Project DEPLOY

Grant Agreement 214158
"Industrial deployment of advanced system engineering methods for high productivity and dependability"

D43 JD3 — Final Assessment and Integration Results

April, 25 2012

http://www.DEPLOY-project.eu

Editors

Alexander Romanovsky (Newcastle University)
Cliff Jones (Newcastle University)

Contributors

Bosch:
Rainer Gmehlich
Katrin Grau
Felix Loesch

Siemens:
Hung Le Dang
Jérôme Falampin
Mikael Mokrani

SSF:
Dubravka Ilić
Timo Latvala
Thomas Långbacka
Laura Nummila
Tuomas Räsänen
Pauli Väisänen
Kimmo Varpaaniemi

SAP:
Andreas Roth
Wei Wei
Sebastian Wieczorek

Åbo Akademi University:
Linas Laibinis
Yuliya Prokhorova
Elena Troubitsyna

CETIC:
Jean-Christophe Deprez
Renaud De Landtsheer
Christophe Ponsard

Heinrich-Heine-Universität Düsseldorf:
Michael Leuschel

Newcastle University:
Alexei Iliasov
Ilya Lopatkin
Alexander Romanovsky

University of Pitesti
Alin Stefanescu

University of Southampton:
Michael J. Butler
Asieh Salehi Fathabadi
Abdolbaghi Rezazadeh

Reviewers

Carine Pascal (Systerel)
Michael Leuschel (Heinrich-Heine-Universität Düsseldorf)

Table of Content

1. Introduction

This deliverable reports on the Final Assessment and Integration Results in deployment work packages WP1-WP4. It consists of four chapters prepared by the project deployment partners (Bosch, Siemens, Space Systems Finland and SAP) in cooperation with the technology providers.

These chapters complement the previous deployment deliverables (D38, D39, D41, D42) by offering general summaries of the deployment activities conducted by the four deployment partners; the material here focuses on assessing the levels of deployment achieved and provides detail on the integration of the project results in the development processes (for systems and software) of the deployment partners.

These chapters address the issues related to achieving the following major project outcomes stated in the DEPLOY DoW:

- Each industrial partner will achieve real deployment of formal engineering methods and tools in development of products
- Each industrial partner will become self sufficient in the use of formal engineering methods within the lifetime of the project
- The deployments will enable us to provide scientifically valuable artefacts including
    - Formally developed dependable systems
    - Results of systems analysis including a rich repository of models, proofs and other analysis results
- The deployments will also enable us to provide a thorough assessment of formal engineering methods through
    - Evidence of productivity and dependability impact from deployment
    - Assessment of the tools and methods
    - Collecting experiences both positive and negative
- By extending the mathematical foundations of formal methods we will deliver research advances in complex systems engineering methods that enable
    - High degrees of reuse
    - High degrees of dependability
    - Effective systems evolution that maintains dependability
    - Modelling and analysis of real time systems
- By building on the existing Rodin tools platform we will deliver a professional open development platform based on Eclipse that
    - Provides powerful modelling and analysis capabilities
    - Is highly usable by practising engineers
    - Is tailored to sector-specific engineering needs
- Through the experience and insights gained in the industrial deployments we will deliver strategies that enable the integration of formal methods and tools with existing sector-specific development processes
- We will deliver training material and courses covering general and sector-specific formal engineering methods.

2. The Automotive Sector

2.1. Deployment Strategies

In WP1 several requirements models, specifications and Event-B models were developed during the lifetime of DEPLOY. The following three sections (Sections 2.1.1-2.1.3) give a short description of how the deployment strategy was separated into three phases whereas Section 2.1.4 gives a short overview of the methodology used in WP1 which was developed during the DEPLOY project.

