Abstract—Building High-integrity Distributed Real-Time (HDRT) systems requires a rigorous methodology to assist in the design and development of verifiable software. This paper describes an approach based on the Model-Driven Engineering (MDE) paradigm to ease the automatic generation of HDRT applications from high-level system models. Since those applications must be amenable to stringent timing analysis, such as the determination of worst-case execution time or schedulability analysis, we present the integration of a set of timing analysis tools with a toolset for MDE. In addition, this paper explores a new approach to integrating the real-time end-to-end flow model with the automatic generation of Ravenscar-compliant source code in distribution middleware.

I. INTRODUCTION

Developing High-Integrity (HI) systems is considered an arduous and challenging task accomplished with strong restrictions and standards compliance. To ease this task, industrial developers have been using the Ravenscar profile [1] in safety-critical domains in recent years. This profile is a subset of rules and coding guidelines for Ada language [2] that ensure certain properties amenable to static verification in order to assist in the development of highly efficient, reliable, and certifiable applications.

More recently, Model-Driven Engineering (MDE) paradigm software development [3] has attracted a high degree of interest. Under this approach a software system is built from a set of high-level models, which undergo a series of transformations that finally result in the executable code. The OMG has standardized a Model-Driven Architecture (MDA), which consists of a hierarchy of modelling levels:

- **Platform-independent models** (PIM) are used to specify a system independently of the computer platform on top of which it will run.
- **Platform-specific models** (PSM) are derived for specific execution platforms by transforming the PIM taking into account the particular characteristics of the chosen platform, abstracted in the form of a platform model (PM). The implementation code is automatically generated from the PSM, ideally with no human intervention.

Applying model-driven engineering to real-time systems requires the use of appropriate modelling languages and transformation rules. In particular, all aspects of concurrent and real-time behaviour which are relevant to the system being developed must be accurately described and preserved through model transformations. To this end, the use of a meta-model based on the Ravenscar profile [1] has proven to be a very useful concept, enabling static timing analysis methods to be applied to high-level system models [4]. In this paper, we describe an architecture that eases the automatic development of High-integrity Distributed Real-Time (HDRT) systems from their design to their implementation and schedulability analysis. The approach adapts and integrates a set of timing analysis tools into an MDE toolset for hard real-time systems.

On the other hand, although the Ravenscar profile has been used with great success in many critical applications, it has mainly been applied for the static analysis of applications running within a single or multiple nodes but without considering the communication networks in the analysis. However, due to the increasing interest within the community to apply this profile to distributed systems, several research works have tried to solve the problem of lack of support for developing distributed systems through the Ravenscar profile, for instance by restricting the Distributed Systems Annex (DSA) for Ada language [5] or by designing a new specific profile [6].

Traditionally, the schedulability analysis has been performed through the real-time transactional model [7], which is currently known as end-to-end flow model in the MARTE specification [8]. Therefore, this paper will explore the integration of the real-time end-to-end flow model into the development of HDRT applications by adapting the Ada API proposed in [9] to the restrictions included in the Ravenscar profile. This API allows the distribution middleware to support real-time capabilities using the end-to-end flow model. Furthermore, an implementation, which can be integrated with the proposed architecture and MDE toolset, is also provided as a proof of concept to validate the correctness of the approach.
The document is organized as follows. Section 2 reviews previous work on the design and development of Ravenscar distributed systems and the integration of the end-to-end flow model in distributed Ada. In Section 3, we describe the integration of timing analysis tools into an MDE toolset for hard real-time systems. In Section 4, we propose a new API to make the end-to-end flow model compatible with the Ravenscar profile. A brief description of how this API has been integrated into a Ravenscar compliant middleware and how to automate the configuration of a high-integrity distributed real-time application are pointed out in Section 5. Finally, Section 6 draws the conclusions.

