**Proceedings Paper:**
Ali, Qurat ul ain, Kolovos, Dimitris orcid.org/0000-0002-1724-6563, Garcia-Dominguez, Antonio orcid.org/0000-0002-4744-9150 et al. (3 more authors) (2024) Advancing Domain-Specific High-Integrity Model-Based Tools:Insights and Future Pathways. In: Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems. MODELS '24 . Association for Computing Machinery, Inc , New York, NY, USA , 104–113.

https://doi.org/10.1145/3640310.3674094

# Advancing Domain-Specific High-Integrity Model-Based Tools: Insights and Future Pathways

Qurat ul ain Ali, Dimitris Kolovos, Antonio Garcia-Dominguez
{quratulain.ali,dimitris.kolovos,a.garcia-dominguez}@york.ac.uk
University of York
York, UK

Michael Bennett, Joe Newton,Piotr Zacharzewski
{mike.bennett,joe.newton,piotr.zacharzewski2}@rolls-royce.com
Rolls-Royce
Birmingham/Derby, UK

## ABSTRACT

Rolls-Royce Control Systems supplies engine control and monitoring systems for aviation applications, and is required to design, certify, and deliver these with the highest level of safety assurance. To allow Rolls-Royce to develop these systems, which continue to increase in complexity, model-based techniques are now a critical part of the software development process. At MODELS 2021 we presented early experiences with using and maintaining a bespoke domain-specific modelling workbench based on open-source modelling technologies, including the Eclipse Modelling Framework (EMF), Xtext, Sirius, and Epsilon. In this paper, we build on our previous paper with further insights, new challenges and lessons learnt as we have advanced and matured our domain-specific solution. We also discuss our experiences with moving towards web based modelling tools based on open-source technologies including Sirius Web, Eclipse GLSP and Eclipse Theia. Rolls-Royce intends to use a selection of these technologies to build a web-based modelling workbench, which will be used to architect and integrate the software for future Rolls-Royce engine control and monitoring systems in a collaborative way.

## CCS CONCEPTS

• **Software and its engineering → Object oriented development**.

## KEYWORDS

Domain specific languages, component oriented architecture, web based modelling, GLSP, EMF

**ACM Reference Format:**
Qurat ul ain Ali, Dimitris Kolovos,, Antonio Garcia-Dominguez and Michael Bennett, Joe Newton,Piotr Zacharzewski. 2024. Advancing Domain-Specific High-Integrity Model-Based Tools: Insights and Future Pathways. In *ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems (MODELS '24), September 22–27, 2024, Linz, Austria.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3640310.3674094


*MODELS '24, September 22–27, 2024, Linz, Austria*

ACM ISBN 979-8-4007-0504-5/24/09
https://doi.org/10.1145/3640310.3674094

## 1 INTRODUCTION

CaMCOA (Controls and Monitoring Component Oriented Architecture) is a reference software architecture designed to support future generations of Rolls-Royce's Controls and Monitoring systems. CaMCOA allows software components to be independently developed and integrated together on multiple computing nodes.

CaMCOA is designed to support aerospace safety-critical software development standards [3, 8]. This has been part of a large transformation initiative to modernise tooling and processes to adopt techniques prevalent in other industries and sectors. Because of the safety-critical nature of the products, the industry has been slow to adopt these, and has objectives that must be met to achieve approval from industry regulators.

To support the application of CaMCOA, a domain-specific modelling environment, named CaMCOA Studio [5], has been developed. CaMCOA Studio is an architecture, integration and modelling workbench for developing high-integrity control systems. It has been developed using various open-source modelling frameworks (including EMF, Epsilon, Sirius and Xtext) over the last seven years. It has a deployed user base of over 50 users within Rolls-Royce and its software supply chain. It is being used in two production engine products and has been used to model, generate, verify and integrate software to control development engines.

Figure 1 shows a high-level view of CaMCOA Studio and how it relates to the reference architecture. The CaMCOA stack is a combination of a service-oriented architecture (SOA) and a layered architecture, with a publisher/subscriber data interface between the services, named the Data-Distribution Layer (DDL). The application comprises services that predominantly consist of Simulink® components. The platform consists of Simulink® and manually-developed C device drivers and infrastructure services. The Simulink® drivers are known as the Node Abstraction Layer (NAL), and typically provide measurements (e.g. temperature or pressure) to the application, which are sourced from attached sensor hardware.

One of the primary purposes of CaMCOA Studio is to edit and view the NAL Deployment Specification Model (NDSM), which describes the requirements for the deployment of device drivers. The requirements describe the deployment of software components to realise the acquisition of signal from hardware. For example, the NSDM describes the deployment of a temperature component that

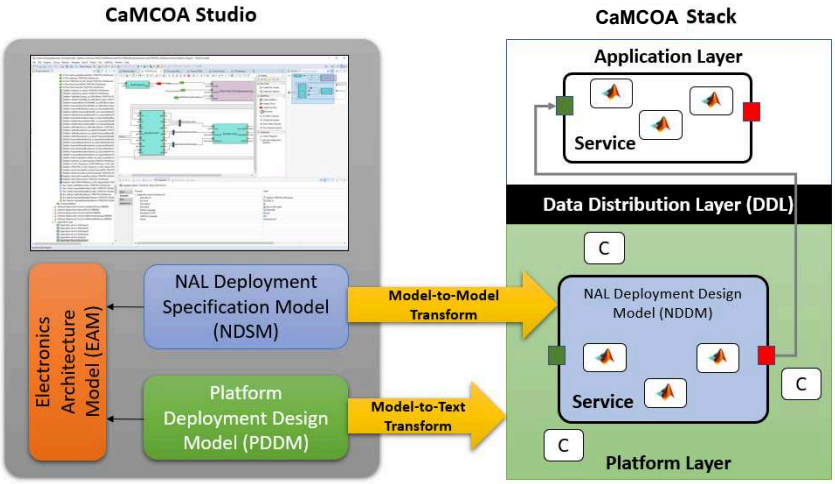


**Figure 1: CaMCOA Studio and Stack**

sources its inputs from an Analogue-to-Digital Converter (ADC) where voltages are scaled appropriately to provide an engineering unit value to the application. Prior to the development of CaMCOA this information was held in textual requirements.

