ABSTRACT

Integrated modular avionics (IMA) platform development should be supported by a model-based harmonized tool environment. This means the integration of tools of different type, origin and purpose. Eclipse’s flexible modular architecture seems ideal as the framework for such a harmonized IMA development environment. It is evaluated how Eclipse could practically fit into this position. The requirements like concurrency, different process roles, and multiple tools are mapped to the Eclipse framework. In addition, open-source extensions for model-based engineering, application development, are integrated in the harmonized tools chain. In order to test the performance, openness and compatibility of Eclipse and the tools from the IMA development process a prototype of six current and future tools is build. The tools are integrated into a common Eclipse instance. The integration covers a common human-machine interface (HMI) and model-based data exchange. The selected tools cover all working areas and tool types expected in the IMA development process. The prototype reveals typical shortcomings of Eclipse as a multi-tool host. The criticality of shortcomings is evaluated, and rules for a harmonized Eclipse environment for IMA development are derived.

INTRODUCTION

Aircraft development is a complex, highly concurrent, and a, therefore, time-consuming and costly process. Corporation between different roles of engineers in this process is mainly document-centered. A speed-up of the process is assumed by introducing model-based engineering concepts, and a formal model-based data exchange between the different roles. In addition, current documents might only accompany the process as documentation, e.g. for certification. Therefore, a new way of incorporating the tools and data of planning, application development, configuration, simulation, integration, and testing for the second generation of integrated modular avionics (IMA2G) is needed. The result shall be a harmonized tool-chain for the so called distributed modular electronics (DME). In order to ensure the openness of such a platform, no proprietary application shall be created but an available application shall be reused.

The software framework for this purpose shall not only provide the interfaces for tools but a software platform, which enables the best suitable cooperation of tools and developers. The Eclipse framework [15] is the most suitable candidate. Advantages like high modularity and platform independence promote Eclipse as the framework for such an integrated development environment. However, Eclipse has to fit into the historically evolved process and the grown tool world. Therefore, a concept study is carried out, in which six current and future IMA tools are integrated into a common development environment for IMA2G, called DME development environment (DDE). The participating tools are representative for all technical and process related issues probably encountered; i.e. among the selected tools are certification-critical, proprietary, commercial-of-the-shelf (COTS), Eclipse-based, and native binary applications. Beside the tool integration, it is investigated how Eclipse supports the development process, i.e. role’s responsibilities and concurrency. Moreover, it is investigated which and how open-source source products, e.g. TOPCASED [22], might support the IMA development process.

This article is organized as follows. In chapter two the technical and process requirements for a common IMA development environment are collected. Based on this a concept for using Eclipse as the framework for tool integration is given. Chapter three
comprises the results of the DDE prototype study. Guidelines for a harmonious integrated tool-chain based on Eclipse are derived in chapter four.

METHOD

The open-source software development environment Eclipse could provide a harmonized human machine interface (HMI) for all tools and the data backbone. Moreover, the Eclipse Modeling Framework (EMF) [12] provides a complete solution to support the process with domain-specific modeling and formal model transformations.

The TOPCASED project started using Eclipse as the base for a development environment for critical systems. This approach shall be extended to the full IMA platform development process. The overall goals are similar to what the automotive industry’s AUTOSAR [19] is aiming at. Eclipse, therefore, needs to provide a solution for all requirements originating from the process and related tools.

REQUIREMENTS

The requirements for an harmonized IMA development environment originate, first, from the scope of IMA development. The scope defines which areas of work and which tools are covered. Second, additional requirements originate from process, which defines the participating roles, their responsibilities, and the data flow during development.