II. RELATED WORK

Since the Ravenscar profile is widely used in the development of HRT systems, it seems natural to consider its possible use for HDRT systems. Previous research includes different approaches which can be classified as follows:

• Ravenscar and Ada DSA. These works are mainly focused on the adaptation of the Ada DSA to be Ravenscar compliant as discussed in [5] and [6]. The main disadvantage of this option is the lack of a standard real-time distributed framework.

• Ravenscar and a custom mechanism to perform the distribution. Under this approach the HDRT systems are built by automatically generating source code from architectural descriptions (i.e. system models). A representative work which proposes a complete framework to build HDRT systems can be found in [10].

This paper follows the second approach, extending the results of the ASSERT project, which is summarized in the next subsection, by adding the end-to-end flow model and a set of tools for timing analysis.

A The ASSERT development process

One of the main results of the ASSERT project is a new development process for distributed embedded real-time systems, and a set of methods and tools supporting the process. The resulting software is executed on a specialized platform, the ASSERT Virtual Machine (AVM) [11], which monitors and enforces real-time properties that cannot be guaranteed by static analysis at design time.

The ASSERT process is based on MDE concepts, and includes the following steps, which are carried out in an iterative way (see Figure 1):

• Modelling phase, in which a system model is built using three different model views: a functional view, an interface view, and a deployment view [12]. The former two views belong to the PIM abstraction level, while the latter one is part of the PSM.

• Model transformation. A concurrency view, also at the PSM level, is automatically generated from the model views. The concurrency view defines the concurrent and distributed architecture of the system in terms of the facilities provided by the underlying platform, e.g. threads, shared data, and messages.

• Feasibility analysis. Timing analysis methods are used to verify that the required real-time behaviour can be attained with the system architecture defined by the PSM. The results of the analysis can be used to iterate on the physical architecture in order to improve the system behaviour.

• Code generation. Source code is automatically generated from the concurrency view and distribution code is generated according to the virtual machine distribution model. Code is compiled to run on the virtual machine execution platform on each computer node.

The above process is supported by two sets of tools: the HRT-UML/RCM toolset [12], which uses an abstract version of the Ravenscar profile as a UML meta-model, and the TASTE toolset [13], which uses AADL [14] as a basic notation for embedding and integrating models generated with a variety of engineering tools. In the following, our work is based on TASTE as it currently has industrial support from ESA.

A key element of the ASSERT development process is the feasibility analysis phase. A prototype of the HRT-UML/
RCM toolset uses an extension of MAST\textsuperscript{1} [15] for timing analysis, while TASTE uses Cheddar [16] for the same purpose. However, Cheddar does not support distribution, which makes it unsuitable for the kind of systems at which the ASSERT process is aimed. In Section III we describe how MAST and other tools can be integrated with TASTE in order to fully support feasibility analysis for HDRT systems.

B The end-to-end flow model and the endpoints pattern

The work presented in [9] proposed a technique to express complex scheduling and timing parameters in a distributed system: the \textit{endpoints} pattern. This technique was aimed at supporting the event-driven end-to-end flow model, and was integrated in different middlewares (for Ada DSA and CORBA), using interchangeable scheduling policies [17]. The main features of this approach are described below:

- \textbf{Identification of two different schedulable entities}. For the processing nodes, \textit{handler tasks} intended to execute remote calls, and for the network, \textit{endpoints} or communication points which are used to transport messages through the network. Both schedulable entities are created explicitly with the appropriate scheduling information.

- \textbf{Definition of a set of abstract interfaces to ease the configuration of the end-to-end flows}. The APIs allow the integration of different scheduling policies (e.g. Fixed Priorities or EDF) and a free assignment of the scheduling parameters for the schedulable entities.

- \textbf{Definition of the Event_Id parameter}. This parameter identifies an end-to-end flow in its execution. Once the application has set the initial event at the beginning of the end-to-end flow, all the subsequent activities (tasks or messages) are scheduled according to the associated event at each moment.

- \textbf{Internal support for the end-to-end flow model}. Middleware is responsible for updating the scheduling parameters and managing the chain of events within the end-to-end flow, hiding the management of the real-time details from the software engineers. The real-time configuration of a distributed application following this model can be generated automatically using CASE tools.