The NAL Deployment Design Model (NDDM) is a refinement of the NDSM realised in Simulink® that includes architectural details such as dataflows. Because of the information provided in the NDSM the development of the NDDM is achieved using an automated model-to-model transformation using the Epsilon Transformation Language (ETL) [12].

The Platform Layer consists of these NAL services and infrastructure code conforming to MISRA-C 2012/C99 that is directly generated by a model-to-text transformation from the Platform Deployment Design Model (PDDM). To enable modelling of the software, CaMCOA Studio also supports the definition of the Electronics Architecture Model (EAM) that includes the processing nodes and hardware interfaces.

The development teams are now working towards the certification of engine software developed using CaMCOA Studio and this has driven development to support verification, requirements and traceability objectives required by aerospace standards, as later discussed in Section 2.1. This has resulted in a need to support frequent and complex model and meta-model refactors.

As CaMCOA is becoming the standard way to create Engine Control Systems on all Rolls-Royce projects, globally, there is a desire to simplify deployment and collaborative development. A number of technologies to enable this are being investigated, including making the tooling web-based, and allowing multiple users to edit the same model at the same time. In this paper we describe our experience of using domain-specific model-based technologies in CaMCOA Studio.

The remainder of this paper discusses the experience of maturing the product, particularly focusing on the challenge of meeting aerospace standards. It reports back on challenges listed in our previous paper, lists further lessons learnt, and then presents current work on adopting web-based modelling technology to facilitate wider adoption within the business.

## 2 EXPERIENCES

### 2.1 Standards Compliance

As our production engine control systems mature towards delivery using CaMCOA, the focus of improving CaMCOA has moved towards achieving standards compliance. The aerospace industry has developed a specific set of guidelines [8] for Model-Based Development. These guidelines define a number of different objectives for *Specification Models* and *Design Models*. The characteristics of the different models in CaMCOA are shown in Table 1. The EAM is out-of-scope of the software guidance, as it is part of the electronics process (and this is considered future work), but the software models must be shown to be conformant to standards which have been developed by Rolls-Royce to describe the structure and content of valid models. The specification model contains High-Level Requirements (HLRs) that describe required behaviour without prescribing an implementation, whereas the Design Model contains low-level Requirements (LLRs) that describe detailed behaviour. The Design Model may contain architecture details, whereas the Specification Model will generally not.

The availability and wider industrial use of these guidelines has meant that there is a lower risk to the use of modelling techniques, because the regulator has standard ways of assessing applicant software. Tools such as Matlab Simulink® are now prevalent for functional modelling and code generation.

*2.1.1 Model and Metamodel Evolution.* In previous releases of CaMCOA Studio, the information in the NSDM, PDDM and EAM was combined in a single persisted XMI file. This caused a number of

| | Specification Model (NDSM) | Design Model (PDDM, NDDM) | Electronics Architecture Model (EAM) |
|---|---|---|---|
| Compliance to ED-218 required | Yes | Yes | No |
| Modelling standards required | Yes | Yes | No |
| Requirements | High-Level Requirements (HLRs) | Low-Level Requirements (LLRs) | Hardware Design Requirements (HDRs) |
| Requirements compliant and traceable to | System Requirements Allocated to Software (SRATS) | High-Level Requirements (HLRs) | N/A |
| Contains architectural details | No | Yes (Software) | Yes (Electronics) |
| Code generated from model | No | Yes | No |

**Table 1: CaMCOA Model Types**

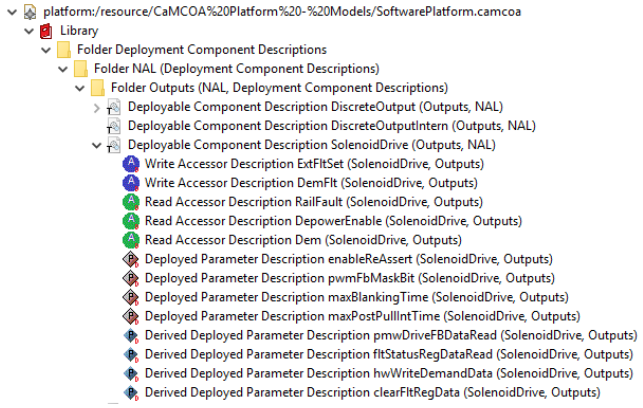


**Figure 2: Example of Library Model Information**

| | NDSM | PDDM | EAM |
|---|---|---|---|
| Deployment model files | 1 | 1 + 1 per service | 1 |
| Library model files | 0 | 1 | 1 |

**Table 2: Decomposition of CaMCOA models into multiple files**

problems, including being able to show compliance with the regulatory guidance; one problem being the ability to review these in isolation for compliance to objectives. This is discussed in detail in the beginning of this section and the split ensures that the review and analysis verification activities of each model are fully self-contained, and changes to one model do not undermine evidence already gathered for a model that has not changed. However, cross-references between the models and some verification activities must take into account these dependencies. As an example, the PDDM references the EAM to account for hardware constraints, but review and automated checking of the PDDM to show conformance to standards can be done without considering whether the EAM is correct. Regardless, testing of the resulting software image must consider whether the EAM is also correct as they are Hardware-Software Integration tests (HSI) that test overall system operation.

There were additional challenges as the team has grown to support the transition from a research prototype into a product. Separate teams now maintain the NDSM+NDDM, PDDM and EAM. Having an a single all-encompassing model increased the chance of merge conflicts and difficulty merging.

