsuite for Model Checking and Analysis**

Cadena's user interface is built using Eclipse [33] and it also leverages Bogor [34], which is a highly customizable and modular model checking framework designed to ease the

development of robust and efficient domain-specific model checkers for verification of dynamic and concurrent software. Cadena and Bogor can be used together to find the global inconsistencies in an application starting from the specification of the behavior of the various components. In particular, Bogor employs advanced reduction algorithms, such as collapse compression, heap symmetry, thread symmetry, and novel partial-order reductions that greatly decrease model checking time. For example, checking several hundred components at once is feasible with Bogor and hence with Cadena.

Cadena decouples various aspects of modeling by requiring that these crosscutting concerns be captured in number of files located in a common project space. The following types of files are used by Cadena:

- **IDL3 file**, which are OMG's standard interface description language metadata describing components and their interfaces.
- **Scenario file**, which describes an assembly of interconnected components, including the value of their configuration properties. Cadena provides a graphical visualizer, a text editor, and a form view editor for manipulating for the `.scenario` file. The equivalent of the scenario file in CoSMIC's PICML is the *assembly* view, which enables graphical editing of properties using GME.
- **Profile file**, which acts as a scenario format definition and validation system by defining the type of the properties that can or must be associated with the different components, the connections, or simply at the global level. Cadena supports three types for properties: STRING, INT, and BOOLEAN. There is no equivalent for the `.profile` file on PICML, which is another motivation for integrating CoSMIC and Cadena.
- **Cps file**, which is the most important file required for Cadena's advanced static and dynamic model checking capabilities. The `.cps` file defines the modes of functioning of a component and the internal interconnections that exist within such components, depending on the mode. Information that can be captured in this file includes conditional behavior, such as a set of inputs of a component having an effect on a set of outputs only when that component is in a particular state. The `.cps` file also defines the modes of functioning of the components and the mapping of input to output signal flows. Cadena's model checker can use this information to detect and avoid distributed feedback leading to distributed system crashes and distributed deadlocks during development stage. There is no equivalent for the `.cps` file in PICML nor in any CCM specification.
- **Cor file**, which is used for the channel correlation information associated with specific connections in the scenario (an identifier links them). In PICML the correlators are not captured yet, although their support is planned for the future, so there currently is no equivalent for these files.

The first step when working with Cadena is loading and compiling the .IDL3 into the CORBA Interface Repository, which is a standard database for storing component interface information. The graphical view and the form view within Cadena can be enabled only after the .IDL3 is loaded in the Interface Repository since such features query the IR for fetching the information about the components. The graphical view of the scenario also sports useful static analysis features, such as cycle detection, forward/backward splicing, and signal chopping described earlier.

3 Demonstrating Tool Integration Capabilities via the Robot Assembly Case Study

This section presents a case study that illustrates the benefits of applying the MDD techniques described in Section 2 to a robot assembly application we have developed that is representative of DRE systems in the process control domain. The model represents an assembly line with robots creating various types of goods, such as watches. The complete source code and MDD tools for the robot assembly example are available in the CIAO release from `http://www.dre.vanderbilt.edu/CIAO`. We present a subset of the overall application below to focus the discussion on the use and integration of CoSMIC and Cadena.

3.1 Structure and Functionality of the Robot Assembly Application

Figure 5 illustrates the five core components in the robot assembly application: `ManagementWorkInstruction`, `WatchSettingManager`, `HumanMachineInterface`, `PalletConveyorManager`, and `RobotManager`. Figure 6 depicts a sequence diagram for the robot assembly production process. The `ManagementWorkInstruction` and `HumanMachineInterface` components interface with humans, whereas the `PalletConveyorManager` and `RobotManager` interface with hardware devices. The normal operation of the robot assembly application involves the following steps:

1. The `ManagementWorkInstruction` asks for a good to be produced by sending an event to the `WatchSettingManager`.
2. The `WatchSettingManager` emits an event to the `HumanMachineInterface` asking to validate the order, the `HumanMachineInterface` accepts by invoking an operation on a CCM facet.

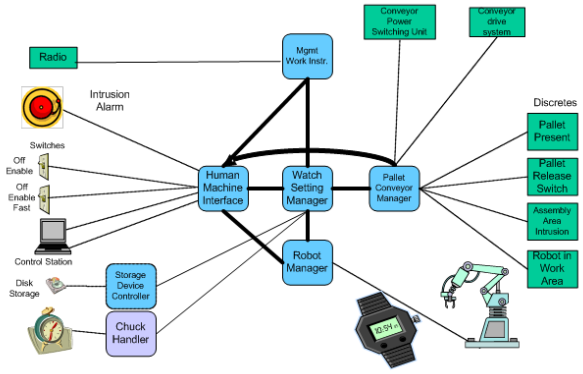

Figure 5: **Robot Assembly Model**

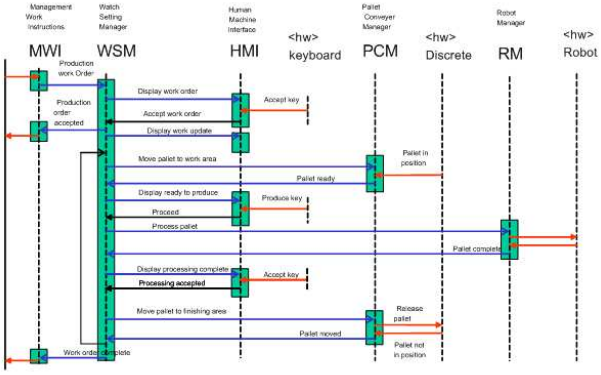

Figure 6: **Robot Assembly Production Sequence**

3. The `WatchSettingManager` notifies the `ManagementWorkInstruction` that the order was accepted, through another event, and then displays the work on the `HumanMachineInterface`.
4. The `WatchSettingManager` emits an event to the `PalletConveyorManager` to move the pallet into position, the `PalletConveyorManager` responds with another event.
5. The `WatchSettingManager` again asks the `HumanMachineInterface` for confirmation to perform a production step, the `HumanMachineInterface` accepts by invoking an operation on a facet.
6. The `WatchSettingManager` asks the `RobotManager` to process the pallet (event), the `RobotManager` performs the job and then responds via an event.
7. The `WatchSettingManager` sends an event asking `PalletConveyorManager` to move the pallet out of working area, the `PalletConveyorManager` notifies back with an event.
8. The `WatchSettingManager` displays the completed work to `HumanMachineInterface` via an event, the `HumanMachineInterface` validates the work via a facet operation call (steps 2-7 can be repeated if there are additional pallets to process).
9. The `WatchSettingManager` sends an event to the `ManagementWorkInstruction` notifying it that the requested job has been completed.