This approach provides support for the real-time model by means of the specification of all the resources involved in an end-to-end flow through a set of Ada APIs:

- \textit{Event_Management} supporting the description of the end-to-end flow architecture via event transformations performed automatically at the transformation points (see [9]).

- \textit{Processing_Node_Scheduling} containing the operations to configure the handler tasks and to assign their scheduling parameters.

- \textit{Network_Scheduling} containing the endpoint configuration and assigning the scheduling parameters for the messages in the communication networks.

Section IV will describe how to make those APIs compatible with the Ravenscar profile, in order to enable TASTE to fully support the end-to-end flow model that is used by the schedulability analysis tools. This approach allows the scheduling and analysis of both processors and communication networks.

III. Integrating Feasibility Analysis in an MDE Software Process

A General approach

Feasibility analysis is a crucial activity in the development of high-integrity real-time systems. It allows software engineers to detect potential timing problems in early development phases, and take corrective actions on the system architecture in order to guarantee that the implementation will provide the required temporal behaviour. When an MDE approach is used, feasibility analysis has to be carried out on system models, as the implementation code is automatically generated from these. More specifically, it has to be performed on platform-specific models, as the temporal behaviour of a system is

\footnote{1. MAST is available at http://mast.unican.es/}
highly dependent on the characteristics of the execution platform. In the case of the ASSERT project, feasibility analysis is carried out on the concurrency view (Figure 1), which models the system under development as a set of concurrent tasks, shared data objects, and communication messages, all of them with timing and scheduling attributes such as periods, deadlines, and priorities.

In order to enable static timing analysis of concurrency models, several conditions have to be fulfilled by the platform and the concurrency model:

a) The execution platform must exhibit a predictable temporal behaviour. This is a required condition of worst-case execution time (WCET) calculations, which in turn are required by all timing analysis methods.

b) The concurrency model must be amenable to static timing analysis. Software features that cannot be analysed or do not have a bounded computation time must be excluded.

c) The distribution mechanism must also exhibit a predictable temporal behaviour. In particular, end-to-end message transmission times must be bounded, and the bounds must be known or computable from other system parameters.

The ASSERT development process fulfils all the above conditions. Condition (a) is ensured by the use of the LEON architecture for the hardware platform, and the ORK+ kernel component of the ASSERT Virtual Machine [18]. Condition (b) is fulfilled by the use of the Ravenscar computational model as an underlying meta-model at all the abstraction levels. Condition (c) is ensured by the code generation and distribution tools, which have been extended to support the end-to-end flow model as described in Section V.

B A toolset for feasibility analysis of HDRT

The TASTE toolset [13] is an integrated set of tools supporting the different ASSERT model views. It uses AADL [14] as a common language for providing an architectural framework for the different components of a system. The toolset includes tools for integrating data specifications, as well as functional code derived from engineering modelling tools (such as Simulink or SCADE) into AADL models. C or Ada functional code can also be manually written by the developers.

A tool is used to derive wrappers for functional code from the interface view, thus ensuring consistency at the interface level. An automatic tool is used to perform the model transformations required to produce the concurrency view from the PIM views and the deployment view. The Ocarina tool [10] is then used to generate concurrency and distribution code skeletons, into which the functional code is embedded (see Figure 2).

The concurrency view is described in AADL, and consists of a set of model entities that can be directly implemented in terms of the underlying execution platform. In AADL terms, the valid model entities at this level of abstraction are periodic and sporadic threads communicating by means of protected data objects. Additional model entities providing access to the middleware functionality are also part of the concurrency view. Compliance with the Ravenscar computational model is ensured by restricting the AADL constructs at the time of generating the concurrency view. Custom-defined AADL attributes are used to describe timing properties, such as periods, deadlines, execution times, and priorities.

