Rational Rose Real-Time Migration to DevOps Model RealTime

-

Introduction

Author: Anders Ek

HCL

Last updated August 9, 2018 for DevOps Model RealTime 10.2.

Contents

1 Introduction

DevOps Model RealTime is a product made for development of soft real-time applications. It is the next generation of the Rational Rose Real-Time (RoseRT) tool. Model RealTime takes an evolutionary leap from the RoseRT technology base because of two main reasons:

- Model RealTime is built on the Eclipse Integrated Development Environment (IDE) platform (www.eclipse.org), and
- Model RealTime is based on the UML2 specification (https://www.omg.org/spec/UML/)

From a practical point of view the Eclipse integration has significant benefits in terms of functionality, usability and customizability. One example is the tight integration with the C++ development environment CDT that provides a full-fledged development environment for C/C++. Model RealTime uses CDT for all code level aspects of developing a real-time application. In addition Eclipse provides numerous other tools that can be used together with Model RealTime. See www.eclipse.org for more information.

Many of the new features added in Model RealTime are intended to support collaboration and agile development methods. For example, the support for agile methods is visible in the focus on efficient build system integration, including enabling various continuous integration schemes. It is also visible in the extended support for parallel development in cross functional teams. The logical model merge functionality in Model RealTime enables development of different features in separate feature branches, allowing for example re-basing and delivery to a main release branch.

However, the change in underlying technology means that the migration from the RoseRT tooling is not completely straight-forward and necessitates some paradigm shifts to accommodate the integrations into Eclipse and Model RealTime. The purpose of this document is to share information and experience from RoseRT to Model RealTime migration projects done in the past to facilitate new migration efforts and provide guidelines on how to best overcome the issues that may arise.

2 The Good News

The good news is that Model RealTime is specifically designed to be a replacement product for RoseRT. Some key aspects:

- Model RealTime is designed to make it easy to migrate models from RoseRT
- Model RealTime has an importer that maps all capsule style RoseRT models to corresponding UML2 Model RealTime models
- Model RealTime has a special RT profile that captures the concepts of capsule, protocol etc from RoseRT
- Model RealTime includes support for the same C++ runtime kernel (more known as the "RT services library") as RoseRT
- Model RealTime contains a code generator that generates almost the same C++ code as the RoseRT code generator
- Model RealTime has a mechanism of incremental migration which allows to break the migration of large scale models into several steps and update elements separately after import without re-importing the whole model.

The Model RealTime importer can directly translate RoseRT models to the Model RealTime model format. It also includes many options to customize and fine-tune the import process.

The preserved RoseRT modeling style, using capsules, ports and other UML-RT concepts, makes it easy for developers experienced in RoseRT design to recognize and use the modeling constructs of Model RealTime because they are to a large extent the same as in RoseRT. The design language is still the same even if the tool is new and the modeling scheme is updated to be compliant with UML2.

The C++ runtime kernel that is the key to development of applications based on RoseRT is preserved with very few (backwards compatible) modifications in Model RealTime. The implication is that if special-purpose modifications have been done to the run-time kernel in RoseRT to adapt it to different target environments they will be easy to port to Model RealTime. In most cases they are likely to work out-of-the-box.

From a migration point of view the implication is that for a C++ modeling project the steps necessary are:

- Move the models from RoseRT to Model RealTime using the import feature of Model RealTime
- Generate C++ code from the imported model
- Test and update the build machinery used to create the applications from the C++ source code to fit into the Model RealTime Eclipse based build system
- Port and run the regression test suites used in the RoseRT context to Model RealTime
- Re-run the regression tests and verify the functionality of the application
- Continue the development in Model RealTime

To summarize: Model RealTime is designed to be a replacement for RoseRT, benefiting from the enhanced features that the Eclipse based development environment provides. The focus is on C++ based application development where the concepts and the runtime kernel from RoseRT is preserved to enable an easy migration path.

3 The Bad News

The bad news is that migrating from one development environment to another is NEVER a trivial exercise and migrating from RoseRT to Model RealTime is no exception. Development environments are complex products, and even though the RoseRT and Model RealTime products are designed to facilitate the migration effort both environments contain a massive amount of detailed settings and options that needs to be understood and taken into account in the migration efforts.