IMA platform development

The IMA platform development process comprises the development of equipment design, platform architecture (systems and IMA), configuration and software. It includes the integration process, which produces the load together with the configuration and the software. Validation and verification contain activities, which qualify and certify the integrated platform, i.e. tests and requirement validation. Validation work is carried out during the design as well as in the integration phase. The verification is carried out during the integration phase only. Following main products result from the development process

- Equipment hardware
- Architecture
- Configuration
- Software
- Validation and verification results
- Load

The harmonized IMA development environment shall cover the full platform development process.
Scopes and tools

In IMA platform development six areas of work are identified [6]. Those are planning, function development, configuration, simulation, integration, and testing. Each area of work contains domain-specific tools. Figure 2 depicts the tools in the different working areas. For technical and business reasons there can be more than one instance of each tool. All areas of work shall be covered by the integrated tool chain.

In addition, the tools operate on overlapping data sets. The data exchange between tools is currently mainly document centered. A development speed-up and less errors are expected by introducing formal data exchange. A harmonized IMA development environment shall provide the backbone for data exchange between different tools as well as different process roles, and concurrently working engineers. Figure 3 depicts the optimal data exchange between the tools of the IMA development process. The harmonized environment shall support an automated data exchange.

Figure 1: Platform Development Process

Figure 2: Different areas of work in IMA2G development and the related tools
Process and roles

Several Roles are involved in the IMA development process. Members of each role have a certain responsibility and, therefore, a certain tools set. In a harmonized IMA development environment the member of each role shall have his tools and data available and shall be guided in his tasks. This comprises restricted access to data, tool and tool functionality. Figure 4 depicts the tool coverage of integrator, function supplier and module supplier.

ECLIPSE

Eclipse is originally a Java development environment. Because of its high flexibility and modularity, however, it is used for other purposes as well, e.g. for other programming languages, web development, hardware development, and software engineering. Eclipse is based on the OSGI standard for modular software [5]. Therefore, Eclipse itself is only a plug-in loader and a set of plug-ins the base functionality. In addition, the Eclipse workbench is a convenient platform independent HMI. The Eclipse public license (EPL) [13] allows free private and commercial use. Of special interest in the scope of an IMA platform development environment is how tools can be integrated, and how the can exchange data in an automated way.

Tool integration

Techniques to integrate applications to Eclipse are listed below. It is distinguished between the integration of Eclipse based extensions and external stand-alone applications.

- Eclipse plug-ins:
  - Copying plug-ins to the plugins folder in the Eclipse root directory makes them available after the next restart of Eclipse, if all requirements are satisfied.
• Links to a remote directory containing plug-ins are contained in the links folder. Links are text files containing the path to the remote directory.

• Using Eclipse software update service calculates the dependencies of plug-ins and downloads and installs dependencies required plug-ins automatically. Update sites are either located in the internet or local in the form of zip archives. [11]

• External applications:
  
  o A file type association of the host operating system (OS) enables opening files with a format unknown by Eclipse itself with the associated application.
  
  o The external tools dialog of Eclipse provides a direct way to start general applications outside Eclipse. It is possible to define arbitrary command-line arguments and environment variables with static arguments or arguments dynamically generated from the workspace content.
  
  o Build chains called by the CDT build action by the invocation of a make script [17].
  
  o A wrapper Eclipse plug-in, which starts an external process with the application to be integrated. This way data exchange during run-time between Eclipse and the external application is possible.
  
  o Building Java applications or plug-ins in Eclipse is usually carried out with. Apache ANT [21] scripts provide a general target based build mechanism, which can be used to build non-java applications as well. In addition, the <exec> command provides the possibility to execute an external application.

Data interconnection

Tools using formal meta-model based data can exchange date using model transformation languages. Eclipse provides good support for the Query View Transformation (QVT) [20] language [18, 8]. Tools using textually stored data, e.g. source code, can exchange data with model-based tools with text-to-model transformations, e.g. text-to-model transformations, however, have the additional effort of creating a meta-model for the textual data.

File-based concurrent working of developers can be realized using version control systems providing an Eclipse integration, e.g. CVS [2], SVN [1], GIT [3], and Mercurial [4]. When using EMF-based data the Connected Data Objects (CDO) [9] can be used to provide synchronization and version control on model object level.