3.2 Applying MDD Techniques and Tools to the Robot Assembly Application

Now that we have outlined the structure and functionality of the robot assembly application, we illustrate how the application and integration of the CoSMIC and Cadena MDD toolsuites help to simplify various design decisions and validation activities. The remainder of this section shows how the semantic validation of models can help detect problems earlier in the software lifecycle, *e.g.*, immediately after the planning of the interfaces and before beginning the implementation of the business logic. Early detection of defects yields fewer code revisions, lower development costs, and shorter time to market. These semantic validations also ensure proper execution in mission-critical contexts, where run-time debugging alone is insufficient.

3.3 Choosing Appropriate Communication Mechanisms

Developers of large-scale component-based DRE systems must determine which communication mechanisms their components should use to interact. A key design decision is whether to use *facets/receptacles*, which define interfaces that initiate/process point-to-point synchronous operation invocations from other components vs. *event sources/sinks*, which indicate a willingness to exchange typed messages asynchronously with one or more components. Applying an MDD tool like PICML (Section 2.1) can help developers reason more effectively about which communication mechanism to select.

Figure 7 shows how we use PICML to model the structure and connections of the robot assembly scenario. By analyzing the PICML model, we can quickly determine that the return value of the facet invocation (point 2 above) is void and there are no out or inout parameters, but the operation is not oneway. Our analysis suggests that a more appropriate feature choice for this use case might be an event rather than a facet. Note that this analysis is much easier when using a graphical tool like PICML, rather than reading hundreds of lines of CORBA IDL3 code.

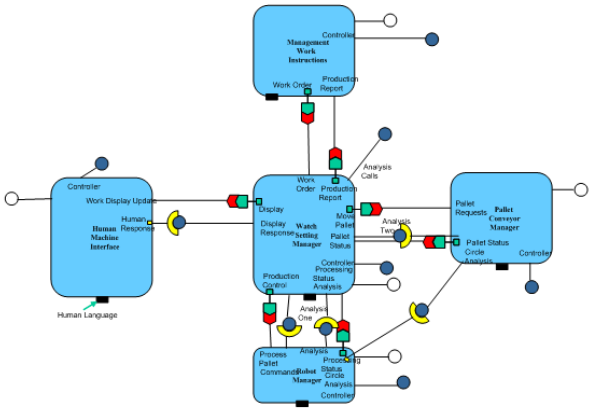

Figure 7: **Robot Assembly PICML Model**

3.4 Detecting Type Mismatches at Design-time vs. Run-time

As mentioned in Section 1, a key theme of MDD is achieving "correct by construction" programs, *i.e.*, MDD tools should detect many errors at design-time rather than run-time. To evaluate this in the context our robot assembly application and integrated CoSMIC/Cadena tools, we first tried to introduce a mistake in our assembly by connecting an additional port to a destination port of the wrong type. This mistake is detected by the GME constraint manager because two ends are not of the same type and are thus disallowed by the PICML paradigm.

To evaluate whether this error gets caught after export to another tool (which in this case is Cadena), we disabled the GME constraint manager in PICML temporarily and attempted to connect the `analysis` receptacle of the `WatchSettingManager` to the `controller` facet of the `PalletConveyorManager`, as shown in Figure 8 with a block arrow.

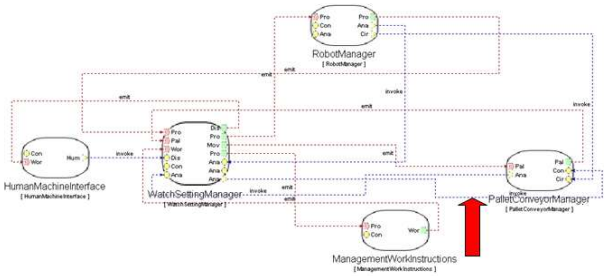

Figure 8: **Robot Assembly PICML Model with Error Introduced**

Now that our RobotAssembly is modeling using CoSMIC's PICML tools, we can export it to the Cadena environment to perform additional analysis and model checking. Figure 9 shows how Cadena detected the wrong connection, removed it upon import, and printed an error message

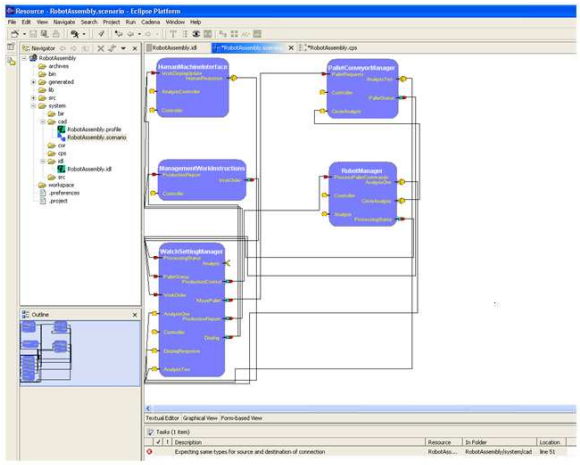

Figure 9: **Robot Assembly PICML to Cadena Model with Error Detected**

3.5 Advanced Model Checking for Component Assemblies

Another important capability provided by MDD tools is advanced model checking, such as the cycle check feature of Cadena, which is useful to reason about the possible deadlocks that may occur in a concurrent system. Since all components interact only with the `WatchSettingManager`, a possible cycle must pass through that component. Right clicking the `WatchSettingManager` component in the graphical scenario view of Cadena and selecting "cycle check" highlights two components of the assembly: the `HumanMachineInterface` and the `WatchSettingManager`, which form a cycle, as shown in Figure 10.

The cycle detection stops after the first detection, which is why only two components are highlighted in the figure. If we disconnect those two components and repeat the cycle check, however, other components will be highlighted. The `WatchSettingManager` affects and is affected by every other component, and this eventually means that every component is in the downstream path of every other component of the assembly.

