Embedded software development
White paper
December 2009

Rational. software

IBM

AUTOSAR: from concept to code.

Introducing support for behavior modeling tool (BMT) implementation, providing automated code and internal behavior generation for AUTOSAR models

Contents

Introduction

As modern vehicles continue to grow in complexity, the automotive industry has been challenged to find more efficient development practices. As a result, the industry established the Automotive Open System Architecture (AUTOSAR) development partnership, which specifies a platform and methodology to improve reuse, quality and efficiency in automotive electronic and electric (E/E) development. Nowadays AUTOSAR is rapidly being adopted by automotive engineers in companies around the world.

In this white paper, we will show how IBM Rational® Rhapsody® software, as part of the IBM Rational software platform for automotive systems, enables you to model E/E architectures, electronic control units (ECUs) and their embedded software and now also support the automatic generation of both C source code and AUTOSAR internal behavior artifacts for AUTOSAR software components (SWCs).

A few definitions

An AUTOSAR application consists of many components, including the software architecture that's composed of AUTOSAR atomic software components and the behavior or implementation of each one.

Unified Modeling Language (UML)/Systems Modeling Language (SysML) are widely used modeling languages for specifying and designing software applications and include powerful behavioral modeling notations such as statecharts and activity diagrams.

Highlights

UML and SysML are widely used standards and enable the use of domain-specific concepts, making them an excellent fit for AUTOSAR.

Describing an AUTOSAR system model using UML/SysML

Graphical models are a powerful method for capturing an AUTOSAR system, providing the benefits of abstraction from a design standpoint while making it easier to communicate the design intent. UML and SysML are widely used standards and enable the use of domain-specific concepts, making them an excellent fit for AUTOSAR. The method we describe here uses a UML/SysML–based AUTOSAR profile with domain-specific AUTOSAR terminology to express the SWCs, interfaces, ECUs and system topology using various diagrams.

Capturing and specifying the system under development

The process for capturing the system under development (SUD), which might be an entire E/E system for a vehicle or a subsystem out of the E/E system, includes the following steps:

The process for capturing a system under development includes a number of steps.

- *Capture the SUD from a functional viewpoint using the software component diagram. This diagram allows engineers to define the SWC and the communication between them over the AUTOSAR Virtual Functional Bus (VFB). We assume that the requirements capture and analysis as well as the logical analysis on vehicle level have been already completed.*
- *Capture the electrical architecture for the SUD using an ECU diagram and a topology diagram. First, the ECU diagram captures the ECUs by describing the hardware configuration details for each individual ECU type and its ports. Second, the topology diagram is used to define the physical (electrical) architecture of the SUD, using ECU instances (described in ECU diagrams) and the CAN, LIN and Flexray buses that connect them.*

Highlights

- *Partition the SUD by mapping the SWCs to the ECU instances in the topology diagram. Thus the SWCs become atomic software components.*
- *Define the scheduling of the different runnable elements for each atomic software component using internal behavior diagrams. An internal behavior diagram specifies the internal scheduling of runnable entities and the interface to the RTE for a specific software component. The actual implementations of these runnable entities (code bodies or the algorithms themselves) are either directly captured in code or defined using a behavioral modeling tool.*

An innovative approach to AUTOSAR implementation modeling

An alternative approach to explicit definition of the AUTOSAR internal behavior of an SWC is to integrate the AUTOSAR system architecture with UML/SysML implementation and design models using Rational Rhapsody software.

An alternative approach to explicit definition of the AUTOSAR internal behavior of an SWC is to integrate the AUTOSAR system architecture with UML/SysML implementation and design models using Rational Rhapsody software. This approach involves using UML/SysML designs, which hide the complexity of the explicit internal behavior design and implementation from the designer. AUTOSAR internal behavior artifacts can be generated automatically by Rational Rhapsody software as needed, based on the actual implementation and supportive tables. Thus, software engineers are able to reduce the amount of time they have to spend handwriting code.

Using an abstraction layer to mask complexity

An abstraction layer allows users to implement an atomic software component with a UML/SysML C-implementation model by defining a mapping between the two domains — the UML/SysML C-implementation domain and the AUTOSAR domain.

An abstraction layer allows users to implement an atomic software component with a UML/SysML C-implementation model by defining a mapping between the two domains—the UML/SysML C-implementation domain and the AUTOSAR domain—based on a one-to-one correspondence between UML/SysML active elements and AUTOSAR runnable entities.

Highlights

RIMBs act as bridges between the AUTOSAR domain and the UML/ SysML implementation domain.

Rational Rhapsody implementation blocks (RIMBs) act as bridges between the AUTOSAR domain and the UML/SysML implementation domain, which the following section explains in more detail.