From a workflow perspective Model RealTime is not a copy of RoseRT; some use cases are performed in a similar fashion in both tools, some are done differently and some RoseRT scenarios are not supported by Model RealTime. It is important to understand the consequences of these differences when implementing a migration effort.

Expect that a significant time is required to prepare for migration. This includes both performing various test migrations and evaluating the outcome as well as preparing for and implementing training programs for the intended user community.

The effort is less concerning specification/analysis models that are used for viewing only compared to projects that generate code and produce running applications. Expect the effort to take longer time for large projects than small projects. Based on experience from already done migration efforts it is realistic to expect the process to take months rather than days or weeks. For a code generation project with 40-50 developers a realistic time schedule might be 6 months from the initial migration attempts to when the team starts using Model RealTime for production development.

One aspect that is very important to take into account is to compare the workflows and tool features used in RoseRT with what is available in Model RealTime. On a detailed level Model RealTime in many respects behave differently than RoseRT. Several of the less used RoseRT features are not supported in Model RealTime. Currently the supported languages are C++ and C, and only C++ is fully supported in the RoseRT importer. C models can be imported but require more post-processing. Mixed C and C++ models are also known to have limited support during import and requires manual post-processing. Furthermore, even if the key features are available in Model RealTime, since all projects are unique there is a risk that a new migration effort will detect some new issues not found in previous migration projects. For larger projects it is likely that either modifications of the used work flows or modifications in either Model RealTime or the import tooling will be necessary.

The conclusion is that migration efforts need to be carefully evaluated and planned. However, as shown by already finalized migration efforts, if done with a realistic mindset migration is both possible and recommended.

4 A Migration Project

A successful migration is a substantial effort and can best be handled as a project with a number of phases and activities. As mentioned above this kind of effort will in most cases take at least a couple of months, and for large projects six months could be a realistic time schedule.

The different activities that are recommended are essentially the following:

- Migration planning/evaluation
- Test migrations
- User training
- Migration week
- Post migration phase

Each one of these activities are described in the following sections.

4.1 Migration Planning/Evaluation

The migration planning/evaluation activity is focused on two aspects:

- Evaluating the feasibility and complexity of the migration effort
- Establishing a plan for the migration activities

The feasibility aspect of the migration is essentially to identify the features and work flows used in RoseRT and compare with what is available in Model RealTime. A key aspect is also to evaluate the platforms in terms of host and target operating systems, build systems, regression testing frameworks etc and understand the feasibility and complexity of integrating Model RealTime with these systems.

The migration planning contains actions to set up an initial timeline for the migration activities and perform an initial identification of available persons, who possess the required competence in RoseRT, Model RealTime, the migration process and the target application to perform the different activities. The key aspect when discussing timeline is to allow for sufficient calendar time for the test migration phase and to understand the project planning for the development project using RoseRT in order to be able to coordinate the migration project with the ongoing development projects. The key persons to identify are:

- Test migration engineers
- Training engineers

These roles are further discussed below.

An additional aspect to take into account in the planning is the release scheme of the Model RealTime product to identify a suitable version to have as target for the migration effort. This is important to take into account, since there might be releases planned to take place during the time span of the migration project. The Model RealTime development team will be happy to participate in this discussion.

4.2 Test Migration

The test migration activity is from an effort point of view in most cases the biggest part of the migration project. This is an activity where a representative set of RoseRT models are imported into Model RealTime and attempts are made to generate code, build running applications and test the generated application to verify that the functionality is correct also after the import. The end goal of the activity is thus to have all or at least a substantial part of the RoseRT models migrated and tested with successful result in the Model RealTime tool.

As part of the test migrations it is also necessary to document and potentially automate pre- or post-migration modifications that are necessary to achieve a successful migration. In addition the import settings that give the best results should be documented.

The complexity of this task is usually not in the model import itself, but rather in adapting the build and test systems to fit with Model RealTime. Other tools are often also used as part of the total development environment and if these tools have an integration with RoseRT, then they may need to be modified to work with Model RealTime.