- Minipilot: The minipilot is a small Event-B model, focused on specific aspects (in the case of WP1, modelling of continuous behaviour and time) to make project members familiar with Event-B
- Pilot: The goal of the pilot is to develop a specific methodology for automotive systems including an industrial process for formal development (necessary for large scale deployment) as well as to provide evidence for sector acceptance (by developing a close-to-production implementation of relevant parts of the cruise control system)
- Enhanced deployment: The enhanced deployment will result in the application of the methodology in the context of other domains which have different characteristics to make sure the methodology is not specific to the pilot application's characteristics only.

2.1.1. Minipilot

Applications realised in the automotive environment tend to be complex and distributed over hard- and software. The idea of the minipilot thus was to capture a manageable, yet typical element of the pilot application to demonstrate the concepts and functional range of Event-B. With switches and buttons being typical elements of the interface between the cruise control system and the driver, we started with a simple on/off switch as well as a button. Later, a three-way and an n-way switch were added. Apart from being simple, yet typical, elements of a cruise control system, the modelling of switches and buttons requires a simple time model. The time aspect is important in virtually all automotive applications.

During the minipilot the project members at Bosch became familiar with Event-B and addressed special aspects of an automotive application.

2.1.2. Pilot Deployment

For pilot deployment within WP1 we chose the cruise control system, an automotive system implemented in software which automatically controls the speed of a car. The cruise control system is part of the engine control software which controls actuators of the engine (e.g. injectors, fuel pumps, throttle valve) based on the values of specific sensors (e.g. gas pedal position sensor, air ow sensor, lambda sensor). Since the cruise control system automatically controls the speed of a car there are some safety aspects to be considered and it needs to fulfil a number of safety properties. For example, the cruise control system must be deactivated upon request of the driver or in case of a system fault.

In the pilot deployment a methodology for applying Event-B was developed (see Section 2.1.4 and [D19] for details). The central aspect of the developed methodology was that the gap between the initial requirements document and the Event-B model is too big to be easily taken in one step.

Therefore we used Problem Frames [J01] between the informal and the formal description of the system.

2.1.3. Enhanced Deployment

For the enhanced deployment in the automotive sector we chose the start/stop system. The start/stop system helps to save fuel and reduce emissions by turning the combustion engine off when it is not needed and turning it on again as soon as it is required. A typical scenario is stopping at a red light. Usually the engine would keep running (consuming fuel and producing emissions) although it is not needed until the light changes to green and the driver continues the journey. If the driver stops at the red light, changes the gear to neutral and releases the clutch, the start/stop system turns the engine off. As soon as the driver presses the clutch again to change the gear, the start/stop system starts the engine and the driver can move forward as usual. But it is not only the driver who influences the start/stop system. Revisiting the situation described before (driver stops at a red light, gear in neutral, clutch released) the start/stop system does not stop the engine if e.g. the state of charge of the battery is below a certain threshold.

During the enhanced deployment we took into account the lessons learnt of the pilot deployment and adapted the development process by introducing an additional step between requirements engineering and formal modelling (see Section 2.1.4 and [D38] for details).

2.1.4. Development Process

In this section we give a short overview of the development process which is described in more detail in [D19] and [D38]. Figure 2.1 shows that development of an Event-B model is only a part of the overall development process (see the grey box labelled DEPLOY WP1).