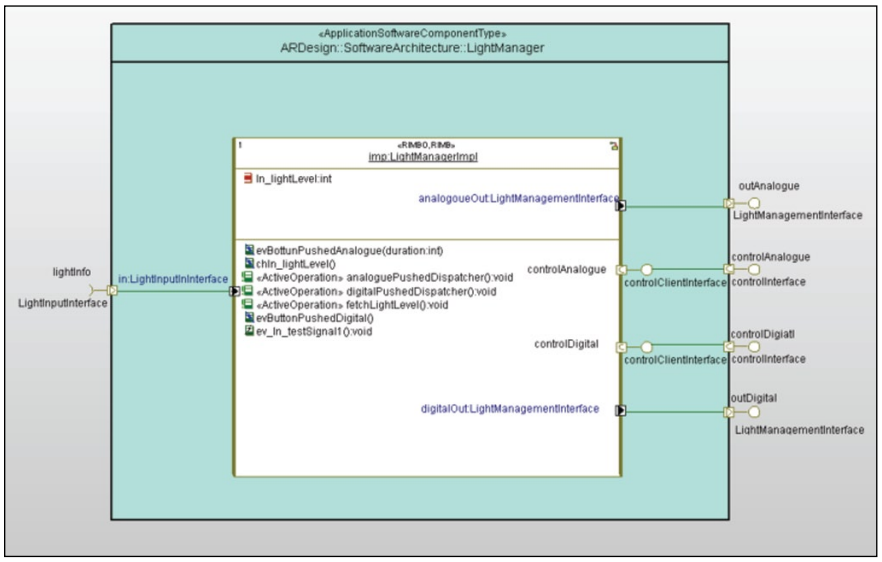

Figure 1: The RIMB acts as an abstraction layer, mapping the green-colored AUTOSAR SWC LightManager to the white-colored SysML/UML implementation LightManagerImpl.

Highlights

RIMBOs are responsible for defining an AUTOSAR internal behavior and implementation definition, along with actual C-code implementation for that atomic software component.

Using Rational Rhapsody implementation blocks to implement AUTOSAR application software components

RIMBs are used by instantiating them as parts inside atomic software component types. Those parts are called Rational Rhapsody implementation block objects (RIMBOs). They are responsible for defining an AUTOSAR internal behavior and implementation definition, along with actual C-code implementation for that atomic software component. In particular, they are responsible for:

- *Receiving data arriving on receiver ports.*
- *Handling the received data.*
- *Sending data over sender ports.*
- *Calling operations on client ports.*
- *Implementing operations on server ports.*
- *Performing calculations and algorithms needed for or during the above, including executing statecharts.*

RIMBs may have (regular) operations and active operations, an active elements table, exclusive area settings, interrunnable attributes and (regular) attributes.

RIMBs may have (regular) operations and active operations, an active elements table, exclusive area settings, interrunnable attributes and (regular) attributes. An operation, active or not, may call other nonactive operations in its implementation. When an operation on a RIMB is designated as active, it will be mapped to AUTOSAR runnable entities. Each active operation defines execution policy as either periodic or asynchronous. In case it is asynchronous, the user will specify an additional activation policy via a dedicated table called the Access & Activation table, described later.

Highlights

RIMB ports, interfaces, port accessors and handlers

Like an AUTOSAR atomic software component, RIMBs have two kinds of ports with corresponding interfaces: RIMB Sender/Receiver (S/R) ports, provided or required, and RIMB Client/Server (C/S) ports, provided or required. The attributes of a RIMB S/R port can be either a C-type or a Rational Rhapsody Event. The RIMB C/S interface cannot have attributes. Both kinds of ports, related attributes and operations can be detailed in Rational Rhapsody software so that all kinds of AUTOSAR internal behavior can be realized.

Rational Rhapsody software can automatically generate several helper functions, freeing developers from having to write specific RTE APIs and facilitating easy handling of received data and maintenance of the application code.

Rational Rhapsody software can automatically generate several helper functions, freeing developers from having to write specific run-time environment (RTE) application programming interfaces (APIs) and facilitating easy handling of received data and maintenance of the application code. These functions are Receivers, Handlers, Receiver-and-Handlers, Senders and Callers.

- ***Receivers.*** *Generated for RIMBs and meant to be called by active operations. A receiver is generated per attribute on a receiver port of the RIMB. It calls RTE APIs to retrieve the value of the AUTOSAR data element connected to that attribute.*
- ***Handlers.*** *Provides useful functionality that users may call in active operations following a call to a receiver.*
- ***Receiver-and-Handlers.*** *Generated per attribute on a receiver port of a RIMB.*
- ***Senders.*** *Generated per attribute on a sender port of the RIMB.*
- ***Callers.*** *Generated per operation on a client port of the RIMB.*

Users can control the behavior of the Rational Rhapsody tool to generate a variety of helper functions.