The single all-encompassing model has been evolved into 3 separate logical models (NSDM, PDDM, EAM), which have been further split into separately persisted model files as shown in Table 2. A set of reusable *Library* assets that can be reused in separate projects has been developed. These reflect the fact CaMCOA is developing a product line of reusable components to deploy. The organisation is split into a core team developing the library assets and multiple project teams working on specific products. All these teams undertake CaMCOA and Simulink modelling activities.An example of the information in the library model is shown in Figure 2. This shows the definition of interface information for a "Solenoid" component, including its inputs, outputs and parameters. This is architectural information used to connect the system components together.

The model restructure has gone hand-in-hand with a reorganisation of the metamodel, which has evolved from a single Ecore EPackage to eleven EPackages that reflect the different areas modelled (e.g. Requirements, NAL, Software, Electronics). This has improved maintainability and made it easier to support concurrent evolution of the models and metamodels, as they are now aligned with team boundaries. Model migration has been a constant challenge, and this is discussed further in Section 3.3.

*2.1.2 Requirements Capture Tool.* Rolls-Royce have experience that shows Commercial-Off-The-Shelf (COTS) tooling requires significant customisation to integrate with other tools. This can be a similar cost to developing a bespoke solution. In line with the rest of CaMCOA Studio, an in-house Requirements Capture Tool (RCT) has been developed to allow engineers to define requirements for manually-coded (i.e. non-Simulink) CaMCOA components. This included a new metamodel defining documents, information, requirements and traceability links with integrated version management. This is integrated into CaMCOA Studio, and allows the engineers to work on models, requirements and code in one integrated environment. Even within manually-coded components, Rolls-Royce seeks to minimise requirements specified in natural language (which can be very ambiguous), and instead use semi-formal descriptions such as state-charts, flow-charts, truth tables and equations to express behaviour. To that end, RCT allows the user to capture input in textual-based syntaxes (including PlantUML, Mermaid and MathJax). This rich-text approach avoids the need to explicitly configure

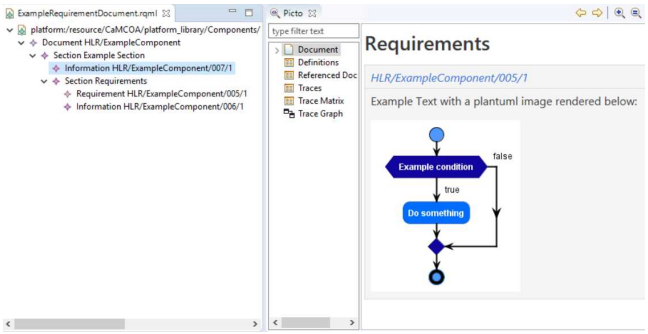


**Figure 3: RCT Document Example with Picto Rendering**

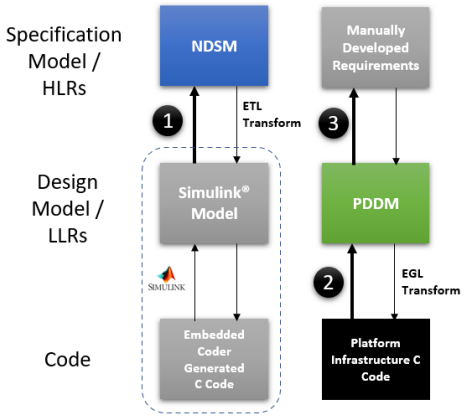


**Figure 4: Model to Model and Code to Model Traceability and Compliance**

bitmap images that are not easily editable, improving maintainability.

RCT models are serialised in YAML (Yet Another Markup Language) format rather than XMI (for readability), and transformed into Markdown and HTML-rendered artifacts using EGL [17]. With the information defined in a model, it is possible to perform model validation in EVL [13] and rendering in Picto [11], and have custom GUI-based support such as Eclipse-based menu items and navigation tools. Currently, the RCT is packaged as a set of plugins within CaMCOA Studio.

*2.1.3 Traceability and Compliance.* As discussed briefly in Section 2.1, an important certification objective is to establish traceability and compliance between lower-level and higher-level artefacts, to ensure all that is required is present and nothing that is not required is present. Informally, the traceability objective ensures that everything in a lower-level artefact (either code or a design model) is linked to a higher-level one (either a design model or a specification model), and compliance necessitates that everything in the lower-level artefact meets the intention of the higher-level

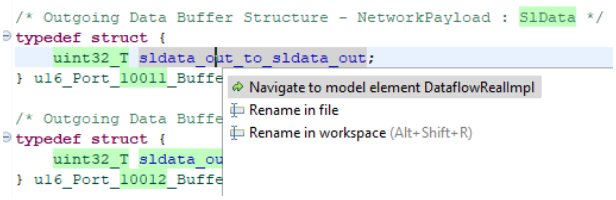


**Figure 5: Deployment Code Analyser Example GUI-based Output**

requirements. It is possible to have complete traceability (everything is linked) but it could be incorrectly linked and thus not have correct compliance.

Figure 4 shows the forward path transformations from high-level artefact to low-level, which imply appropriate traceability and compliance checks that must be made: ❶ Simulink NAL model to the NDSM, ❷ Platform infrastructure code to the PDDM, and ❸ PDDM to manually developed HLRs

These objectives can be checked by manual review or using an appropriately qualified tool. For example, Simulink provides qualified tool support, so is not considered further in this paper. A review can be challenging if appropriate tool support is not available. In particular, reviewing code against a model from which the code was generated can be difficult, due to the size and complexity of the models and the transformations that consume them. This has led to the development of a number of tools that make traceability information more visible to engineers, to speed up compliance preparation. The aim is eventually to provide qualified tools that eliminate manual code review.

*Deployment Code Analyser.* The Deployment Code Analyser (DCA) was designed to meet help meet objectives MB.A-5.1, MB.A-5.2, and MB.A-5.3 from RTCA DO-331/ED-218 [8]. These are respectively to show that:

(1) The source code is accurate and complete with respect to the LLRs.
(2) The source code matches the data flows and control flows defined in the software architecture.
(3) The source code does not trace to model elements that do not represent low-level requirements.