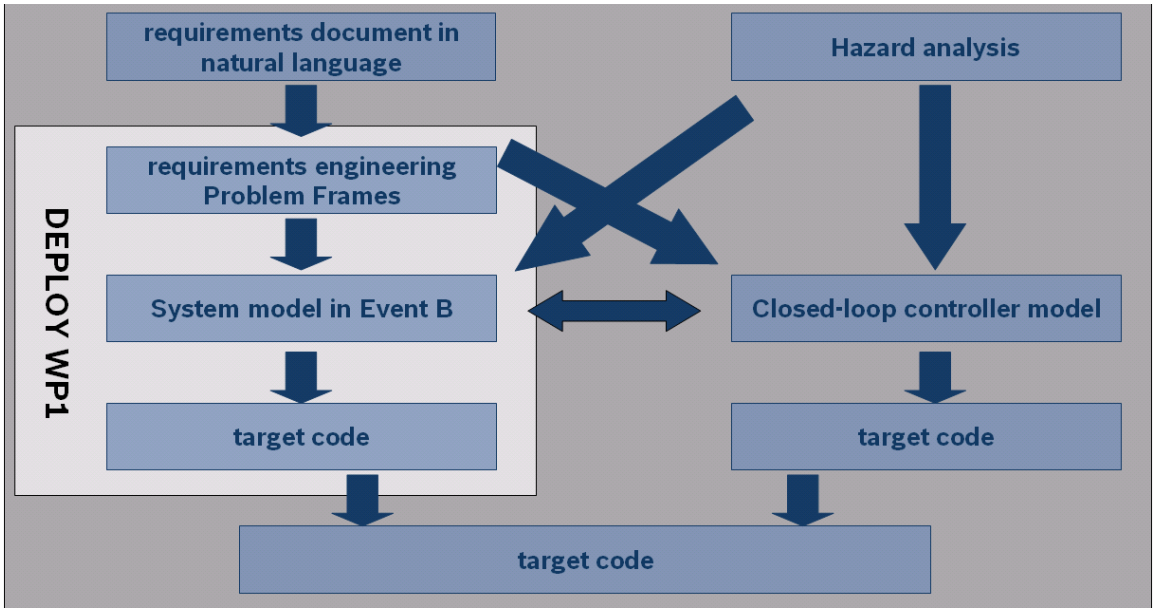


Figure 2.1. Overall development process - Pilot Deployment

Figure 2.2 presents the WP1 methodology developed during the DEPLOY project in more detail. The gap between the informal (natural language) world and the formal (Event-B) world is too big to be easily taken in one step and therefore was divided into several smaller steps. We added two intermediate phases: Problem Frames and System Specification with RSML [LHHR94]. For a detailed description of these steps see [D19] and [D38].

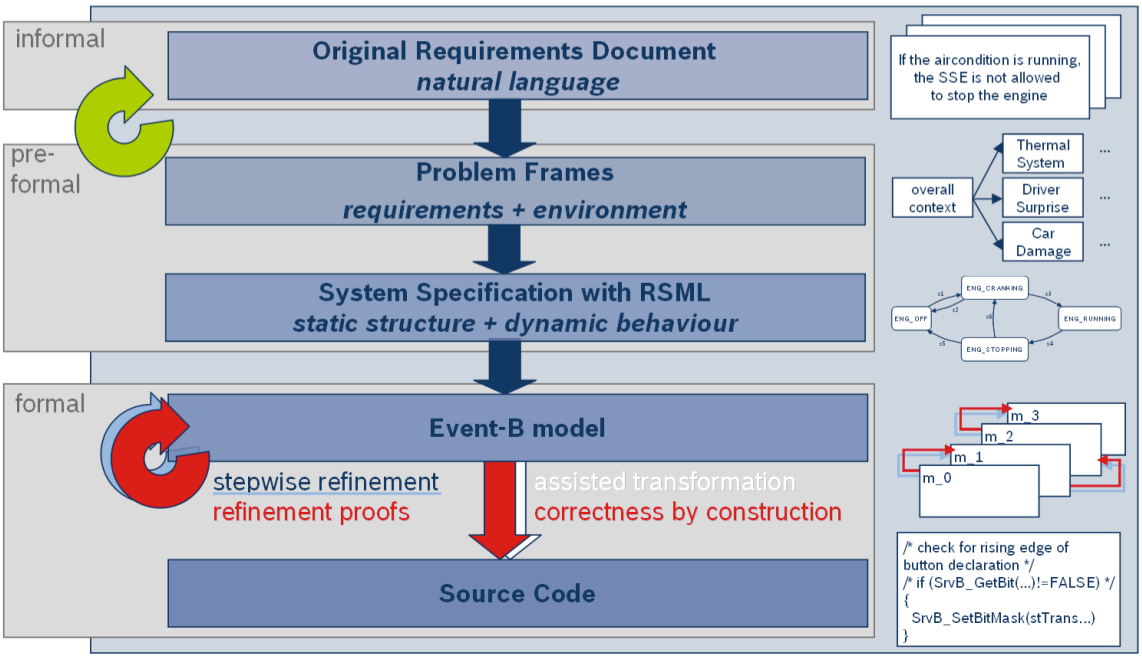


Figure 2.2. Development Process

2.2. Deployment Assessment

Three different aspects are addressed in this section:

1. The quantitative results of applying our requirements engineering method during the pilot deployment
2. The results concerning the specification and formal modelling
3. The results concerning the gathered proof obligations in the Event-B model

2.2.1. Requirements

Figure 2.3 shows the quantitative results of applying our requirements engineering method to the cruise control system. On the x-axis the total effort in working hours is shown. On the y-axis four curves are plotted. The first curve shows the total number of requirements as a reference. This curve changes over time due to the addition of new requirements (shown in a separate curve) and the rejection of original requirements (shown in the third curve).

In total we ended up with nearly half the number of text units (requirements) needed to describe the required functionality of the cruise control system. Thus, by applying our requirements engineering method, we were able to reduce the total number of text units (requirements) by more than 40 percent. The total effort we spent on the restructuring and improvement of the requirements amounts to approximately 300 working hours.

The evaluation results clearly show that by applying our structured requirements engineering method the understandability and coherency of the requirements can be increased while at the same time the total number of requirements can be reduced by more than 40 percent. Furthermore, our concept of hierarchical requirements allows us to differentiate between a complete requirements set on a system level and a more detailed requirements set on a functional level.

As a result of this we do not always expect to reduce the number of requirements by 40 percent. But we are convinced that we are usually able to reduce the number of requirements and to increase the quality in the same step.

For more details about the pilot deployment see [D19].

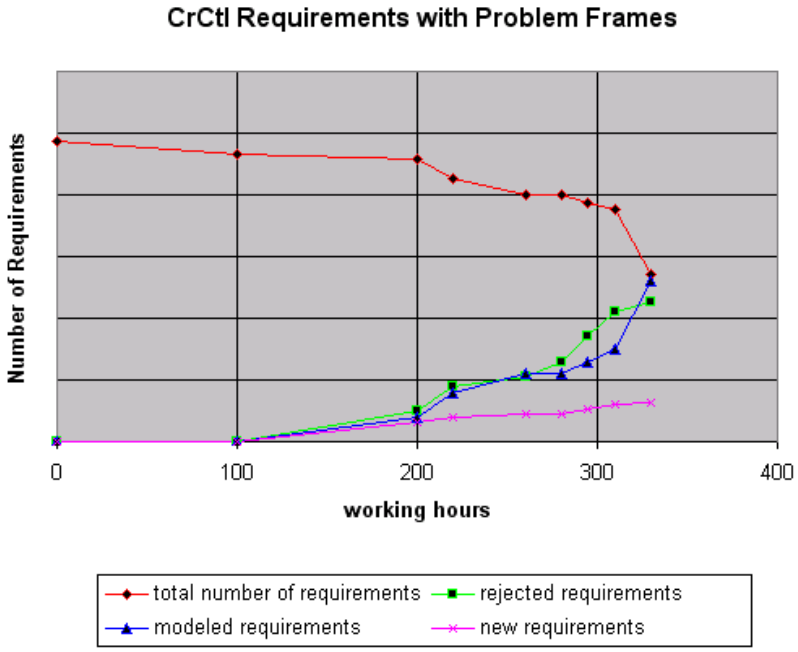


Figure 2.3 Quantitative Results – Requirements

2.2.2. Specification and Formal Modelling

During pilot and enhanced deployment both systems were formally modelled with Event-B. During pilot deployment (cruise control) the Problem Frames model was taken as input for the formal specification in Event-B. Although we provided a mapping between problem frame elements and Event-B elements the formal modelling phase itself took around 4 months. The formal modelling phase consisted of translating the requirements stated in problem frames into Event-B models. Due to the different structure of the Problem Frames model and the Event-B model and the missing support for an architectural design in Event-B this process took considerable time.