Note that the user may control the tool behavior to generate any and all of those helper functions.

Highlights

The new code generation capabilities for AUTOSAR in Rational Rhapsody software relieve the software engineer from having to handwrite code, thus reducing the risk of manual errors.

RIMB attributes and interrunnable variables

RIMBs may have regular attributes. All operations, active or not, may access those attributes. Users who want their code to work regardless of whether the SWC type is singly or multiply instantiated should access attributes only via generated setters and getters. A RIMB attribute may be designated as an inter-runnable variable. In that case, Rational Rhapsody software will generate AUTOSAR interrunnable variables, and the setters and getters will use the RTE API to access it. This is another example of how the new code generation capabilities for AUTOSAR in Rational Rhapsody software relieve the software engineer from having to handwrite code, thus reducing the risk of manual errors.

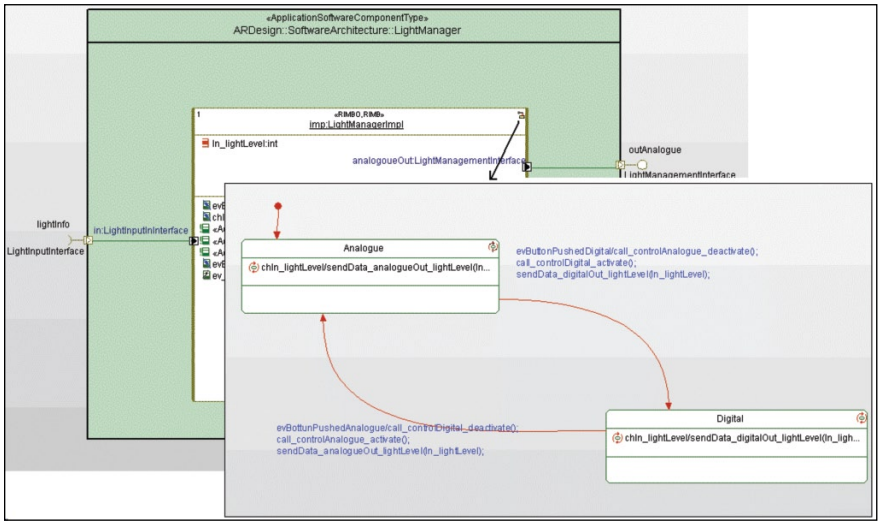

Figure 2: The RIMB logic may be defined using a statechart, leveraging the automatic generated accessors, which directly link the RTE APIs with the model.

Highlights

By linking the RTE APIs with the model, you can reduce the likelihood of coding errors.

```
... /* Accessors code: */
static Std_ReturnType receiveAndHandleEvent_in_buttonPushedAnalogue(void) {
 int duration;
 Std_ReturnType retval = Rte_Receive_lightInfo_buttonPushedAnalogue(&duration);
 if(retval == RTE_E_OK){
            RiCGEN(&LightManagerImpl, evBottunPushedAnalogue(duration));
 }
 return retval;
}
...
static Std_ReturnType call_controlAnalogue_activate(void) {
 return Rte_Call_controlAnalogue_activate();
}

... /* Generated code for the Statechart of LightManagerImp */
static RiCTakeEventStatus LightManagerImpl_rootState_dispatchEvent(void * const void_me, short id) {
...
 switch (LightManagerImpl.rootState_active) {
.....
 case evBottunPushedAnalogue:
 {
 RiCSETPARAMS(&LightManagerImpl, evBottunPushedAnalogue);
 {
 /*#[ transition 2 */
 call_controlDigital_deactivate();
 call_controlAnalogue_activate();
 sendData_analogueOut_lightLevel(In_lightLevel);
 /*#]*/
 }
 LightManagerImpl.rootState_subState = LightManagerImpl_Analogue;
...
```

Figure 3: The generated code for the statechart and the generated accessors directly link the RTE APIs with the model, thus reducing the likelihood of coding errors that can be common in manual programming.

The Access & Activation table

The Access & Activation table specifies the attributes for each active element.

The Access & Activation table specifies the following for each active element:

- *The activation policy of the active element*
- *The data elements on ports to which the active element has access*
- *The operations on ports to which the active element has access*
- *The interrunnable variables to which the active element has access*
- *The exclusive areas in which the active element runs or which it may enter*

Highlights

The table has the following columns:

- ***Active element:*** *denotes an operation that will be the entry point for a runnable entity*
- ***Activation policy:*** *relates to the RTE event to trigger the activation of the operation*
- ***Context:*** *defines the RIMB ports, to be used as context for the selection of relevant interfaces, to define the specific elements in the "elements" column*
- ***Elements:*** *defines the specific related elements in the interfaces implementing the ports*

The information defined in the Access & Activation table drives some of the generation of AUTOSAR internal behavior and implementation artifacts.

