Leveraging Java and JDT for Scripting to Enable AUTOSAR Engineering à la MATLAB

Stephan Eberle, Geensys

About…

- Stephan Eberle
  - Paris, France
  - Development lead of Geensys’ AUTOSAR Builder product
  - Committer for EMFT Teneo component
  - 10+ years experience in OO and automotive embedded
  - Frequent presenter at conferences and events
- Geensys
  - Global Embedded Electronics & Networked SYstem Solutions
  - Embedded system engineering and IP
  - Embedded system engineering tools
  - Consulting services
Outline

• AUTOSAR from 4000 meters
• Success factors for AUTOSAR tool environments
• Definition of an open AUTOSAR tool platform
• Building extensions using the MATLAB/Simulink way
• Conclusion

Objectives of AUTOSAR

= AUTomotive Open System ARchitecture
• Mastering the ever increasing number of functions and complexity of E/E systems
  • Getting out of "one function – one ECU" trap
  • Transparent collaboration between OEMs and suppliers and reduction of integration pain
  • Reuse of E/E functions
    ▪ In different product lines (high end .. low cost)
    ▪ With different hardware platforms/communication systems (protection of investment)
How it works

- Decoupling of application software from ECU hardware and communication systems through VFB/RTE
- Formal (XML-based) description of
  - Application software components
  - ECU hardware
  - System topology
- Arbitrary mapping of application software components to ECUs
- Formal (XML-based) configuration of standardized basic software modules
  - Operating system
  - Device drivers
  - Communication drivers

Déjà vu?

- JRE
  - Fixed function set
  - Implemented according to JSRs
  - Dependent on machine
  - Independent of application software
- RTE
  - Only functions requested by application software
  - Generated according to system description
  - Dependent on machine
  - Dependent on application software
  - Resource consumption optimized
Geensys' Eclipse-based AUTOSAR Tooling

AUTOSAR Builder
- AUTOSAR Authoring Tool (AAT)
- ECU Extract (EEX)
- Generic Configuration Editor (GCE)
- AUTOSAR Simulator (ASIM)
- SW-C Conformance Validation Tool (SCVT)
- Runtime Environment Generator (RTEG)
- Configuration Code Generator (CCG)

Eclipse RCP/Plug-in SDK

Demo
- Software component description
- Software component validation
- ECU mapping and extract
- BSW configuration
Outline

• AUTOSAR from 4000 meters
• Success factors for AUTOSAR tool environments
• Definition of an open AUTOSAR tool platform
• Building extensions using the MATLAB/Simulink way
• Conclusion

Scope of AUTOSAR

• Standardization in AUTOSAR is focused on
  • Enabling exchangeability and reuse of E/E functions
  • Decoupling of application software from hardware & communication

• Consequence
  • Covers only a part of the design phase of E/E systems
    ▪ Very detailed, grey box like description of component boundaries
    ▪ But not: internal structure and behavior of components
  • Missing connection to upstream and downstream development activities, e.g.
    ▪ High-level functional design
    ▪ Component implementation
To do list for AUTOSAR adaptors

- Coping with migration issues, e.g.
  - Import/conversion of legacy description formats
  - Conversion between AUTOSAR releases
- Handling of domain/vendor-specific aspects within AUTOSAR, e.g.
  - Automatic configuration of BSW modules from ECU Extract
  - Restriction of AUTOSAR design activities wrt custom development process and roles
- Integration with non-AUTOSAR development tools, e.g.
  - Model-based design tools
  - Build tool chains

What does all that mean to AUTOSAR tooling?

- AUTOSAR tool environments
  - Will never be complete out of the box
  - Must be highly adaptable to domain/vendor-specific contexts
- Idea
  - Provide an open and extensible AUTOSAR tool platform as common core for off-the-shelf and custom tool components
  - Final environment must enable both creating and using AUTOSAR tool components

Solution 1:

Extend Eclipse Platform towards an AUTOSAR tool platform
What does all that mean to AUTOSAR tooling? (cont’d)

- Creation of custom AUTOSAR tool components requires
  - Detailed knowledge of domain/vendor-specific context
  - Strong expertise of tool development infrastructure and technology

- Role/competence gap: involved domain/vendor experts typically
  - Have general programming knowledge
  - But are rarely true tool smiths and Eclipse experts

- Idea
  - Provide a really simple way to create custom AUTOSAR tool components by
    - Hiding the complexity of underlying development infrastructure and technology
    - Enabling custom tool components to be developed on a minimum learning curve basis

Learning from a similar case: MATLAB/Simulink

- Interesting characteristics
  - Powerful model-based engineering environment
  - Well-known, long lasting … successful
  - Based on an open and extensible core being maintained and evolved by The MathWorks
  - Anyone else can build custom “ToolBoxes” and “BlockSets” on top of it using a simple C++ like scripting language