The initial experience of developing a code generator showed that developers were putting excessive functional and algorithmic code in the code generation templates. To enforce the separation of concerns between the architectural modelling of CaMCOA Studio and the detailed descriptions of behaviour done in C code or Simulink models, the architectural models and the code generators have been kept free of complex algorithmic code, and a separate process (the DCA) checks the compliance between the CaMCOA architectural models and the C code. This separation of concerns simplifies certification, as it decouples checking that the architectural descriptions match the architecture (via model/code matching), from checking that the functional descriptions in C/Simulink meet the functional requirements (e.g. via testing). In order to provide feedback to the users, DCA uses trace information emitted by the

EGL templates during the model-to-text transformation: by parsing the Abstract Syntax Tree (AST) of the source code using the Eclipse CDT (C Development Tools) Framework, it is able to establish matches between model elements and emitted code. The tool is able to ignore syntactic and irrelevant features of the C language. Deployed early in the development lifecycle, the tool helps the developers to refactor the models and code to be compliant. The non-compliant code can be removed by either adding additional model information, or by moving this into component code, where compliance and traceability are checked in relation to component requirements (which are captured with the RCT, as described in Section 2.1.2).

The DCA was initially designed as an interactive tool to highlight code sections within the Graphical User Interface (GUI). Users can navigate to the model element(s) used to generate a particular section of highlighted code (see Figure 5). Later updates included a batch implementation of the DCA on a set of generated code files, which produces a metric to indicate what percentage of the generated deployment code can be directly traced to a model element. This tool is packaged as part of CaMCOA Studio, and also runs in a Continuous Integration (CI) environment. At present the tool is a designed to be a review aid, but in future the desire is to eliminate the manual review.

*NAL Deployment Design Model Analyser.* The NAL Deployment Design Model (NDDM) Analyser is a prototype tool designed to provide similar functionality to the DCA. A major challenge is interfacing Simulink® with the Eclipse Modelling Framework. It currently uses Simulink® APIs to support interactive navigation from a generated Simulink® model to the CaMCOA model element used to generate it. This was achieved by starting a server upon launch of CaMCOA Studio, to receive Simulink model information from Simulink APIs needed for editor navigation in CaMCOA Studio, which could be triggered with Simulink requirement links generated using Epsilon Transformation Language (ETL) [12] scripts. This was developed with audits in mind, where there is a need to demonstrate compliance and traceability in a live environment. It improves the time taken to perform software reviews by providing a faster and clearer route from generated Simulink models to the corresponding CaMCOA models. This tool will eventually acquire a traceability/compliance metric similar to the DCA. However, it was recognised that reviewing the Simulink model against the CaMCOA specification is an easier job than reviewing code against the CaMCOA design model due to the there being much more detail in the design model and code. Therefore, the development of the DCA has taken priority.

## 2.2 User Experience

With a growing number of customers using CaMCOA Studio on engine production projects, the need to mature the usability of the product became a priority. Key updates included custom Eclipse-based GUI menu items and headless products to automate activities, along with improving our Continuous Integration (CI) pipelines that automate regression testing and analysis tooling (such as schedulability analysis and code conformance checks). Additionally, improvements have been regularly made to the Sirius diagrams, which provide a more user friendly approach to navigating and editing

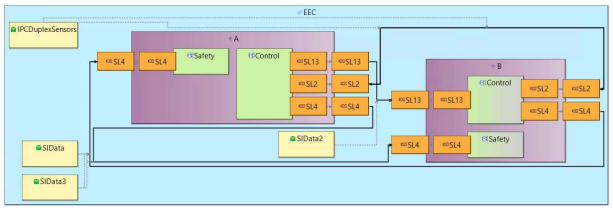


**Figure 6: Improved Sirius Layout**

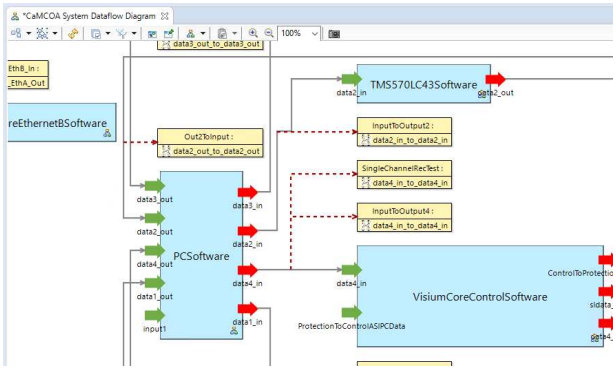


**Figure 7: Improved Dataflow Diagram**

proposed deployment designs. A key example included reformatting our diagrams to support the auto-layout functionality provided by the Eclipse Layout Kernel (ELK) [1] project. Many of our diagrams contained nested containers, and when element-based edges were connected between two elements over a container boundary, the auto-layout functionality did not work well. Therefore, we developed the diagrams to minimise the occurrences of this, moving the elements used to connect element-based edges to border nodes of top-level containers, as shown in Figure 6.

Another key refactor was to the dataflow diagrams and model. These diagrams connect service inputs and outputs. In addition to the layout problem, these suffered from being too large and complex. The Sirius representations file that accompanied the semantic model regularly became broken. The following improvements were implemented:

- The ability to hide elements in the diagram using a filter (e.g. stubbed dataflows and messages)
- Split the dataflow diagrams into three separate diagrams: (1) communication that is local to a computing node, (2) channel dataflows that are between channels (our system has dual-redundant channels running identical software for safety purposes) and (3) system dataflows that are between computing nodes.

This made the system easier to maintain, to review and to edit. An example of the improved system dataflow diagram is shown in Figure 7. This work required a large migration script: the challenge of migration is discussed in Section 3.3.

## 3 CHALLENGES

The overall challenge for Rolls-Royce has been to support, maintain and grow CaMCOA since the previous paper whilst it is used on live projects. This is due to the specialist knowledge required, and the variety and relative complexity of the underpinning frameworks compared to traditional Rolls-Royce software tools. Retention and growth of specialist modelling knowledge within the business is a key challenge.