In order to speed up the formal modelling phase and to close the gap between the informal and the formal world, we decided to introduce another phase between requirements development in Problem Frames and formal modelling in Event-B, namely specification. During specification the requirements stated in the problem frame diagrams were modelled using RSML, a notation for

state machines. Using this notation the effort for formal modelling in Event-B could be reduced from 4 months (120 days) to 6 weeks (45 days). Parts of this reduction have been made possible because the complexity of the formal modelling task was reduced to a task of simply translating the state machines described in RSML into Event-B events.

Figure 2.4 shows a comparison of the required effort for formally modelling the cruise control and the start/stop system. On the y-axis the total effort for formal modelling in Event-B is shown. This effort only includes the part of developing the formal model in Event-B and does not include the effort required for proving invariants and other properties of the model. As you can see from Figure 2.4 the required effort for modelling the start/stop system was only 45 days whereas the cruise control system required approximately 120 days for formal modelling. This increase in productivity was partly due to the fact of the introduction of the specification phase.

For more details about specification with RSML and formal modelling with Event-B see [D19] and [D38].

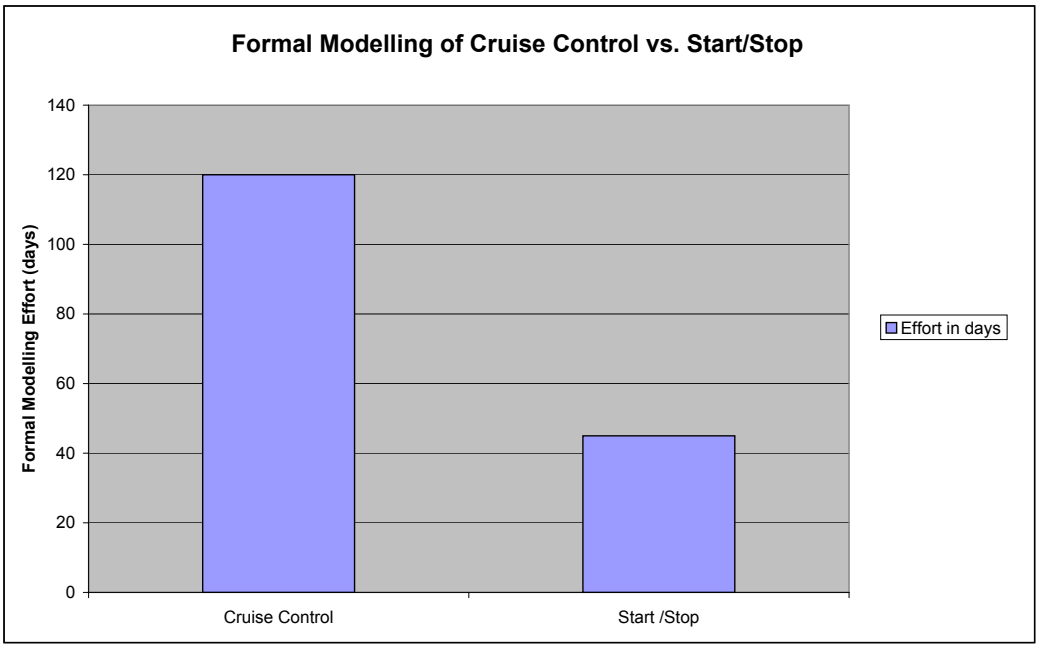


Figure 2.4. Quantitative Results - Formal Modelling

2.2.3. Proof Obligations

In the second application (start/stop system) we started not with a requirement set, we developed the needed requirements by ourselves. We did not produce a comparable chart to the one for the cruise control system. In the second pilot we were more interested in gathering evidence that the proof obligations, which arise using Event-B, are manageable.