Experience from performed migration activities shows that it is likely that issues will be encountered during the test migration phase that require contacts with HCL or HCL partner services, support or development. Unfortunately each individual non-trivial project tends to have its own specific circumstances that require some special treatment. From a migration project planning point of view this must be taken into account to allow for sufficient calendar time for the test migration activity.

4.3 User Training

The user training requirements depend on the general competence level of the development teams, but this must at least include sessions on how to use Model RealTime for engineers that know RoseRT, and sessions on the differences between the modeling languages used in RoseRT and Model RealTime. In many situations it is also beneficial to include Eclipse level training as part of the effort.

One possible way of implementing the training is to base it on the "train the trainers" concept. This is based on identifying key engineers that will become the initial experts of Model RealTime, train these "power engineers" in the new tool and then have them train the rest of the team.

4.4 Migration Week

The "migration week" is the period that ends with the development team being up and running using Model RealTime in their daily work. The required time for this activity of course depends on the complexity of the models and development environment and the size of the design team. However, a one week period has shown to be a realistic estimate. During this period the following should take place:

- The final work is done in RoseRT and a baseline is established
- The models are migrated from RoseRT to Model RealTime
- The complete development team gets access to Model RealTime and training is performed

The establishment of a baseline in RoseRT should preferably also include running all regression test chains a last time and verify that the models are correct in RoseRT before they are moved to Model RealTime.

The model migration may start with some manual modifications of the RoseRT model if this has been deemed necessary during the test migration period. Then the models are imported into Model RealTime and then potentially some manual post-processing is done on the imported models. Finally application code is generated and build is done based on the imported models and all regression test suites are executed to verify that the imported models are correct and that the application behavior is the same as before.

The training of the development team does not have to take place during the migration week. However, from a timing point of view it usually makes sense since the model migration in most cases is done by a subset of the engineers so there are a number of days where the RoseRT models are frozen and the Model RealTime models are not yet available. It therefore makes sense to spend these days on user training. Experience has also shown that if users are trained too early, long before they actually will use Model RealTime on a frequent basis, then there usually will be a need for a refresh course once the migration is completed.

4.5 Post Migration Phase

When the team starts to use Model RealTime, experience has shown that presence of engineers with Model RealTime knowledge facilitates the adoption of the product and reduces the initial problems for large teams to get up-to-speed using Model RealTime for development. If in-house expertise ("power users") exists this may be enough, but if this is lacking it is recommended to consider support from HCL services or from an HCL Partner.

During the post migration phase it has been found very useful to develop a project specific FAQ document as a means to quickly spread information to the team members. A scheme including daily meetings to discuss issues found when starting working with Model RealTime has also been used with good results.

Based on the experiences by the team it can also be a good idea to formalize practices by creating standardized preference settings or some small Eclipse plugins to streamline the work flow. However, usually such tasks are not urgent to be done right at the time of migration, even if they can be helpful.

5 Summary

Model RealTime is designed to give RoseRT users a path forward and enable an access to the features and capabilities that exist in the Eclipse and Model Real-Time environment.

The concepts used in RoseRT to describe models are preserved in Model RealTime in order to facilitate the transition to the new environment.

On code level, the Model RealTime C++ code generator is designed to work with the same run-time kernels as is used in RoseRT, so any adaptions or modifications of these kernels will in most cases work with no (or very limited) extra effort.

However, upgrading development environment and migrating from one environ-ment to another is never a trivial exercise and in order to be successful the effort needs to be planned as a project. In case of non-trivial models or large teams it may also make sense to consider migration support from HCL or an HCL partner.

6 Appendix: Links and References

The following documents related to RoseRT migration are also available at the same place as this document:

- Rational Rose Real-Time Migration to Model RealTime – Getting Started Guide
- Pre-migration Best Practices from Rose RealTime to Model RealTime
- Migration Best Practices from Rose RealTime to Model RealTime

For more information on the other tools and technologies that are enabled by migrating from RoseRT to Model RealTime see the following links:

Eclipse: http://www.eclipse.org/

Unified Modeling Language (UML): http://www.omg.org/spec/UML