Solution 2:

Provide AUTOSAR tool platform with a MATLAB/Simulink-like scripting facility
Outline

- AUTOSAR from 4000 meters
- Success factors for AUTOSAR tool environments
- Definition of an open AUTOSAR tool platform
- Building extensions using the MATLAB/Simulink way
- Conclusion
Outline

• AUTOSAR from 4000 meters
• Success factors for AUTOSAR tool environments
• Definition of an open AUTOSAR tool platform
• Building extensions using the MATLAB/Simulink way
• Conclusion
Translating MATLAB/Simulink principles to Eclipse

1. MATLAB Programming language (M-files)
   - Groovy, Ruby, Monkey?
     - Too emerging, not established enough
     - Difficult to get placed in vertical industries
   - Java!
     - Broadly known, well-understood
     - “Old” enough to be adoptable in vertical industries

Translating MATLAB/Simulink principles to Eclipse (cont’d)

2. Toolboxes (“bundle” of M-files)
   - Plug-ins?
     - Statically handled OSGi bundles
     - Not deployable/executable on the fly
     - Separate environments for development and execution
     - Handling not obvious for non tool smiths, creates need for dedicated trainings
   - Plug-lets!
     - Dynamically handled OSGi bundles
     - Deployable/executable on the fly and within same environment
     - Write scripts and see their effect with minimum roundtrip time
     - Intuitive handling, enables to get started very quickly
New Scripting/Plug-let Environment for Eclipse

• Main Features
  • Java-based “scripts”
  • Bundled up in plug-lets
  • Dynamic compilation and deployment
  • Editable and executable within same environment

• Basic Building blocks
  • Plug-let Runtime
  • Plug-let Development Tools
  • Scripting Runtime
  • Script Development Environment

Plug-let Runtime and Development Tools

• Conversion of plug-in projects to plug-let projects
  • Via context menu action in Package Explorer
  • Identification through new header in the manifest file: `Automatic-Registering-Services: true`
  • Dynamic installation and start of bundle

• BundleListener
  • Scans starting plug-lets for scripts (classes implementing `IScript`)
  • Loads, instantiates, and registers each script as OSGi service

• ResourceChangeListener
  • Updates (i.e. reinstalls and restarts) plug-lets when contained scripts have been created or changed
Plug-let Runtime and Development Tools (cont’d)

- Plug-lets view
  - Displayed information:
    - Plug-lets with nested categories and scripts
    - Activation state of plug-lets and scripts
  - Supported operations:
    - Start/stop of plug-lets and scripts
    - Import/export to/from workspace
    - Creation of new plug-lets and scripts

Scripting Runtime

- Reads scripts from OSGi service registry
- Contributes scripts to Eclipse UI (where applicable)
  - Users just write script code and don’t need to (but still can) cope with Eclipse extension points
- Runs scripts according to execution semantics of underlying script type
- Currently supported script types and execution semantics:
  - **ActionScript**
    - Launched on demand via context menu option
  - **BatchScript**
    - Launched via Ant task at command line; with or without arguments
  - **RulesScript**
    - Runs some validation rules and reports results via Problems view
  - **CallbackScript**
    - Launched automatically in response to some user interaction
  - **ViewScript**
    - Generates some graphical output in a dedicated Display view
Script Development Environment

• Largely reuse popular Eclipse JDT:
  • Script editing
    % JDT Java Editor
  • Dynamic script compilation
    % Incremental JDT Java Builder
  • Script debugging
    % RCP application reusing JDT debug kernel
    % Communicates via socket connection with host environment

Demo: Import/conversion of legacy description formats

• Starting point: Illustrative example of proprietary embedded application description format (“Sphinx”)
• Objective: Import/conversion to AUTOSAR system description
Conclusion

• Need for **customizability:**
  - AUTOSAR is used in many different domain/vendor-specific contexts
  - It is unlikely that a single AUTOSAR tool can satisfy all needs
  - Providing an open and extensible tool platform is crucial for achieving the required degrees of customizability

• Need for **simplicity:**
  - People working on domain/vendor-specific extensions are normally not tool smiths
  - MATLAB/Simulink shows how customization of complex engineering environments can be kept simple
  - Proposed Java-based scripting/plug-let approach tends to add this capacity to Eclipse

Outlook

• **OSEE**
  - Platform for full life cycle embedded engineering environments
  - Geensys’ AUTOSAR TDK could be regarded as extension covering automotive E/E function design
  - We would be happy to collaborate!

• **e4**
  - JavaScript support, PDE support, development tools
  - Geensys’ Java-based scripting/plug-let environment seems like an early case study for that
  - We would be happy to contribute!