In Figure 2.5 the absolute number of proof obligations (4215) for the start/stop system model is shown. From these 4215 PO (proof obligations) the majority was proven automatically. 425 have to be proven manually.

Figure 2.6 shows a more detailed view on the 425 manual proofs. 400 of them are very easy (needed time < 1min), 12 of them are medium (needed time < 1h). 13 of the manual proofs are difficult (needed time > 1h). This evaluation shows that it is in principle feasible to prove a system of the size of the start/stop system.

For more details about enhanced deployment see [D38].

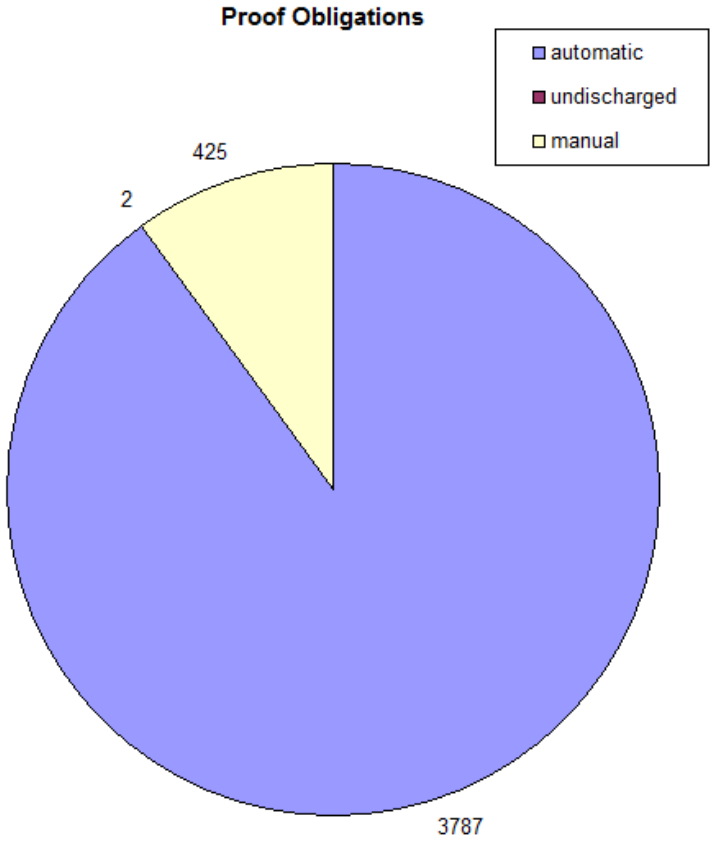


Figure 2.5. Quantitative Results - Automatic vs. manual proof obligations

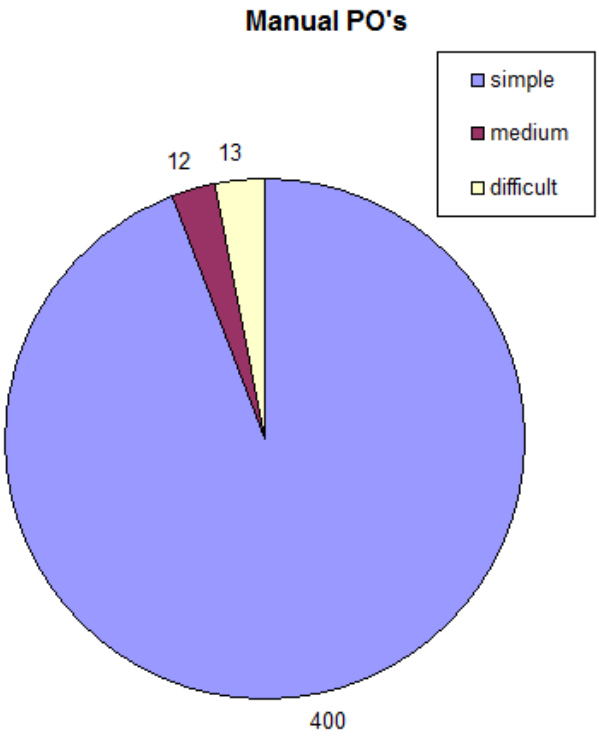


Figure 2.6. Quantitative results - Distribution of manual proof obligations

2.3. Assessment of method and tools

The Rodin platform has made big progress during the runtime of DEPLOY. These improvements of the platform itself and the development of several plug-ins according to our needs is a success. Especially the reaction time of the tool development team to the industrial needs was very good and is a good example of the collaboration of industrial and academic partners. Beside these very positive aspects, Rodin itself still needs further improvement to be considered mature enough for industrial use. In the following two sections several aspects in which improvement is needed are listed. Section 2.3.1 concentrates on issues related to tools, Section 2.3.2 lists methodological aspects.

2.3.1 Tools

Performance

There is a big progress in performance aspects. Nevertheless during the modelling of the start stop system we still run in performance limitations of the tool. Keeping in mind the relative small size of the start stop system (compared to other industrial applications) the performance is not good enough. Further improvement is needed.

Stability

With stability issues we have nearly the same experience as with performance. There has been big progress, which is very good, but improvement is still needed. During the modelling of the

start/stop system we had several system crashes. System crashes in a production environment are not acceptable. The stability of the Rodin platform itself is very mature, but in addition with several plug-ins the stability decreases significantly.

Team development

Due to prioritisation of other topics there is progress in support of team development, but not enough. Parallel development is not possible (or only in an ad hoc manner). Versioning of proofs is not possible.

Proof Obligations

Most of the proof obligations generated by Rodin were either discharged automatically or could be easily proven manually. This is a very promising result as we expected the proofs to be more difficult and time-consuming.

2.3.2 Method

Architecture