Since we have at least one cycle we cannot be certain that deadlocks do not occur. The deadlocks for such a model are "implementation defined", which means that they might or might not be avoided with a wise implementation. In any case the system cannot be validated from a model point of

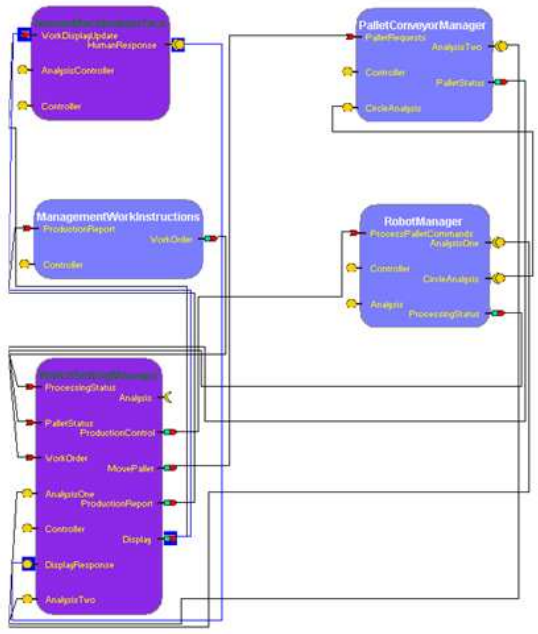

Figure 10: **Robot Assembly Modeless Cycle Detection in Cadena**

view. Examining the production sequence diagram in Figure 6 above, however, clearly shows that no deadlock can occur. This information clashes with the analysis from Cadena due to the fact that we did not specify modal information in our components, *i.e.*, different operational modes that can cause deadlocks are not captured in the models.

For the semantics shown in Figure 6's production sequence diagram, most components can remain stateless but at least two need a state: the `WatchSettingManager` and the `HumanMachineInterface`. The sequence diagram implicitly defines the following seven states for the `WatchSettingManager`: (1) *WaitingWorkOrder*, (2) *WaitingAcceptWorkOrder*, (3) *WaitingPalletReady*, (4) *WaitingProceed*, (5) *WaitingPalletComplete*, (6) *WaitingPalletMoved*, and (7) *WaitingProcessingAccepted*. In each of these states, no more than one input port affects output ports, and not all the output ports are affected (in facts never more than three for each mode). The other input and output ports behave as if they were disconnected. For the `HumanMachineInterface`, we need to specify that a *DisplayWorkUpdate* cannot trigger an *AcceptWorkOrder* otherwise a feedback cycle with the `WatchSettingManager` will evidently arise. So at least two states are needed, but it's better to specify all four semantically detectable states: (1) *WaitingNewWorkOrder*, (2) *WaitingDisplayWorkUpdate*, (3) *WaitingReadyToProduce*, (4) *WaitingDisplayProcessingComplete*.

More precisely, this behavior can be captured in a Cadena property specification (`.cps`) file shown below:

```
module RobotAssembly {
  component WatchSettingManager {
    mode status of {
       WaitingWorkOrder,
       WaitingAcceptWorkOrder,
       WaitingPalletReady,
       WaitingProceed,
       WaitingPalletComplete,
       WaitingPalletMoved,
       WaitingProcessingAccepted
    }
    init status.WaitingWorkOrder ;

    dependencydefault: none;
    dependencies {
      case status of {
        WaitingWorkOrder:
          WorkOrder -> Display;
        WaitingAcceptWorkOrder:
          DisplayResponse.WorkOrderResponse ->
            MovePallet, Display,
            ProductionReport;
        WaitingPalletReady:
          PalletStatus -> ProductionReport;
        WaitingProceed:
          DisplayResponse.ProductionReport ->
            MovePallet;
        WaitingPalletComplete:
          ProcessingStatus -> MovePallet;
        WaitingPalletMoved:
          PalletStatus -> Display;
        WaitingProcessingAccepted:
          DisplayResponse.ProductionReadyResponse ->
            ProductionReport, MovePallet;
      }
    }
  }
  component HumanMachineInterface
  {
    mode status of
    {
      WaitingNewWorkOrder,
      WaitingDisplayWorkUpdate,
      WaitingReadyToProduce,
      WaitingDisplayProcessingComplete
    }
    init status.WaitingNewWorkOrder;
    dependencydefault: none;
    dependencies {
      case status of {
        WaitingNewWorkOrder:
          WorkDisplayUpdate ->
            HumanResponse.WorkOrderResponse;
        WaitingDisplayWorkUpdate:
          WorkDisplayUpdate -> ;
        WaitingReadyToProduce:
          WorkDisplayUpdate ->
            HumanResponse.ProductionReadyResponse;
        WaitingDisplayProcessingComplete:
          WorkDisplayUpdate ->
            HumanResponse.PalletInspectionResponse;
      }
    }
  }
}
```

Introducing the `.cps` file into Cadena sets our modal specifications for this project. The remaining description in this section refers to the modal view of the scenario illustrated in Figure 11. The two components for which we have defined the

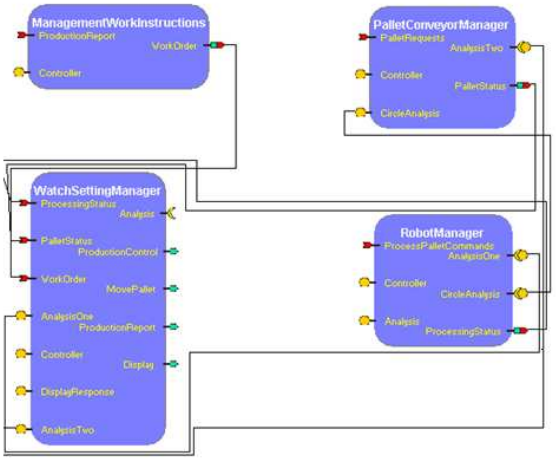

Figure 11: **Robot Assembly: Modal View in Cadena**

states must be set to a globally consistent state, *i.e.*, we cannot set the `WatchSettingManager` in the *WaitingPalletComplete* state while the `HumanMachineInterface` is in the *WaitingNewWorkOrder* state.[3] We therefore set the `WatchSettingManager` in the `WaitingAcceptWorkOrder` state and the `HumanMachineInterface` in the `WaitingDisplayWorkUpdate` state. As a result, only the connections that belong to the current mode will be shown (see Figure 11). Since the cycle analysis will detect any cycles in any of the modes, the current model can be validated against deadlocks.

There are certain conditions that cannot be validated against distributed deadlocks. To prove this, we connected the following additional port: `WatchSettingManger/Analysis` receptacle to `RobotManager/Analysis` facet. Note that the following two ports were already connected: (1) `RobotManager/CircleAnalysis` receptacle to `PalletConveyorManager/CircleAnalysis` facet and (2) `PalletConveyorManager/AnalysisTwo` receptacle to `WatchSettingManager/AnalysisTwo` facet.