There is ongoing tension between the need to add new features and refactor for maintainability, whilst the product is in active use. CaMCOA is continuously deployed via release pipelines and regressions are found quickly by the customers. Improving performance (see section 3.1) of validation, generation and interactive use continues to be an area of focus. Currently, refactoring is slow due to high-coupling between areas (for example functionality is sub-optimally distributed across multiple plugins, one case being the GUI handling), and difficulties in performing model migration (see section 3.3). This is compounded by modelling challenges, due to the needs of the aerospace guidance not being fully considered as the product evolved from a research prototype.

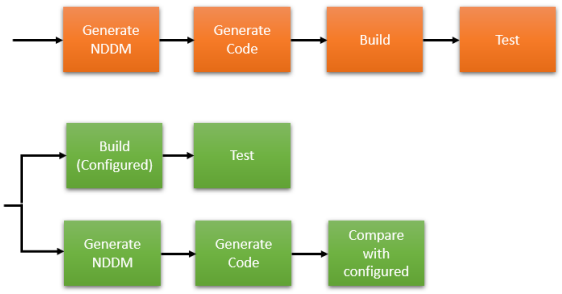


**Figure 8: CaMCOA Pipeline Optimisation**

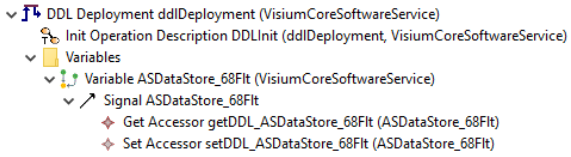


**Figure 9: CaMCOA Derived Element Example**

### 3.1 Performance

Performance has consistently been an area of concern. The tool is used interactively and headlessly via CI, and is part of pipelines that can take several hours. CI performance impacts the speed of the feedback loop for engineers as tests must pass before work is completed. Interactively, work is also impacted by processes that take multiple minutes, disrupting concentration.

Transforming the NDSM to the Simulink® NDDM can take tens of minutes, due to the overhead of the underlying Simulink® Java interface. This has been partially addressed by parallelising the generation of independent parts of the model and also by making more use of native Matlab scripting. One option to improve this is to generate an intermediate representation using a model-to-text transformation, producing Matlab scripts, such that use of the Java interface is minimised. This has been employed in specific areas, leading to significant improvements when manipulating Data Dictionaries (improvements of 90% have been seen in these areas). Another approach is to refactor the Epsilon scripts to minimise costly model operations. For example, it has been found that re-parenting Simulink® objects is particularly expensive, requiring deep-copying of objects. This is believed to be due to a limitation of the Simulink® interface and has not yet been addressed. If addressed, it is believed that model generation times could drop significantly (estimated 50%).

Model validation using Epsilon Validation Language (EVL) and model-to-text transformations for direct code generation used to take several minutes to execute. As these delays would frustrate users, several techniques have been employed to improve this. Firstly, the activities have been moved to run in background threads, so the the user can continue to work in an interactive GUI environment. Additionally, the built-in Epsilon profiler has been used to locate performance bottlenecks. It was found that one particular recursive function was performing a model search on every invocation. This was contributing over 90% of the execution time. Validation and code generation times have been reduced from several minutes to around one minute at the time of writing.

In the CI environment, the pipelines have been optimised as shown in Figure 8. In our previous approach, shown in orange, only the CaMCOA NSDM model was configured and thus to perform testing the Simulink® models (the NDDM) and then the source code has to be generated before build and test. By configuring the source code and parallelising the generation with the test stages the speed of the overall end-to-end CI job could be reduced by more that 30%. The comparison ensures that what is configured matches what is under configuration control.

### 3.2 Derived Transient Model Features

The CaMCOA models have started to make use of derived (calculated at run-time) and transient (not persisted to XMI) elements in addition to derived elements and attributes. The use of derived features of EMF is generally to make the model easier to navigate, but it also fulfils an important purpose for certification to help remove detail from the code generator and more information in the PDDM. There are some features of PDDM (variables to hold data data in the DDL are one example) that we do not want to let users manually create. They can be inferred from the dataflow model. However, we need to be able to be able to show traceability and compliance to the model. Therefore we need these to be shown in the model. They need to be elements as they often have child elements and complex attributes. An example is shown in Figure 9.

The downside of this approach is that derived element handling is complex and relies on correct handling of model events. Pessimistic re-creation of derived elements can slow down model activities significantly. There is an open question whether this is the best approach. Other approaches may be to use validation and quick fixes to allow a user to create real, persisted elements, provide an intermediate model-to-model transformation or investigate the incremental derivation approaches in [10, 14].

### 3.3 Model Migration

Our metamodels have continued to evolve at a fast pace, as we add support for new components and refactor the models for maintainability and aerospace compliance purposes. The metamodel has evolved 43 times since a migration capability was introduced in 2021. Prior to this point migrations were handled manually. This experience contrasts sharply with Simulink® where 2 different versions of the tool have been used in the same period.

When metamodel changes are introduced, an Epsilon Flock [16] migration script is written to ensure customers are able to migrate the models correctly. The features of CaMCOA Studio which rely on the model structure to retain functionality must also be updated including updating all test models, the Sirius description and the code/model generators. Fundamental metamodel refactors such as the NDSM refactor which separated the NDSM from the PDDM, remained in progress for a long time which resulted in repeated merging cycles. Some of the issues experienced included:

(1) The scale of some changes proved challenging to implement without errors, often causing regressions (eg. broken Sirius diagrams) because there were limited GUI CI tests.
(2) Migrations would sometimes succeed for a single increment but fail when run as part of a multi-stage migration. This was often due to calling generated EMF Java code from the Flock scripts. This changes as the metamodel evolves and hence old migration scripts would break.
(3) Incorrect archive versions of the metamodels would be configured as migrations were designed on long-lived feature branches. As the feature branch progressed other non-breaking metamodel changes were introduced. This caused certain migrations to break if the customer used a metamodel feature that was not in the archive.
(4) References to semantic model elements from the Sirius representations model are not always correctly migrated. This leads to invalid cross references and CaMCOA Studio being slow to open the Sirius project.