Performing feasibility analysis on the concurrency view requires two kinds of analysis tools. First, the Worst-Case Execution Time (WCET) of each model component must be
analysed. Once all model elements are annotated with their respective WCET values, schedulability analysis can be carried out for all execution flows.

We have experimented with two kinds of WCET analysis tools which can be used in this framework. AiT\(^1\) uses a processor model, whereas RapiTime\(^2\) uses timing measurements on the actual hardware together with coverage analysis to provide accurate estimates of the execution times of sequential code blocks \[19\]. RapiTime was selected as it is more convenient for automatically generated code. The tool provides WCET estimates that are recorded in a database from which a results report is produced.

RapiTime requires the code to be instrumented. We have modified the original Ocarina scripts in order to include timing information in the code. The WCET data obtained with RapiTime are fed back into the concurrency model by using a tool, rapitime2aadl, which has been built by us to this purpose. The data are inserted in the AADL concurrency model as attributes for threads and operations (Figure 3).

Several tools can be used for schedulability analysis as well. Cheddar \[16\] was previously integrated with TASTE, but it only supports centralized systems and is thus inappropriate for our purposes. On the other hand, MAST \[15\] supports response time analysis of distributed systems. Therefore, we decided to use MAST for the feasibility analysis phase of the TASTE toolset.

MAST uses a model of real-time systems which is based on a set of basic concepts: processing resources, scheduling servers, shared resources, operations, and end-to-end flows, each of which has different kinds of attributes. In order to use it with TASTE, we have developed a tool that generates a MAST model from an AADL concurrency view model. The tool is called aadl2mast, and generates one or more MAST model elements for each AADL component.

The conversion tool generates some elements in the MAST model for each component in the concurrency view. Table 1 shows some examples of translation relationships between AADL components and MAST entities.

The rapitime2aadl and aadl2mast tools are based on an AADL parser developed with lex/yacc. Both tools use this parser as a common frontend to parse the concurrency view model, although they do it in different ways. The aadl2mast frontend searches the AADL components partially listed in Table 1, from which the backend component of the tool generates the MAST model entities. On the other hand, rapitime2aadl uses the parser to find the names of the subprograms that have WCET attributes in the AADL model. The tool then looks for the corresponding WCET values obtained by RapiTime, which are stored in a MySQL database, and fills out the corresponding AADL attribute values in the concurrency view model. In this way, the frontend, i.e. the AADL parser, can be reused for other possible mappings, although the backends are specific for MAST and RapiTime.

MAST can be used to apply different timing analysis techniques to the real-time system model. It produces as a result worst-case response times for all end-to-end flows, as well as scheduling parameters such as priorities. The results are translated again to the AADL code as timing and scheduling attributes. A tool named mast2aadl is under development for this purpose.

The current prototype supports only single-node implementations, as the TASTE toolset does not manage the priority-based networks supported by MAST. Support for distribution can be added taking into account the integration with the end-to-end flow model as it is described in Sections IV and V, where we will propose the strategy to coordinate middleware and schedulability analysis tools based on the end-to-end flow model and including the communication networks. According to the latter, the analysis tools included in MAST could be applied to the overall distributed system (processors and communication networks), following the holistic approach, introduced by Tindell and Clark in \[20\], that considers the entire system as a whole.

### IV. Integration of the Real-Time End-to-End Flow Model with the Ravenscar Profile

As we have previously mentioned, the endpoints pattern is not compatible with the Ravenscar profile and should be adapted to be used in HDRT systems. In particular, the following restrictions must be considered:

- **Focused only on FIFO _Within_Priorities_ dispatching policy.** The approach proposed in \[9\] is independent of the selected policy. However, this flexibility does not violate this restriction because the choice of available policies remains implementation-defined. Furthermore, keeping this flexibility enables future profile extensions as proposed in \[21\].