We do not have any semantic or behavioral specifications for these analysis ports, so we must assume that operation calls on the facets can affect any analysis receptacle on the same component, and can happen in any mode of the three components. To reflect this scenario we add the following lines for the `WatchSettingManager` into the `.CPS` file:

```
...
dependencies {
  AnalysisOne.CallingBackTwo
              -> Analysis.CicrleCallOne,
                 Analysis.CallingBackOne;
  AnalysisTwo.CircleCallThree
              -> Analysis.CicrleCallOne,
                 Analysis.CallingBackOne;
  case status of
    ...
}
```

The resulting scenario shows a cycle illustrated in Figure 12 in at least one mode (and in this particular case, in all the modes). Thus, armed with the knowledge we have we can only assume that if there is a deadlock avoidance it has to be at the implementation level. This model cannot be validated against distributed deadlocks without further knowledge on the semantics at the modal level.

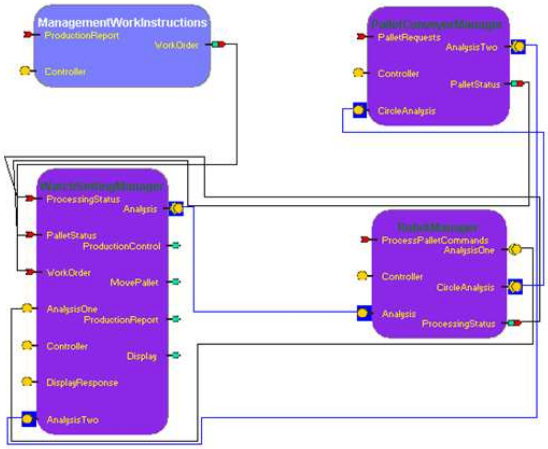

Figure 12: **Robot Assembly: Cadena Model After Circle Analysis**

4 Approaches to Integrating Modeling Tools for DRE Systems

Section 3 highlighted the interoperability between CoSMIC and Cadena in the context of a robot assembly application case study. More generally, however, multiple model-driven software development tools, each providing different capabilities like configuration, deployment, schedulability analysis, or model checking, are used by developers of mission-critical DRE systems software. An integrated tool chain that seamlessly integrates these multiple tools is needed to significantly enhance the development of DRE systems by addressing the production, validation, and verification capabilities that help to (1) identify bugs early in the development stage, (2) reduce total development costs, (3) reduce time to market, and (4) significantly increasing the reliability and safety-criticality of the DRE systems.

[3] A Bogor script could be used to to check the consistency of the assembly across all the consistent states and state changes of the components, but this is outside the scope of this book chapter.

This section describes key challenges that arise when providing an integrated modeling tool chain capability and discusses our solutions to resolve these challenges, which are incorporated within the CoSMIC MDD tool chain. The remainder of this section outlines the challenges faced while making multiple tools interoperable. We focus on three important challenges that are ordered by increasing complexity, where a subsequent challenge can be solved only when the previous challenge has been addressed.

Challenge 1: Identifying an Inter-Tool Communication Model

Context. Different modeling tools provide different capabilities. For the development of safety-critical DRE systems, however, it might be necessary to use the capabilities of a number of such modeling tools. What is required is a communication model for interoperability among various modeling tools that will allow back and forth seamless interworking among the tools. For example, in our case our prime goal was to be able to transfer the project back and forth between CoSMIC and Cadena, while keeping the user intervention to a minimum.

Problem. Seamless interoperability among tools is key ultimately to the success of DRE systems. However, such an interoperability may be hard to achieve. For example, CoSMIC and Cadena had nothing in common with regard to the type of project files they used. Moreover, under many aspects the two tools do not even capture the same type of information. The `.scenario` and `.IDL3` files of Cadena appear to have equivalent representation in CoSMIC, but even for such information there are subtle differences between the two tools. The `.profile`, `.cor` and above all the `.cps` files do not have any equivalent in CoSMIC. Conversely, 80% of the information in CoSMIC does not have an equivalent in Cadena.

Moreover, many of the standard interoperability solutions available for tool interoperability cater to a specific concern. For example, the Analysis Interchange Format (AIF) [35] developed by the DARPA MoBIES [36] program provides interoperability by promoting seamless exchange of only analysis data among tools. Similarly, the Hybrid Systems Interchange Format (HSIF) [36] (also developed in the MoBIES program) provides model exchanges for those systems that are modeled as hybrid systems but do not allow exchanging analysis information. On many occasions, an interchange format might not support a feature of a tool and thus a decision to avoid using that feature significantly decreases the value of the tool. Moreover, it is also not desirable to create a one-to-one solution since this approach does not scale as the number of tools with different capabilities increases. It is therefore necessary to develop a framework that allows seamless interoperability of all desired features among tools without creating point solutions.

Solution. We used the OTIF (Open Tool Integration Framework) [11], developed as part of the DARPA MoBIES program. OTIF is aimed at integrating tools that were not previously intended to interoperate. It consists of application-specific tool adapters, semantic translators, a backplane, and a manager. The backplane provides a communication and subscription/notification mechanism for other tools. The backplane also acts as a common repository for the data stored in a canonical syntactical format, but which may have different semantics.

The OTIF backplane supports standard CORBA [37] communication capabilities, thereby allowing distributed interoperability in addition to platform-independent interoperability. Custom tool-specific adapters must be developed by the DRE developers who wish to export desired tool-specific data to the backplane. Another tool wishing to interoperate with this tool must provide an adapter that converts data on the backplane to the format it desires. We have developed appropriate tool adapters for CoSMIC and Cadena, along with semantic translators that help these two tools to interoperate via the common OTIF backplane.

Challenge 2: Developing Mechanisms for Data Transforms Across Tools

Context. Tool interoperability imposes certain challenges when it comes to transfer of information from one tool to another. An important concern here is that of making the concerned tools understand each others data formats and their semantics. What is needed is a mechanism that allows exporting and importing tool-specific data using the tool-interchange framework, such as OTIF.

Problem. Since each modeling tool involved is catered to solving different aspects of DRE systems, each tool has its own format and semantics for data and their internal representation. There is minimal overlap between tools other than some common aspects pertaining to DRE systems. For example, in CoSMIC and Cadena, the common information is restricted to the fact that both tools are tailored to address concerns of DRE systems that use the CORBA Component Model (CCM) [17]. These commonalities are restricted to artifacts, such as the IDL descriptions and assembly information of components.

Whatever the location of storage of the tool-specific data, there is a need for at least one point during a round trip communication in which the information from CoSMIC and the information from Cadena has to be merged: the common subset of information has to be found and merged from the two tools, and in that point there is need for a translator which can understand both semantics.

Solution. The solution to resolve this problem should be based on identifying an information model, *e.g.*, based on XML, for the data that is needed for both the tools. For example, in CoSMIC this implies generating the information that are captured by the `.scenario` file in the form of XML descriptors; and then making a plugin for importing this XML format into Cadena. The reverse direction for this format incorporates the changes suggested by the Cadena analysis tools into the CoSMIC models. This approach therefore involves writing a model interpreter for CoSMIC to generate the information needed by Cadena for analysis. Cadena then reads this format and performs the analysis. If this format is rich enough, the analysis results can be put back into the same format. CoSMIC then imports this information back into the models. A similar approach could be used for the other files used by Cadena.