To handle these problems, new CI tests were introduced to ensure a new migration did not break old ones (see Figure 10 for an example of the CI running a multi-stage migration test) and ensure that metamodel versions were correctly archived. Guidance has also been introduced and there are now around five engineers trained in the development of migration scripts.

One of the key improvements made was to provide guidance that during a migration, manual changes to the model should be avoided. Because migration scripts are demanding to write, engineers were leaving this until last. This meant that during feature development, time was wasted merging in other model changes manually into the feature branch. Instead the user should develop the Flock script early, meaning that other changes to the model can be merged and then the migration can be run without any manual model merging.

## 4 LESSONS LEARNT

A summary of the lessons drawn from the process of developing and using CaMCOA Studio within the business are enumerated below:

- It has been found that it is possible to build and deploy a complex domain-specific modelling capability for engine control systems within the business, building on mature Eclipse/EMF-based open-source frameworks.
- The benefits of developing CaMCOA Studio and its associated tools (eg. RCT) have compared favourably with adopting COTS tools and customising them to work well together.
- Sirius has proven to be a feature-rich and robust graphical editor development framework and has been able to support many different diagram types within CaMCOA Studio. ELK adds sophisticated auto-layout capabilities to Sirius which are essential for managing large diagrams.
- It has also been possible to develop a strategy for certification and compliance, however this has required significant refactoring and further work is required to achieve compliance with aerospace guidance developed by industry experts and accepted by certification authorities.
- Traceability between models and code is essential for review and certification activities. Using dedicated languages/frameworks for model-to-text transformation that record traceability/provenance information instead of string concatenation is therefore recommended.
- Care must be taken to develop a model structure that supports traceability and compliance objectives and an approach to adding more detail to a model is required (e.g. derived elements) to avoid introducing non-compliant complexity into the code generator.
- Splitting models and metamodels across multiple files is helpful to reduce conflicts in a collaborative development environment. It also helps align models to team roles.
- Most of the performance issues encountered have been due to inefficient coding (e.g. unnecessary repeated model-wide searches) and as a result of crossing tool boundaries (e.g. interacting with the Simulink® API), and rarely due to fundamental inefficiencies of the underpinning modelling and model management frameworks of CaMCOA Studio. Profiling tools are essential for identifying performance issues in complex model management programs (e.g. model-to-text transformations).
- The cost of metamodel evolution sharply increases with the number/complexity of downstream tools (e.g. graphical syntaxes, transformations) that depend on it.
- Without seamless and low-overhead model migration it would not have been possible to rapidly deploy CaMCOA Studio to a set of live projects.
- While tools such as Flock can be used to automate model migration activities, diagram migration remains an open challenge.
- Being able to run model management programs in a headless mode (without a GUI) is essential for continuous integration activities.

An overall lesson is CaMCOA Studio development has required skills that Rolls-Royce traditionally doesn't have. Training, retaining and recruiting skilled engineers needs continual focus. The challenge of maintaining and expanding the knowledge base to meet the growing demands from the business for model-based tooling and CaMCOA is likely to be an ongoing challenge.

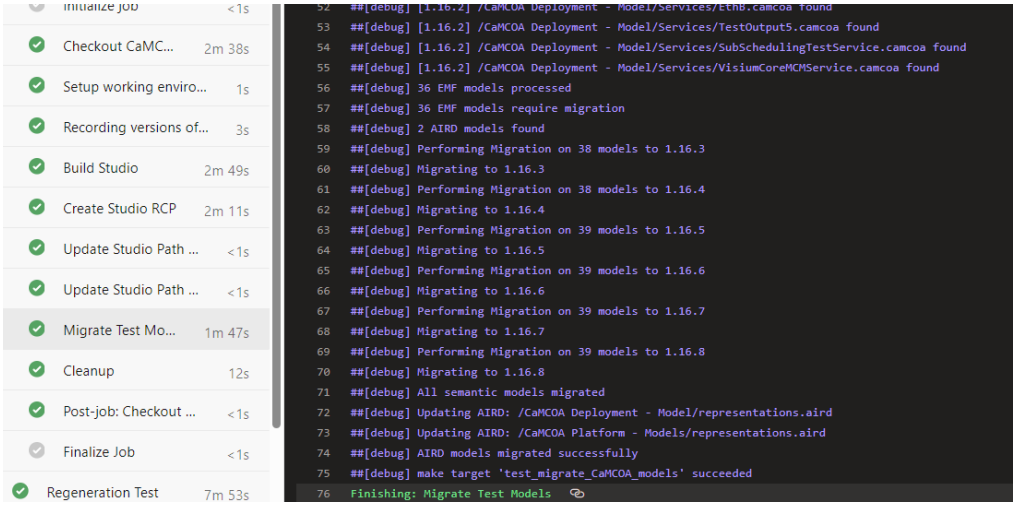


**Figure 10: CaMCOA CI Metamodel Migration Check**

## 5 MOVING TO THE WEB

There is a growing appetite to shift from desktop-based to web-based development environments. Platforms such as Eclipse Theia [7] and technologies like EMF.cloud [6] and Sirius Web [9] enable more than text-focused IDEs: they make it possible to build complex modelling tools based on modern web technologies. Rolls-Royce aspires to move to a cloud/web-based environment (CaMCOA Cloud) for several reasons, including:

- Enabling engineers to collaborate and work on the same model at the same time using a shared workspace.
- Avoiding complex and brittle installation and upgrading processes on individual engineer workstations, making it friction-less for new and ephemeral users.
- Migrating computationally expensive processes to a powerful and scalable back-end, and sharing physical resources. An engineer would still be able to launch computationally-heavy simulations and analyses, even when their local machine does not have enough computational power.
- Modernising the look-and-feel of the application by using contemporary web technology stacks.