Process roles

Process roles can be considered by defining the set of plug-ins (tools) required for the task of a specific role. The user in a certain process role will than only install the plug-ins of his role. In addition, specialized Eclipse perspectives for each task can be created. Restricted data access has to be implemented in the data model or data storage.

DDE-PROTOTYPE

A prototype of an integrated IMA development environment shall help investigating the stability, compatibility, performance, scalability of Eclipse practically. Six current and future IMA tools have been selected, such that all areas of work are covered, and that each way of tool integration is tested. Special focus is on COTS and certification critical tools. The tools integrated are

• A newly developed Eclipse EMF-based modeling and evaluation tool for IMA platform.

• Two COTS software development environments (SDE) for A653 compatible operating systems from different vendors. Both are Eclipse rich client platform (RCP) applications.

• A certified configuration table code generator as a binary stand-alone application.

• A newly developed configuration editor for IMA2G modules. This is a stand-alone Java application.
An Eclipse RCP based COTS test suit for TTCN-3 standardized testing.

Eclipse 3.4 has been selected for integration, since it is compatible to all selected Eclipse-based tools. In each case the original tool version is used for integration. Stand-alone tools are installed separately and are than integrated into Eclipse. Table 1 summarizes the properties of the integrated tools and the selected way of integration. In addition, Eclipse open-source extensions EMF [12], C development environment (CDT) [10], and TOPCASED [22] are integrated.

**Table 1: Classification of tools integrated in the DME development environment prototype. Last column gives the integration method for Eclipse.**

<table>
<thead>
<tr>
<th>Tool</th>
<th>Eclipse based</th>
<th>Certified</th>
<th>External</th>
<th>COTS</th>
<th>Integration Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform Evaluation</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>Plug-in copy</td>
</tr>
<tr>
<td>Configuration Editor</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td>File-type association</td>
</tr>
<tr>
<td>Code generator</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td>External tools</td>
</tr>
<tr>
<td>Software Development Environment A</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td>Eclipse packet manager</td>
</tr>
<tr>
<td>Software Development Environment B</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td>Linked environment</td>
</tr>
<tr>
<td>Test Suite</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td>Plug-in copy</td>
</tr>
</tbody>
</table>

**RESULTS**

The concept resulted in a prototype that integrated the selected tools as far-as possible, which states Eclipse good technical flexibility. However, during integration stumbling-stones showed up, which prevents a ready to use DDE; e.g., technically Eclipse enables the integration of native plug-ins as well as external tools. Latter, however, might break platform independence. Native Eclipse applications suffer from version incompatibilities of the framework or commonly used extensions, like editors. Even if the technical integration succeeds there can be problems from the end-uses view, beginning from the installation procedure, over licensing, up to general usability. Table 3 summarizes all identified issues and their criticality.

**Table 2: Perceived issues while creating and using the DDE-prototype and their criticality. Minor means the user can live with this error. Major means the issue has to be solved for a harmonized IMA development environment.**

<table>
<thead>
<tr>
<th>Minor</th>
<th>Major</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Page 6 of 11
Integration/Installation

Overall the integration of each tool into Eclipse succeeded. However, two of five tested ways integration showed to be infeasible for a common development environment.

- The installation by copying plug-ins from one Eclipse instance to another is error-prone since plug-in dependencies are not validated.
- External tools settings can not set programmatically being imported otherwise. In addition, this needs an separate installation of the tool, which prohibits a single installation of the IMA development environment.

Performance

Table 3 depicts the start-up performance of the DDE prototype. It shows that the DDE prototype needs almost twice the start-up time of the stand-alone Eclipse. The file size and the amount of plug-ins triples, however. In addition with the overall short start-up time this predicts a good scalability of the Eclipse platform.