To achieve this behavior requires support by the communication model for seamless data interchange. OTIF accepts a Unified Data Model (UDM) [38] interface to the data for the backplane. UDM provides a development process and set of supporting tools that generate C++ programmatic interfaces from UML class diagrams of data structures. These interfaces and the underlying libraries provide convenient programmatic access and automatically configured persistence services for data structures as described in the input UML diagram. We leverage these capabilities for the data exchange between CoSMIC and Cadena.

A typical process of using the UDM is as follows:

1. A UML metamodel for the tool under consideration is created in GME using GME's UML modeling paradigm.
2. The information in the UML diagram is converted to an XML file using the GME interpreter supplied with the GME UML environment. The format of these XML files is UDM's representation of UML class diagram information.
3. The UDM executable program in the UDM framework is used to generate the paradigm-dependent API files.
4. The user includes these files, along with other, generic UDM headers libraries into a C++ project.

Fortunately, the modeling paradigms, such as CoSMIC's PICML, built using GME environment already expose a UDM interface without additional efforts. On the Cadena side, the Eclipse framework does not provide one, so we created a UML class diagram for the Cadena models. For this we leveraged GME to create a UML model that reflects the Cadena internal meta model and followed the process outlined above.

Challenge 3: Achieving Lossless Semantic Transfers of Data

Context. For any successful tool interoperability comprising data interchange, it is imperative that the exchanged data be transferred without loss of any semantic value. Only then can the full benefits of all involved tools be leveraged by DRE systems developers.

Problem. Lossless semantic transfers of tool-specific data is an arduous and complex task since many of the tools are tailored to address different aspects of DRE systems and therefore deal with different types of data having their own semantics and representation. Thus, there are a number of mismatches between data supported by individual tools and how they are managed by the tool. For example, we outline the differences between CoSMIC and Cadena formats below:

- Cadena scenario supports properties on connections (both event sources/sinks and invoke connections) while CoSMIC's PICML does not.
- PICML, being compliant to the CCM specification, supports all connection types, such as *emit*, *publish* and *invoke*, while Cadena does not distinguish between emit and publish.
- PICML supports QoS requirements on connections to be passed to the deployment run-time for validity checks and potential optimizations at deployment stage, while Cadena does not support this.
- PICML supports multiple senders and multiple receivers for a publisher/subscriber connection, while Cadena does not.
- Cadena supports only STRING, INT and BOOLEAN attribute types while PICML supports Boolean, Byte, ShortInteger, LongInteger, RealNumber, String, GenericObject, GenericValueObject, GenericValue, TypeEncoding and TypeKind.

With these constraints, a simple lossy export and import algorithm which would lose information not captured by the other side was easy to realize, however, would force the user to reenter information twice on either tool, thereby increasing the effort and also increasing the chance of inconsistencies in information maintained across the tools. What is desired, therefore, is an *enter-once* approach whereby once an information has been entered using either PICML or Cadena, the data transfer algorithm must preserve the data and its semantics in most but a few exceptional circumstances.

One approach to handle these issues is to merge the different data handled by individual tools to form a superset that is then maintained by the OTIF backplane. The issues that arise have to deal with the data representation in individual tools. For example, due to the monolithic nature of the GME project

files, the whole project file in CoSMIC is needed for a merging algorithm.

Solution. Transferring the complete set of information between the tools did not help to implement the merging algorithm, so we had to drop the idea. Moreover, an algorithm based on concentrating the information on a single tool at all times was preventing a possible future use of a shared OTIF backplane with simultaneous access by multiple developers. We therefore decided to perform a transfer of the .scenario, and implicitly the .profile files from Cadena. We also needed to transfer the information regarding the IDL3, which is discussed later. The merging was then decided to be performed at PICML side, generating the full .scenario and .profile files which would overwrite their previous versions at Cadena side.

The PICML data export to Cadena involve the following steps:

1. Every assembly makes a separate Scenario file. The full path name of the assembly from the RootFolder is encapsulated in a property called `PICML_pathname` which is stored by Cadena and eventually returned to PICML unchanged. This is needed to match the same source assembly on PICML side when reimporting. The assemblies which are in a folder named ”noexport” will not be exported nor reimported (and hence will not participate in the transfer).
2. Assembly-level properties are transfered to Cadena as scenario-level properties if the type is supported by Cadena, otherwise they are retained at PICML side.
3. All the PublishConnectors are checked and the newly created ones are flagged with a unique ConnectorID. The ConnectorID is put in a Requirement with a magic name which is disregarded by the DnC runtime.
4. All the PublishConnector are checked for the presence of a Requirement with another magic name called `CadenaProperties`. If found, all the properties encapsulated inside such a requirement are outputted as properties on the EventSource-to-Sink corresponding connection in Cadena (this supplies for the lack of properties on connectors at the PICML side).
5. All the components which have an output emit or an invocation connection are checked for a property with a magic name: CadenaEIProperties (where EI stands for Emit-Invoke). This property contains a string which is the dump of an XML file which can describe multiple properties for each receptacle to facet or event source to event sink emit connection being outputted in output from that component. The embedded file is parsed and the contained information is extracted and sent to Cadena. (This supplies for the lack of properties on emit and invoke connections at PICML side)
6. All the component instances are browsed and their name and type are transfered to Cadena. The attached properties are transfered to Cadena only if they are of a type which is supported by Cadena, otherwise they are retained on PICML side. For all the components, each connection to a remote port or to a PublishConnector is passed to Cadena.

At this point the XML file containing the information about the scenario (and implicitly about the profile) is sent to the OTIF backplane. At Cadena side it is fetched, de-encapsulated from XML and dumped to disk, possibly overwriting a preexisting version.