Generating the AUTOSAR internal behavior and implementation artifacts

The information defined in the Access & Activation table will drive some of the generation of AUTOSAR internal behavior and implementation artifacts. So Rational Rhapsody software will generate a runnable entity for every active element. All access elements, RTE events and wait points are automatically generated. The tool will analyze the concrete connectors between the RIMB ports and where they are used and will define the specific AUTOSAR internal behavior artifacts accordingly. In some cases, the user may choose to use existing elements in the AUTOSAR design instead of generating new ones. This capability allows users to mix manually defined artifacts with artifacts generated by Rational Rhapsody software.

Highlights

AUTOSAR internal behavior artifacts are derived from the Access & Activation table.

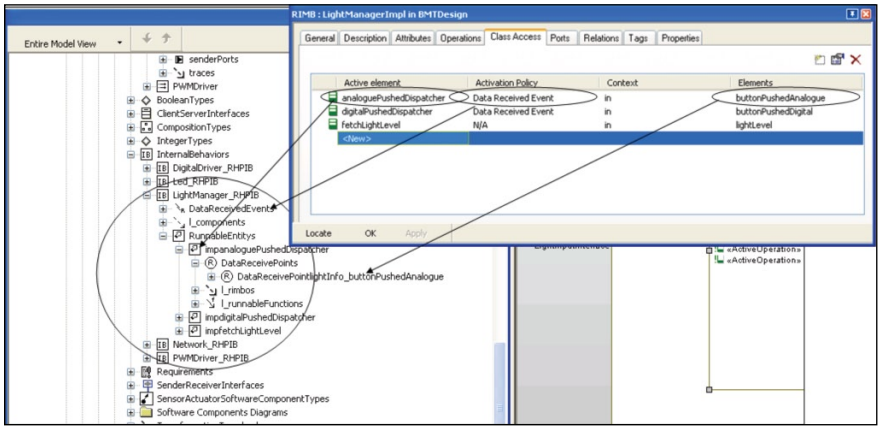

Figure 4: The AUTOSAR internal behavior artifacts shown in the browser at the left-hand side are derived from the Access & Activation table, shown here for the LightManager RIMB type.

Gaining flexibility and supporting reuse with the abstraction layer

The design concepts in Rational Rhapsody software support a great deal of flexibility in defining the behavior for an AUTOSAR SWC while allowing access to all AUTOSAR internal behavior features.

The abstraction layer connects the AUTOSAR and UML/SysML C-implementation domains, based on the concept of UML/SysML active elements. The contained active elements of a RIMB map one-to-one with an AUTOSAR runnable entity. The design concepts used support a great deal of flexibility in defining the behavior for an AUTOSAR SWC while allowing access to all AUTOSAR internal behavior features. The AUTOSAR activation policy for active elements that may use any AUTOSAR RTE event as a trigger is fully supported. The category of the generated AUTOSAR runnable entity may be controlled via the receiveMethod tag and interrunnable variables. Exclusive areas are also supported. The active operations and the RIMB doExecute() can contain any code, which adds further flexibility. In addition, as the RIMB may refer to other UML/SysML implementations simply by instantiating those, the possible reuse level is high and extends beyond AUTOSAR implementations.

Conclusion

We have shown how an AUTOSAR system model design can be defined using a UML/SysML design and implementation tool, and then how the actual implementation of the various AUTOSAR atomic software components can be defined using the commonly used UML/SysML design and implementation concepts, in an integrated single environment.

We believe that the suggested approach has inherent benefits associated with it. By using a well-known and commonly used design language such as UML/SysML, you can reduce the need for highly skilled AUTOSAR design engineers to perform common engineering tasks and software development. You can also create an environment that supports greater reuse of intellectual property and makes it easier to develop clear documentation of it. We see it as a useful and practical engineering solution on top of the explicit definition of the internal behavior and implementation sections defined in the AUTOSAR standard.

Rational Rhapsody software makes it easier for you to gain the benefits from the presented approach. The code generation capabilities for AUTOSAR leads to improved code quality, faster time to market and improved developer productivity and efficiency. Furthermore, because Rational Rhapsody software is tightly integrated with the IBM Rational software platform for automotive systems, you can gain additional benefits, such as integrated requirements sharing across globally distributed teams and projects and improved collaboration in the product development process.

For more information

To learn more about how IBM can help you model and generate AUTOSAR-compliant C code, contact your IBM representative or IBM Business Partner, or visit:

ibm.com/software/rational/solutions/automotive/invehicle.html

IBM Corporation
Software Group
Route 100
Somers, NY 10589
U.S.A.

Produced in the United States of America
December 2009

RAW14190-USEN-00