---

1. AiT is available at http://www.absint.com/ait
2. RapiTime is available at http://www.rapitasystems.com/rapitime

### Table 1. AADL-MAST translation

<table>
<thead>
<tr>
<th>AADL component</th>
<th>MAST entity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>Processing_Resource</td>
</tr>
<tr>
<td>Thread</td>
<td>Scheduling_Server</td>
</tr>
<tr>
<td></td>
<td>Operation (enclosing)</td>
</tr>
<tr>
<td></td>
<td>Transaction (end-to-end flow)</td>
</tr>
<tr>
<td>Protected data</td>
<td>Shared_Resource</td>
</tr>
<tr>
<td>Passive data</td>
<td>Operation</td>
</tr>
<tr>
<td>Subprogram</td>
<td>Operation (simple)</td>
</tr>
</tbody>
</table>

---

\[19\] AADL component MAST entity
• The set of tasks in the system is fixed and created at library level. The Processing_Node_Scheduling interface provides operations to create handler tasks at configuration time, which it is not Ravenscar compliant, and thus requires a review of the proposal in [9].

• Tasks have static scheduling parameters. Although the current approach allows handler tasks to update their scheduling parameters at runtime according to the retrieved Event_Id, this feature is not compatible with Ravenscar and must be disabled. However, the transmission of the Event_Id parameter remains necessary in order to share the same handler task among multiple end-to-end flows.

Although this model does not violate any further Ravenscar restrictions, there are some other aspects that middleware implementations should take into account:

• Prevent the use of task attributes. The middleware implementation for Ada DSA and CORBA [17] uses task attributes to store the Event_Id parameter.

• All tasks are non-terminating. Operations to destroy handler tasks, which are provided by the Processing_Node_Scheduling interface, must be disabled.

Each of these considerations must be addressed within the set of interfaces shown in Figure 4. This figure presents the Ada package hierarchy for the new API. The modifications proposed over the original API in [9] are detailed as follows.

A Event Management Interface

Final users should configure the sequence of events within an end-to-end flow, and the middleware will be in charge of automatically setting the appropriate event at the transformation points of the remote call as shown in [9]. This interface is Ravenscar compliant and therefore it does not require any modification.

B Processing Node Scheduling Interface

Handler tasks are responsible for awaiting remote requests and processing them. As we stated before, the proposal in [9] to create and manage handler tasks relied on the dynamic creation of tasks which is forbidden for Ravenscar systems. The new API uses a set of Ada packages instead: a Processing_Node_Scheduling package to perform the registration and identification of tasks in the system, and a set of child packages to create tasks with the appropriate scheduling parameters.

One child package per scheduling policy is required (see Figure 4). Since handler tasks must be created explicitly at library level, the new API considers the creation of tasks through a generic package which has been demonstrated to be a suitable approach [22]. This generic package includes the following parameters and operations:

- Task scheduling parameters: The scheduling parameters are set statically via a pragma.
- Task properties: Basic properties associated with a task (e.g. the stack size).
- Wait_For_Incoming_Events: Procedure to wait for an incoming event at the specified endpoint. A single handler task could process several requests matching different end-to-end flows.
- Create_RPC_Endpoint: Function to create the endpoint where the handler task will listen for incoming requests.
- Process_Event: Procedure to process the received message and perform the associated job.

Furthermore, this generic package can be completed by including several optional subprograms; for instance, to execute the basic initialization operations required within each middleware implementation or to execute recovery procedures when any error is detected. Finally, the body of each child package contains the operations associated with handler tasks. Each handler task consists of a loop with three single actions: wait for an incoming event, perform the event transformation and process the event.

C Network Scheduling Interface

The overall response time of a distributed system is strongly influenced by the underlying networks and therefore networks must be scheduled with appropriate techniques. This API addresses this aspect by making the communication endpoints visible, and by associating scheduling parameters to the messages sent through them. The approach in [9] is already compatible with Ravenscar systems and thus it can remain unaltered.

V. IMPLEMENTATION OF THE REAL-TIME END-TO-END FLOW MODEL WITHIN A HI MIDDLEWARE

This section describes how the endpoints pattern has been integrated into Ocarina and PolyORB-HI [10]. Figure 5 shows the architecture of Ocarina which comprises two different parts: a frontend, which processes the system model described in the input file, and a backend, which implements the strategies to generate the source code for different targets. The current version supports the AADL modeling
language as input and several targets, such as those based on the PolyORB-HI middleware, as output.