### 5.1 Sirius Web

The first technology we explored was Sirius Web[1], which is an open-source web-based graphical editor development framework from the developers of the Sirius framework (Obeo), that we already use in CaMCOA Studio. At the time, Sirius Web appeared to be more geared towards standalone web applications and there was little documentation on integrating Sirius Web editors with web-based IDEs such as Eclipse Theia. Also, all the Sirius Web examples we could find made use of a database to store models, while our intention was to keep storing models as files so that they could continue to be version-controlled in Git repositories.

[1]https://eclipse.dev/sirius/sirius-web.html

### 5.2 Eclipse GLSP

Eclipse GLSP (Graphical Language Server Protocol) is an open-source framework for developing graphical model editors provided by EclipseSource [4]. Eclipse GLSP adapts the LSP client/server architecture to diagram editing, implementing a similar protocol, and providing reusable components for the development of custom web-based diagram editors. Client implementation is decoupled from diagram logic implementation, enabling the integration of clients in any technology. GLSP encapsulates all the language know-how on the server side, which helps reuse existing frameworks (e.g. EMF) and diagram editors. It also enables the management of large models by keeping them on the server, and not loading their entire contents on the browser. The front-end only focuses on rendering and user interaction.

GLSP provides strong integration with Eclipse Theia and examples of developing editors that operate on file-based models, and therefore it has been tentatively selected as the technology of choice for CaMCOA Cloud.

## 6 CAMCOA CLOUD

The envisioned architecture for CaMCOA Cloud is illustrated in Figure 11. The main features of CaMCOA Cloud are expected to include:

### Real-Time Collaborative Model Development

In the absence of robust tools for diagram comparison and merging, it is challenging for multiple engineers to work on the same CaMCOA models at the same time using file-based version control. To overcome this challenge, in CaMCOA Cloud we plan to investigate real-time collaboration options, building on the model change notification facilities of GLSP. Expanding the tool's support for collaboration is seen as an enabler for Rolls-Royce's exploitation of new and adjacent markets, such as UAM (Urban Air Mobility) vehicle design, where the business will need to collaborate with third parties in order to truly exploit this market potential.

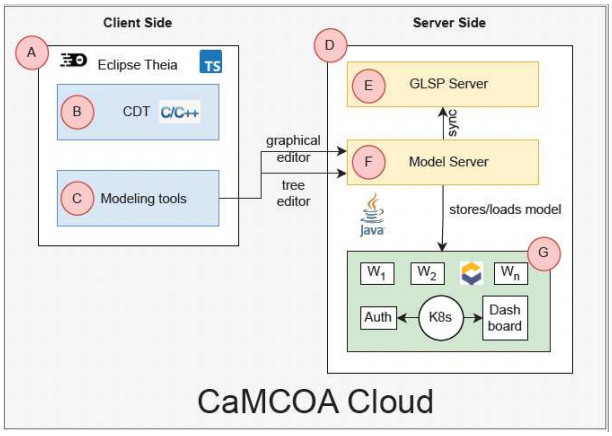


**Figure 11: Architecture of CaMCOA Cloud**

## Multi-domain Modelling

In order to model complete control systems, CaMCOA Studio needs to ensure that its digital platform modelling capability is (a) generic enough to support those domains (for example some safety-critical systems have triple-redundant capabilities rather than dual-lane) and (b) remove any concepts that are dedicated to engine control and monitoring. The short-term focus has inevitably meant that modelling of our engine control systems has been the key focus.

Another key enabler will be better integration with systems modelling tools such (for example Capella [15]) to allow a more seamless transition between the systems and software modelling process. As an example, currently some physical modelling is duplicated in the systems and software processes. A common metamodel or transformation would reduce or eliminate this duplication.

Wider multi-domain modelling support will support the ambition to apply and market CaMCOA to wider internal projects and external partnerships.

## Cloud-based Integration

This is a key requirement for transforming CaMCOA Studio into a "Software as a Service" solution. CaMCOA Studio is a fully desktop-based solution, requiring local installation and making maintainability (updates to latest versions) and portability difficult. Cloud-Based integration represents a key enabler for supporting wider deployment of CaMCOA Studio within the business and with potential partners.

### 6.1 Feasibility Study

To evaluate the maturity and feature-completness of web-based technologies that could support CaMCOA Cloud, we have developed a graphical model editor using GLSP and Eclipse Theia using a subset of the CaMCOA DSL for electronics modelling, which is shown in Figure 12. Our observations from the GLSP ecosystem so far are as follows:

- GLSP is very flexible to extend and integrate with other web-based technologies.
- At the same time it is a complex and fast-evolving framework with limited documentation and few up-to-date examples.
- GLSP is very code-heavy compared to Sirius.
- Tools and facilities that are provided out of the box in Eclipse/EMF/Sirius (e.g. tree-based editor, property view, problems view) need to be implemented almost from scratch in Theia/GLSP.
- To persist models in XMI (for compatibility with the existing CaMCOA Studio), we need to use a Java-based backend and a Typescript-based front-end. We have found debugging across two different languages to be significantly more challenging than debugging a Java-based solution within Eclipse.
- Frequent updates to GLSP cause code breakage and regressions. This issue has also been reported by the developers of OpenBPMN [2], which is one of the most prominent adopters of GLSP.

As a consequence, in parallel to our continued exploration of GLSP, we are also monitoring the development progress of Sirius Web and keeping it under consideration for future feasibility studies.

## 7 CONCLUSIONS

This paper has reported our experiences with developing an advanced domain-specific model-based development workbench using open-source Eclipse-based modelling technologies at Rolls-Royce, and the lessons we have learnt on the way. We have also outlined our future plans for moving our modelling workbench to the cloud and the relevant opportunities and challenges we have identified.