This is one of the biggest drawbacks of Event-B and Rodin. There is no significant support of architecture in Event-B itself and therefore also in Rodin. Model refinement is not a sufficient replacement for model decomposition and the supported decomposition/modularization possibilities were not suited for our needs.

Time

Event-B has no support for modelling time. With the flow plug-in we are able to specify the order of events which can be seen as an abstraction of time. This is encouraging progress but not sufficient.

2.4 Training

The training and documentation material was incomplete at the beginning of the DEPLOY project especially as new plug-ins emerged which also need documentation, tutorials, etc. But as there were enough Event-B experts involved in the project, questions could be asked directly. Although the project members at Bosch were not familiar with Event-B at the beginning of the project we achieved a good level of experience in applying Event-B and were able to gain a high degree of autonomy.

We emphasized on several occasions the importance of good documentation (to make knowledge transparent and available) and as a reaction to this feedback the project spent considerable effort to improve documentation. Nevertheless the support of a community with expertise in different aspects of Event-B and Rodin (e.g. prover configuration, manual proofs, refinement strategy) must not be underestimated as well as the benefit of having experts supporting the education in comparison to solely relying on documentation.

2.5 Conclusions

This section discusses the overall conclusions (taking into account both the pilot and enhanced

deployment and therefore the experience of 4 years working with Event-B) about Event-B and the industrial use of Event-B.

Closed loop controller. There is no practical way of modelling closed loop controllers in Event-B and therefore it is not possible to prove properties about continuous behaviour. This is a completely open field with the need for lot of research. One of the main questions is: is it useful to try this and if yes how could this be done?

Time. Time was from the very beginning an open issue and is still. In the embedded systems market there is a strong need to include at some point during the development a notion of time. Including ordering of the events (cf. the flow plug-in developed at Newcastle University) is a first step to address this problem.

Gap between informal and formal world. We made encouraging progress in WP1, which gives us confidence that this problem is (or could be) solved. What is needed here is more experience in applying the developed methods and strong tool support. As mentioned in Section 2.2 we were able to increase the quality of the requirements by applying Problem Frames as well as by formally proving invariants in Event-B.