PolyORB-HI is a lightweight distribution middleware compatible with the restrictions specified by the Ravenscar profile. It is distributed with the Ocarina tool as an AADL runtime that provides all the required resources (i.e. stubs, skeletons, marshellers and concurrent structures) to build high-integrity distributed systems. The current software release provides three runtimes depending on the target system: PolyORB-HI-C, PolyORB-HI-Ada and PolyORB-HI-QoS.

To validate the proposed approach, Ocarina and PolyORB-HI have been extended to provide a new backend or code generation strategy called PolyORB-HI-Endpoints (see Figure 5), which, together with the implementation of the endpoints pattern, presents the following features:

- **Automatic code generation.** The proposed approach has been seamlessly integrated within the Ocarina architecture by developing a new backend to automatically generate the source code based on the real-time end-to-end flow model.

- **Automatic management and transformation of events.** It comprises an extension of PolyORB-HI to provide marshalling and unmarshalling primitives for the Event_Id parameter, and the implementation of the Event_Management interface which allows the middleware to internally manage the event associations at the transformation points.

- **Fixed-priority scheduling for the processors.** According to the Ravenscar profile, it includes an implementation of the Processing_Node_Scheduling interface that has been developed for fixed-priority policy.

- **Fixed-priority scheduling for a new network service.** A new network service has been developed and integrated in PolyORB-HI to use the handler tasks to directly wait on the net for incoming requests, thus avoiding I/O decoupling. Furthermore, a fixed priority version has been implemented for the Network_Scheduling interface.

- **Adaptation code to integrate the middleware internals into the proposed model.** Middleware built on top of the endpoints pattern requires some glue code to handle and map its internal structures consistently and integrate the management and utilization of the communication endpoints.

- **Real-time configuration code.** A configuration file including the initialization code that has to be run at start up time is needed. This file is composed of the calls to the API operations for creating the handler tasks and endpoints required to support the end-to-end flows of the application.

A prototype implementation of the PolyORB-HI-Endpoints backend on x86 architecture and a UDP-based network has been developed as a proof of concept. The network uses the 802.1p specification [23] to prioritize different message streams. This prototype requires the configuration file to be generated by hand, although automatic generation of the real-time configuration code from the PSM is planned.

VI. CONCLUSIONS AND FUTURE WORK

The paper presents an approach that facilitates the automatic development of HDRT systems. The TASTE toolset has been taken as a starting point for supporting the approach. The integration of RapiTime (for WCET estimation) and MAST (for schedulability analysis) with TASTE provides the necessary support to perform a static verification of the end-to-end deadlines in HDRT systems. Furthermore, the work described in this paper integrates the real-time end-to-end flow model with the automatic generation of Ravenscar-compliant source code and the distribution middleware.

The tools presented in this paper (aadl2mast, rapitime2aadl, and the PolyORB-HI-Endpoints backend) are prototypes which have been developed with the sole purpose of demonstrating the validity of the approach. The conversion tools between AADL and MAST only include the MAST features related to processors, and we plan to add the fixed-priority based communication networks used in MAST as future work; the new PolyORB-HI-Endpoints backend provides the basis to support it. Other plans for the near future include completing the tools in order to provide full round-trip feasibility analysis considering the WCET data, and the automatic generation of endpoint configuration files from AADL models.

The described toolchain has been evaluated to be used for the development of the on-board computer software of the UPMSat-2 micro-satellite with promising results. The satellite attitude control algorithms are being developed with

1. http://www.idr.upm.es/tec.espacial/06_UPMSAT.html
Simulink, and generated code will be integrated with other manually coded software components using TASTE. The transformation tools, together with RapiTime and MAST, will be used to perform timing analysis of the system, in order to assess their use on a real system.

REFERENCES