## ACKNOWLEDGEMENTS

This research was supported by a Knowledge Transfer Partnership co-funded by InnovateUK & Rolls-Royce plc. and by the SCHEME InnovateUK project (#10065634).

## REFERENCES

[1] [n. d.]. *Eclipse Layout Kernel.* Retrieved Mar 28, 2024 from https://eclipse.dev/elk/
[2] [n. d.]. *OpenBPMN.* Retrieved Feb 23, 2024 from https://www.open-bpmn.org/index.html
[3] RTCA (Firm). SC 167. 2012. *Software considerations in airborne systems and equipment certification : DO178C / ED-12C.* RTCA, Incorporated.
[4] Dominik Bork, Philip Langer, and Tobias Ortmayr. 2024. A Vision for Flexible GLSP-Based Web Modeling Tools. In *The Practice of Enterprise Modeling*, João Paulo A. Almeida, Monika Kaczmarek-Heß, Agnes Koschmider, and Henderik A. Proper (Eds.). Springer Nature Switzerland, Cham, 109–124.
[5] Justin Cooper, Alfonso De la Vega, Richard Paige, Dimitris Kolovos, Michael Bennett, Caroline Brown, Beatriz Sanchez Piña, and Horacio Hoyos Rodriguez. 2021. Model-Based Development of Engine Control Systems: Experiences and Lessons Learnt. In *2021 ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS).* 308–319. https://doi.org/10.1109/MODELS50736.2021.00038
[6] Eclipse Foundation. [n. d.]. *EMF Cloud.* Retrieved Feb 23, 2024 from https://eclipse.dev/emfcloud/
[7] Eclipse Foundation. 2024. *Eclipse Theia.* Retrieved Feb 23, 2024 from https://theia-ide.org
[8] RTCA (Radio Technical Commission for Aeronautics). 2011. DO-331 Model-Based Development and Verification Supplement to DO-178C and DO-278A. (2011).
[9] Eclipse Foundation. 2024. *Sirius Web.* Retrieved Feb 23, 2024 from https://eclipse.dev/sirius/sirius-web.html
[10] Daco C Harkes, Danny M Groenewegen, and Eelco Visser. 2016. IceDust: Incremental and eventual computation of derived values in persistent object graphs. In *30th European Conference on Object-Oriented Programming (ECOOP 2016).* Schloss-Dagstuhl-Leibniz Zentrum für Informatik.
[11] Dimitris Kolovos, Alfonso de la Vega, and Justin Cooper. 2020. Efficient generation of graphical model views via lazy model-to-text transformation. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems* (Virtual Event, Canada) *(MODELS '20).*

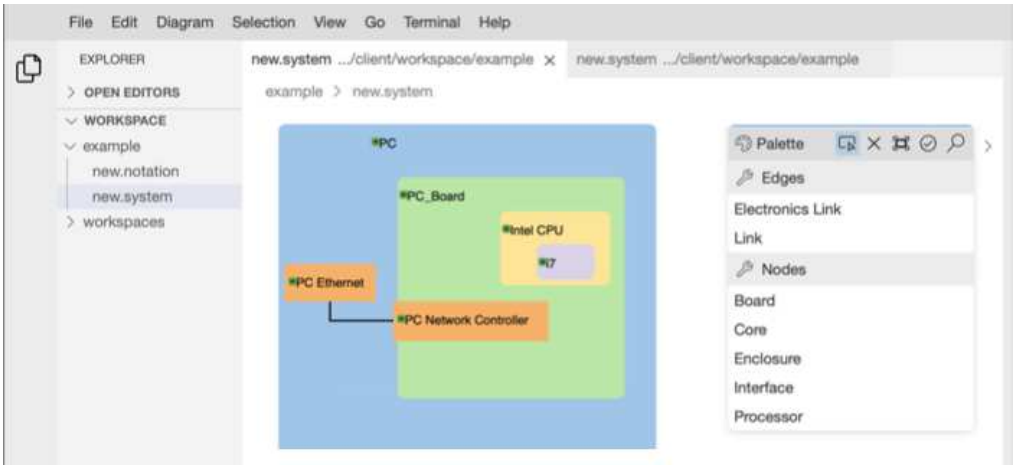


**Figure 12: Screenshot of the Electronics DSL graphical editor in Eclipse Theia**

Association for Computing Machinery, New York, NY, USA, 12–23. https://doi.org/10.1145/3365438.3410943

[12] Dimitrios S Kolovos, Richard F Paige, and Fiona AC Polack. 2006. Eclipse development tools for epsilon. In *Eclipse summit Europe, eclipse modeling symposium*, Vol. 20062. 200.

[13] Dimitrios S Kolovos, Richard F Paige, and Fiona AC Polack. 2009. On the evolution of OCL for capturing structural constraints in modelling languages. *Rigorous Methods for Software Construction and Analysis: Essays Dedicated to Egon Börger on the Occasion of His 60th Birthday* (2009), 204–218.

[14] István Ráth, Ábel Hegedüs, and Dániel Varró. 2012. Derived features for EMF by integrating advanced model queries. In *Modelling Foundations and Applications: 8th European Conference, ECMFA 2012, Kgs. Lyngby, Denmark, July 2-5, 2012. Proceedings 8*. Springer, 102–117.

[15] Pascal Roques. 2017. *Systems architecture modeling with the Arcadia method: a practical guide to Capella.* Elsevier.

[16] Louis M Rose, Dimitrios S Kolovos, Richard F Paige, and Fiona AC Polack. 2010. Model migration with epsilon flock. In *Theory and Practice of Model Transformations: Third International Conference, ICMT 2010, Malaga, Spain, June 28-July 2, 2010. Proceedings 3*. Springer, 184–198.

[17] Louis M. Rose, Richard F. Paige, Dimitrios S. Kolovos, and Fiona A. C. Polack. 2008. The Epsilon Generation Language. In *Model Driven Architecture – Foundations and Applications*, Ina Schieferdecker and Alan Hartman (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1–16.