During Cadena export to PICML, the transfer over OTIF acts in the reverse way. The key points of the merging at PICML side are roughly as follows:

1. Using the PICML_pathname information, the same assembly of the export is matched so that the modifications can be performed in the correct place.
2. Based on the names of the component instance, the components are matched.
3. Based on the ConnectorIDs, the PublishConnectors are matched. At PICML side, the components and the PublishConnectors which have no match at Cadena side are considered deleted by the Cadena user and thus get destroyed at PICML side. The properties and requirement which only refer to those, also get destroyed.
4. The components and PublishConnectors at Cadena side which are unmatched at PICML side are considered newly created, and get created into PICML.
5. All the emit and invoke connections at PICML side are deleted, and are recreated new from the information at Cadena side.
6. All the properties on PICML components and at assembly-level get browsed. For those for which the type could have been passed to Cadena side, a match to the properties at Cadena side is attempted. If the match fails, those PICML properties are considered to be deleted by the Cadena user, so they are destroyed at PICML side.
7. On all the properties on components and scenario-level at Cadena side, a match is attempted at PICML side. If the match succeeds, the value is updated at PICML side, otherwise this is considered a new property created by the Cadena user so a new property gets created at PICML side.
8. The last two steps are repeated again for the properties on the PublishConnectors, with the difference that the match is attempted inside the Requirement called CadenaProperties if existing. The newly created properties also get created in there (if a requirement with such a name does not exist, it gets created and attached to the PublishConnector).

9. The two steps are repeated again for the properties on emit and publish connector, but this time the match is attempted on the XML content of the magic property CadenaEIProperties on the component which has got the outgoing emit or invoke connection. Again, this is created if needed.

To perform these steps, we used the GReAT (Graph Rewriting And Transformation) [39] tool. GReAT is a GME application that can be used to graphically define graph transformation among networks of objects which are accessible with UDM. GReAT shortens the development times significantly, since it is much more readable and maintainable than a normal third-generation programming language, such as C++ or Java. Both GME project files and XML files whose schema can be defined with an UML diagram can be accessed with UDM. This was the case, so GReAT appeared to be an optimal choice.

A GReAT transformation can be run interpretatively during the development and then it can be used to generate C++ (.cpp and .h) files which can be compiled for a release version of the transformation. The current version of this import export transformation counts more than 2,000 elements (graph pattern nodes) and 13,500 lines of C++ code. Figure 13 illustrates the architecture of this transformation process. A bidirectional *GReAT-based tool adapter and semantic translator* converts PICML assemblies to and from XML files conforming to the adopted interchange schema, which was chosen to be as near as possible to the semantics of Cadena `.scenario` and `.profile` files. The schema, known to the backplane, is used to read and validate the XML file upon arrival on the backplane. At every upload of a new interchange XML file onto the backplane, the Tool Adapters get notified of the availability of such new component assembly and are prompted for the download. On the Cadena side, a simpler *Java-based Cadena tool adapter* converts the XML to `.scenario` and `.profile` files and vice versa. The graph diff and merge algorithm is activated during the backplane-to-PICML import and is implemented inside the *GReAT-based PICML tool adapter and semantic translator*.

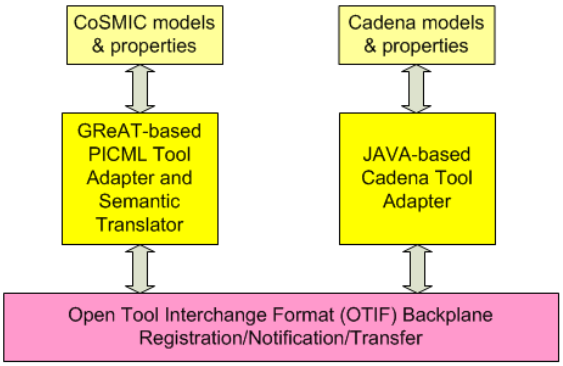

Figure 13: **CoSMIC-Cadena Interoperability via OTIF and GReAT**

5 Related Work

Our work on mode-based software extends earlier work on model-integrated computing (MIC) [40, 41, 42, 43] that focused on modeling and synthesizing embedded software. Examples of MIC technology used today include GME [20] and Ptolemy [44] (used primarily in the real-time and embedded domain) and MDA [2] based on UML [45] and XML [46] (which have been used primarily in the business domain). Previous efforts using MIC technologies for QoS adaptation have been applied to embedded systems comprising digital signal processors or signal detection systems [25, 47], which have a small number of fairly static QoS requirements. In contrast, our research on integrating CoSMIC and Cadena focuses on enhancing and applying MIC technologies at a much broader level, *i.e.*, modeling and controlling much larger scale DRE systems with multi-dimensional simultaneous QoS requirements.

Other related work on model-driven analysis and development is the Virginia Embedded System Toolkit (VEST) [48] and Automatic Integration of Reusable Embedded Systems (AIRES) [9]. VEST is an embedded system composition tool based on GME [20] that (1) enables the composition of reliable and configurable systems from COTS component libraries and (2) checks whether certain real-time, memory, power, and cost constraints of real-time and embedded applications are satisfied. AIRES provides the means to map design time models of component composition with real-time requirements to run-time models weaving timing and scheduling attributes within the run-time models. Although VEST and AIRES provide modeling and analysis tools for real-time scheduling and resource usage, they have not been applied to QoS-enabled component middleware, which is characterized by complex interactions between components, their containers and the provisioned services, and across distributed components via real-time event communication or request/response. Moreover, our research on the integration of CoSMIC and Cadena involves whole-system global analysis of large-scale DRE system for end-to-end timing constraints, as well as configuration and deployment.

Another project aimed at tool integration is the Open Tool Integration Framework (OTIF) [11], which is being developed at the Institute for Software Integrated Systems (ISIS). As opposed to our approach – where most features of Cadena and CoSMIC were developed separately and with no initial idea

of subsequent integration – OTIF explicitly provides a framework for integrating tools developed as part of the DARPA MoBIES project [36]. Their workflows are fairly complex and allow interoperations in multiple directions among the tools. These flows are not lossless in most cases, however, so they were able to obtain a closed ring of communication in one case only.

OTIF provides a communication framework with facilities for storing various versions of the same set of data written in different formats, subscription/notify mechanism, and automatic triggering of application-specific translators when certain data format are submitted to the backplane (data repository). However, OTIF requires that the actual (application-specific) semantic translators and the (application-specific) tool adapters for actually performing the communication and the translation be provided by the user. Our work represents an improvement over the previous uses of OTIF since with a wise choice of the interchange format and transformation semantics we are able to accomplish a closed-ring round trip and lossless communication between the two development environments that differ widely.

6 Concluding Remarks

Large-scale, distributed real-time and embedded (DRE) systems are increasingly being used to control critical aspects of global infrastructure. For instance, DRE systems are now deployed in commercial air traffic control, military systems, electrical power grid, industrial process control, and medical imaging domains. Some of the most challenging problems facing the DRE systems community are those associated with producing software for real-time and embedded systems in which computer processors may control physical, chemical, or biological processes or devices. In most of these systems, the right answer delivered too late becomes the wrong answer, *i.e.*, achieving end-to-end quality of service (QoS) in addition to functional correctness is essential. It is imperative therefore to validate and verify the DRE software for functional correctness and QoS properties.