<table>
<thead>
<tr>
<th></th>
<th>DDE Prototype</th>
<th>Eclipse 3.4.2 Classic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start-up time*</td>
<td>5.5 s</td>
<td>3 s</td>
</tr>
<tr>
<td>Size</td>
<td>917 MB</td>
<td>174 MB</td>
</tr>
<tr>
<td>Number of plug-ins</td>
<td>932</td>
<td>323</td>
</tr>
</tbody>
</table>

Table 3: Performance comparison of the integrated IMA development environment prototype and Eclipse Classic (*Intel Core 2 Duo @ 3Ghz, 3 Gb RAM, Windows XP 32-bit)

Stability

The prototype is overall stable. In case of incompatible tools or missing dependencies some functionality is not available, but owing to Eclipse’s exception handling the prototype never crashed.
General Usability/HMI

The flexibility of the Eclipse workbench allows to integrate nearly all functionality and UI concepts. However, each tool vendor has a slightly different usage philosophy. Especially the integration of former RCP applications breaks the concept of a harmonized HMI. Perceived effect range from the user does not know where to find the functionality he needs up to he is confused by persisting functionality, which does not make sense at the current stage of development. Major issues are:

- Misplaced actions: Actions are menu or toolbar entries. If those actions are defined globally, they appear always even if the actions can not be executed or are not needed in the current scope, e.g. resource, editor or view.

- Accelerators (key shortcuts) are likely to be registered multiple times by different actions. This stops the shortcut from working. In general such name clashes could happen with other plug-in properties, e.g. plug-in id, as well however this is unlikely to appear.

- Ambiguous naming for similar functionality causes confusion, e.g. one tool names the perspective for application development “application development” the other names it “Product XY”.

Versioning

All integrated Eclipse based tools depend on open-source extensions, other than the Eclipse base framework, e.g. both A653 development environments depend on the CDT. Depending mean that plug-ins of the tool depend on plug-ins of the extension. Despite the name of the required plug-in, its version is defined. Integrated tools happened to depend on the same open-source plug-ins, but in different versions. In general, Eclipse is able to handle multiple versions of the same plug-in at the same time. This means extensions requiring the same plug-in in different versions can co-exist within one Eclipse instance. However, this seems a fragile construct and resulted in incompatibilities that made it impossible to run tools, or functionality in the same Eclipse instance.

Licensing

All integrated COTS tools use a licensing mechanism to prevent unauthorized use. In a single Eclipse instance licensing caused two problems. First, two tools expected required the license file with the same name in the Eclipse root directory such that only tool can be started with a valid license at a time. Second, a missing license caused one of the tools to stop the whole Eclipse instance instead of disabling the related functionality. Latter prevents also the use of tools that have a valid license.

DISCUSSION

In general Eclipse showed-up to be flexible enough to integrate each type of tool. Even the integration of certified tools is no problem. Moreover, it can cover the requirements for model-based, concurrent development, as well as the different roles of the process. The good support for external applications allows the integration of Eclipse based extensions as well as stand-alone applications in several ways. The performance of Eclipse is good with many tools integrated, and the scalability is mainly limited by the hosting hardware. However, the shortcomings identified with the DDE prototype prohibit an of-the-shelf use of Eclipse and IMA tools. Above noted issues have to be considered on the framework and on the tool provider’s side. In the following are, therefore, suggestions to solve the major issues of Eclipse as a harmonized tool environment.

BASIC FRAMEWORK

A basic framework should be managed by a higher authority, e.g. the airframer. This authority should manage the version of Eclipse and provide a list of common open-source extension and their version. To ensure the openness of such a platform the original releases of Eclipse should be the basic framework. The release versions should be specified and updated from time to time. The authority should be notified on required foreign Eclipse extensions and insert them into the list. For practical reasons the authority could host an Eclipse update site with all specified plug-ins.

In addition, a basic set of plug-ins should be provided by the authority, which includes

- a mechanism to register installed tools, such that a central database is available on what is installed for which purpose. This could support the user of the development environment with a guided graph of tools and actions to carry out. A good example is the GMF Dashboard [14]. Each tool provider should be able to insert his tool into the dashboard.
• In addition, a process role managing plug-in should help automatic enabling and disabling of view, actions, and tools depending on the current users process role.