Model-driven development (MDD) of software engineering processes is emerging as an effective paradigm for addressing the challenges of DRE systems. MDD is a software development paradigm that applies domain-specific modeling languages systematically to engineer computing systems. This paper describes the challenges in integrating modeling environments that have different foci, however, whose collective strengths resolve the challenges of developing DRE systems software. In this regard, we illustrate how we have integrated our CoSMIC DRE systems configuration and deployment modeling environment with the Cadena model checking tool using the OTIF tool integration environment.

The lessons learned using our integrated CoSMIC and Cadena tool chain for a robot assembly case study illustrated that:

- Not every MDD tool will offer same capabilities, but a collection of these is required to develop DRE systems, which is why a grade of interoperability between the tools is necessary.
- A partial and user-assisted interoperability, though easier to realize, would not guarantee against human mistakes during the exports from one tool and imports into the other. So, all efforts should be made for automating the communication process as much as possible. In particular, in order to guarantee consistency, the need for manual replication of information has to be avoided at all costs.
- A bidirectional communication among the tools is the only way that allows the user to edit the model locally, on whichever MDD tool is currently in use, while maintaining the ability to transfer the changes back to the other tools in an automatic manner, ensuring consistency.
- When achieving tool integration, the most important issues to consider are interoperation communication model, data interchange format, and solutions to achieve lossless data transforms.
- Complex transformation algorithms become more manageable when working at the meta-level. A couple of hundred of well structured graphical transformation rules are faster to write and easier to read and maintain than 13,500 lines of equivalent C++ code.
- To perform transformations at the meta level, the access to a graph structure representing the meta-model is required. When not provided directly, an intermediate step through an XML representation of the meta can be used instead.
- Most of the message flow in our robot assembly case study is asynchronous and most communication is performed via events, though some callbacks are performed via invocations on facet operations. This is hard to see from the production sequence diagram in Figure 6, but a MDD tool like PICML in CoSMIC and the Cadena's Scenario graphical view can show which communications are performed through event emissions and which are invocation on operations. This is shown with a MDD tool more clearly and concisely than reading tons of CORBA IDL3 interfaces raw. MDD tools also usually allow efficient browsing through the components and interfaces, up to the data types being exchanged.

Acknowledgments

The authors would like to acknowledge our collaborators Dr. John Hatcliff and Jesse Greenwald from Kansas State University for their help on the Cadena tool and with the Cadena/CoSMIC integration.

References

[1] Richard E. Schantz and Douglas C. Schmidt, "Middleware for Distributed Systems: Evolving the Common Structure for Network-centric Applications," in *Encyclopedia of Software Engineering*, John Marciniak and George Telecki, Eds. Wiley & Sons, New York, 2002.

[2] Object Management Group, *Model Driven Architecture (MDA)*, OMG Document ormsc/2001-07-01 edition, July 2001.

[3] Gabor Karsai, Janos Sztipanovits, Akos Ledeczi, and Ted Bapty, "Model-Integrated Development of Embedded Software," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 145–164, Jan. 2003.

[4] Jack Greenfield, Keith Short, Steve Cook, and Stuart Kent, *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*, John Wiley & Sons, New York, 2004.

[5] Aniruddha Gokhale, Krishnakumar Balasubramanian, Jaiganesh Balasubramanian, Arvind Krishna, George T. Edwards, Gan Deng, Emre Turkay, Jeffrey Parsons, and Douglas C. Schmidt, "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*, 2004.

[6] Institute for Software Integrated Systems, "Component Synthesis using Model Integrated Computing (CoSMIC)," www.dre.vanderbilt.edu/cosmic, Vanderbilt University.

[7] Nanbor Wang, Douglas C. Schmidt, Aniruddha Gokhale, Craig Rodrigues, Balachandran Natarajan, Joseph P. Loyall, Richard E. Schantz, and Christopher D. Gill, "QoS-enabled Middleware," in *Middleware for Communications*, Qusay Mahmoud, Ed. Wiley and Sons, New York, 2003.

[8] John Hatcliff, William Deng, Matthew Dwyer, Georg Jung, and Venkatesh Prasad, "Cadena: An Integrated Development, Analysis, and Verification Environment for Component-based Systems," in *Proceedings of the 25th International Conference on Software Engineering*, Portland, OR, May 2003.

[9] Sharath Kodase, Shige Wang, Zonghua Gu, and Kang G. Shin, "Improving scalability of task allocation and scheduling in large distributed real-time systems using shared buffers," in *Proceedings of the 9th Real-time/Embedded Technology and Applications Symposium (RTAS)*, Washington, DC, May 2003, IEEE.

[10] John A. Stankovic, Hexin Wang, Marty Humphrey, Ruiquing Zhu, Ramasubramaniam Poornalingam, and Chenyang Lu, "VEST: Virginia Embedded Systems Toolkit," in *Proceedings of the IEEE Real-time Embedded Systems Workshop*, London, UK, Dec. 2001, IEEE.

[11] Institute for Software Integrated Systems, "Open Tool Integration Framework," www.isis.vanderbilt.edu/Projects/WOTIF/.

[12] Paul Allen, "Model Driven Architecture," *Component Development Strategies*, vol. 12, no. 1, Jan. 2002.

[13] Object Management Group, "Model Integrated Computing PSIG," http://mic.omg.org.

[14] Lockheed Martin Aeronautics, "Lockheed Martin (MDA Success Story)," `www.omg.org/mda/mda_files/LockheedMartin.pdf`, Jan. 2003.

[15] Looking Glass Networks, "Optical Fiber Metropolitan Network," `www.omg.org/mda/mda_files/LookingGlassN.pdf`, Jan. 2003.

[16] Austrian Railways, "Success Story OBB," `www.omg.org/mda/mda_files/SuccessStory_OeBB.pdf/`, Jan. 2003.

[17] Object Management Group, *CORBA Components*, OMG Document formal/2002-06-65 edition, June 2002.

[18] Nanbor Wang and Christopher Gill, "Improving Real-Time System Configuration via a QoS-aware CORBA Component Model," in *Hawaii International Conference on System Sciences, Software Technology Track, Distributed Object and Component-based Software Systems Minitrack, HICSS 2003*, Honolulu, HW, Jan. 2003, HICSS.

[19] Jeff Gray, Janos Sztipanovits, Ted Bapty, Sandeep Neema, Aniruddha Gokhale, and Douglas C. Schmidt, "Two-level Aspect Weaving to Support Evolution of Model-Based Software," in *Aspect-Oriented Software Development*, Robert Filman, Tzilla Elrad, Mehmet Aksit, and Siobhan Clarke, Eds. Addison-Wesley, Reading, Massachusetts, 2003.

[20] Akos Ledeczi, Arpad Bakay, Miklos Maroti, Peter Volgysei, Greg Nordstrom, Jonathan Sprinkle, and Gabor Karsai, "Composing Domain-Specific Design Environments," *IEEE Computer*, Nov. 2001.

[21] A. Bondavalli, I. Mura, and I. Majzik, "Automated dependability analysis of UML designs," in *Proc. of Second IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, 1998.

[22] S. Gokhale, J. R. Horgan, and K. S. Trivedi, "Integration of specification, simulation and dependability analysis," in *Workshop on Architecting Dependable Systems*, Orlando, FL, May 2002.

[23] S. Gokhale, ""Cost–constrained reliability maximization of software systems"," in *Proc. of Annual Reliability and Maintainability Symposium (RAMS 04) (To Appear)*, Los Angeles, CA, January 2004.

[24] S. Gokhale and K. S. Trivedi, ""Reliability prediction and sensitivity analysis based on software architecture"," in *Proc. of Intl. Symposium on Software Reliability Engineering (ISSRE 02)*, Annapolis, MD, November 2002.

[25] Sandeep Neema, Ted Bapty, Jeff Gray, and Aniruddha Gokhale, "Generators for Synthesis of QoS Adaptation in Distributed Real-Time Embedded Systems," in *Proceedings of the ACM SIGPLAN/SIGSOFT Conference on Generative Programming and Component Engineering (GPCE'02)*, Pittsburgh, PA, Oct. 2002.

[26] Don Batory, Vivek Singhal, Jeff Thomas, Sankar Dasari, Bart Geraci, and Marty Sirkin, "The GenVoca Model of Software-System Generators," *IEEE Software*, vol. 11, no. 5, pp. 89–94, Nov. 1994.

[27] Ira Baxter, *DMS: A Tool for Automating Software Quality Enhancement*, Semantic Designs (www.semdesigns.com), 2001.

[28] T. A. Henzinger and S. Sastry, Eds., *Hybrid Systems: Computation and Control - Lecture Notes in Computer Science*, Springer Verlag, New York, NY, 1998.

[29] Object Management Group, *Deployment and Configuration Adopted Submission*, OMG Document ptc/03-07-08 edition, July 2003.

[30] Jeff Gray and Ted Bapty and Sandeep Neema and Douglas C. Schmidt and Aniruddha Gokhale and Balachandran Natarajan, "An Approach for Supporting Aspect-Oriented Domain Modeling," in *Proceedings of the 2nd International Conference on Generative Programming and Component Engineering (GPCE'03)*, Erfurt, Germany, Sept. 2003, ACM.

[31] Jonathan M. Sprinkle, Gabor Karsai, Akos Ledeczi, and Greg G. Nordstrom, "The New Metamodeling Generation," in *IEEE Engineering of Computer Based Systems*, Washington, DC, Apr. 2001, IEEE.

[32] Object Management Group, *Unified Modeling Language: OCL version 2.0 Final Adopted Specification*, OMG Document ptc/03-10-14 edition, Oct. 2003.

[33] Object Technology International, Inc., *Eclipse Platform Technical Overview: White Paper*, Object Technology International, Inc., Updated for 2.1, Original publication July 2001 edition, Feb. 2003.

[34] Robby and Matthew Dwyer and John Hatcliff, "Bogor: An Extensible and Highly-Modular Model Checking Framework," in *In the Proceedings of the Fourth Joint Meeting of the European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2003)*, Helsinki, Finland, Sept. 2003, ACM.

[35] Gabor Karsai, Sandeep Neema, Arpad Bakay, Akos Ledeczi, Feng Shi, and Aniruddha Gokhale, "A Model-based Front-end to ACE/TAO: The Embedded System Modeling Language," in *Proceedings of the Second Annual TAO Workshop*, Arlington, VA, July 2002.

[36] DARPA Information Exploitation Office, "Model-Based Integration of Embedded Software (MoBIES)," `www.darpa.mil/ixo/mobies.asp`.

[37] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, 3.0.2 edition, Dec. 2002.

[38] Magyari E. and Bakay A. and Lang A. and Paka T.and Vizhanyo A. and Agrawal A. and Karsai G., "UDM: An Infrastructure for Implementing Domain-Specific Modeling Languages," in *The 3rd OOPSLA Workshop on Domain-Specific Modeling, OOPSLA 2003*, Anaheim, CA, Oct. 2003, ACM.

[39] Karsai G. and Agrawal A. and Shi F. and Sprinkle J., "On the use of Graph Transformations in the Formal Specification of Computer-Based Systems," in *Proceedings of IEEE TC-ECBS and IFIP10.1 Joint Workshop on Formal Specifications of Computer-Based Systems*, Huntsville, AL, Apr. 2003, IEEE.

[40] Janos Sztipanovits and Gabor Karsai, "Model-Integrated Computing," *IEEE Computer*, vol. 30, no. 4, pp. 110–112, Apr. 1997.

[41] David Harel and Eran Gery, "Executable Object Modeling with Statecharts," in *Proceedings of the 18th International Conference on Software Engineering*. 1996, pp. 246–257, IEEE Computer Society Press.

[42] Man Lin, "Synthesis of Control Software in a Layered Architecture from Hybrid Automata," in *HSCC*, 1999, pp. 152–164.

[43] Jeffery Gray, Ted Bapty, and Sandeep Neema, "Handling Crosscutting Constraints in Domain-Specific Modeling," *Communications of the ACM*, pp. 87–93, Oct. 2001.

[44] J. T. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt, "Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems," *International Journal of Computer Simulation, Special Issue on Simulation Software Development Component Development Strategies*, vol. 4, Apr. 1994.

[45] Object Management Group, *Unified Modeling Language (UML) v1.4*, OMG Document formal/2001-09-67 edition, Sept. 2001.

[46] "Extensible Markup Language (XML) 1.0 (Second Edition)," `www.w3c.org/XML`, Oct. 2000.

[47] Sherif Abdelwahed, Sandeep Neema, Joseph Loyall, and Richard Shapiro, "Multi-Level Online Hybrid Control Design for QoS Management," in *Proceedings of the 24th IEEE International Real-time Systems Symposium (RTSS 2003)*, Cancun, Mexico, Dec. 2003, IEEE.

[48] John A. Stankovic, Ruiqing Zhu, Ramasubramaniam Poornalingam, Chenyang Lu, Zhendong Yu, Marty Humphrey, and Brian Ellis, "VEST: An Aspect-based Composition Tool for Real-time Systems," in *Proceedings of the IEEE Real-time Applications Symposium*, Washington, DC, May 2003, IEEE.