RULES FOR TOOL DEVELOPERS

Most important is that the tool developer does not change the Eclipse itself, i.e. no changes to the Eclipse core plug-ins, or the workbench are allowed, which are not provided by extension points. Even modifications with extension points that have global effect, e.g. a start-up dialog, should be used with care. For a harmonized HMI in general the Eclipse User Interface Guidelines [16] should be applied. In addition, the tool should implement Eclipse’s standard mechanisms for help, documentation or cheat sheets, if applicable. Names of views, perspectives and actions should primarily be named on their purpose and not on the product. Tools should always be in form of Eclipse plug-ins. Binary applications should be packed in Eclipse bundles, and should than be called from a wrapper plug-in. The way of software distribution should be an Eclipse update site, since this enables dependency validation and uninstalling. The update site could be online on a (password protected) websites or offline as an archive. The licensing mechanism of the tool should be conflict free, and a missing license should only disable related plug-ins and not the whole Eclipse instance. Following this rules the tool developer can still deliver the same tool as stand-alone Eclipse RCP application.

TOOL INTERCONNECTION

The overall amount of data in the IMA development process is big. A standard data format for all related data is not advisable. Moreover, this would make future changes difficult. Therefore, it is suggested to have domain-specific data in each tool. However, the data model should be defined formally as a meta-model in a standardized language. In order to give other tools the chance to retrieve or deliver data (parts of) the meta-model should be open.

CONCLUSION

The scope of IMA platform development is to model the platform, derive the configuration, generate the load, and finally simulate and validate the integration. This process is concurrent and comprises several process roles and areas of work. Each area of work contains domain-specific tools. Those tools are in-house developments, or COTS tools of different vendors. Eclipse can represent the process roles by certain perspectives and plug-in sets, and provides integration capabilities for all tool types. In addition, it provides strong modeling and model transformation capabilities, which can be used for model-based development and automated tool data exchange. All identified requirements could be mapped to Eclipse.

A prototype of an integrated IMA development environment is build from six selected tools, which cover stand-alone, Eclipse-based, COTS and certified tools. Integrated are two software development environments, model-based planning, a code generator, a configuration editor and a test suit. The prototype revealed most integration methods as unfeasible. In addition, it is revealed that the integration in Eclipse in not sufficient for a harmonized HMI, e.g. the ambiguous naming and different usage philosophies are problematic. Additional problems are noted in versioning and licensing. Nevertheless, the flexibility, performance, and stability of Eclipse is feasible for a common IMA development environment.

A harmonized IMA development environment is only possible if the identified issues are solved on environment maintainer and tool provider site. Therefore, it is suggested to specify and maintain a basic framework comprising Eclipse and commonly used open-source extensions. The usability should could be improved by a user-guiding dashboard. Tool developers should always use Eclipse bundle mechanism, and the update site based installation service. The should take care on general UI rules and conflict-free licensing. In general they should use Eclipse as framework for their application and not as a part of it.

OUTLOOK

Up to here the installation, the HMI, and the technical compatibility of the tools and Eclipse has been tested. The next step is to investigate formal standardized ways of data exchange between tools. This investigates model-transformation languages, and other concepts like the Speeds-Bus [7]. In addition, an IMA development related set of plug-ins is to be defined, which is the baseline for process roles management and user support like the proposed dashboard.
REFERENCES


ACKNOWLEDGMENTS

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under SCARLETT grant agreement n° ACP7-GA-2008-211439 (www.scarlettproject.eu).

DEFINITIONS/ABBREVIATIONS

CDO Connected Data Objects
COTS Commercial-Off-The-Shelf
DDE DME development environment
DME Distributed Modular Electronics
EMF Eclipse Modeling Framework
EPL Eclipse Public License
HMI Human-Machine Interface
IMA Integrated Modular
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avionics</td>
<td>Operating System</td>
</tr>
<tr>
<td>QVT</td>
<td>Query View Transformation</td>
</tr>
<tr>
<td>RCP</td>
<td>Rich Client Platform</td>
</tr>
<tr>
<td>SDE</td>
<td>Software Development Environment</td>
</tr>
</tbody>
</table>