Formal modelling. We made significant progress in formal modelling of industrial size applications. Beside the progress, especially the cruise control application, exposed the borders of formal modelling and the tools support for formal modelling in Rodin. There is room for improvement.

Flexibility. The Rodin tool is flexible enough to be adjusted to different needs via the development of plug-ins. For example the flow plug-in (see [Ili11]) helped us to graphically specify a desired order of events which was necessary to prove important properties of our system.

Industrial development process. There is an open issue left in supporting state of the art industrial development processes. Especially the supporting processes (configuration management, variant management, team development, version management) are not (or not good enough) supported by Rodin.

Scalability. Rodin does not scale with large applications. There is work left related to decomposition (architecture), performance and stability.

Taking these conclusions into account, Event-B is a promising method for industrial use especially in the embedded market. For a full deployment the listed open issues have to be solved.

References

[D19] DEPLOY Deliverable D19: D1.1 Pilot Deployment in the Automotive Sector WP1. http://www.deploy-project.eu/pdf/D19-pilot-deployment-in-the-automotive-sector.pdf.

[D38] DEPLOY Deliverable D38: D1.2 Report on Enhanced Deployment in the Automotive Sector WP1. http://www.deploy-project.eu/pdf/DEPLOY

[Ili11] Alexei Iliasov. Rodin - New Flows Plugin. http://wiki.event-b.org/index.php/Flows, January 2011.

[J01] Michael Jackson. Problem Frames: Analyzing and structuring software development

problems. Addison-Wesley Longman Publishing Co., Inc. 2001.

[LHHR94] Nancy G. Leveson, Mats Per Erik Heimdahl, Holly Hildreth, and Jon D. Reese. Requirements specification for process-control systems. IEEE Trans. Softw. Eng., 20:684-707, September 1994.

3. The Transportation Sector

These are the abbreviations used in Chapter 3.

Abbreviation	Definition
ATPF	*Full Automatic Train Protection*
ATC	*Automatic Train Control*
ATO	*Automatic Train Operation*
ATP	*Automatic Train Protection*
ATPR	*Automatic Train Protection Restricted manual mode*
ATS	*Automatic Train Supervision*
Automaton	*State machine*
CBTC	*Communication Based Train Control*
CC	*Carborne Controller*
EB	*Emergency Braking*
EdithB	*Tool developed by STS, that automatically generates refinements of a B model*
EFS	*Equipment Functional Specification*
FMEA	*Failure Mode and Effects Analysis*
FSA	*Functional Safety Analysis*
FTA	*Fault Tree Analysis*
HMI	*Human-Machine Interface*
IXL	*Interlocking*
MAL	*Movement Authority Limit*
NV	*Non Vital*
PHA	*Preliminary Hazard Analysis*
RAMS	*Reliability, Availability, Maintainability, Safety*
SAS	*System Architecture Specification*
SIL	*Safety Integrity Level*
SRS	*System Requirements Specification*
STS	*Siemens Transportation Systems*
TO	*Train Operator*
V	*Vital*
Vital	*Safety critical*
ZC	*Zone Controller*

3.1. Realised deployments

In the DEPLOY project context, for the first time, Siemens tried to define a process using Event-B in modelling Transportation Systems. As Siemens has considerable experience of applying formal methods to software components of railway systems, for DEPLOY the challenge is to raise this to the level of overall systems in order to address system safety. Siemens has been using B method for more than 15 years, and a considerable investment has been made in tools and methods. In particular, an automatic refinement tool (Event-B) has been developed to allow the (almost) automatic production of the concrete B model from the abstract B model. It is therefore important that the use of Event-B at the system level does not impose new investments at the software level.

For this purpose, Siemens defined a process including Event-B for the Transportation Systems development. This process was applied to carry-out minipilot and pilot prototypes of the CBTC "manage operating modes" function. By developing minipilot and pilot prototypes, it appeared quite quickly that probabilities had to be added in the model. An experiment has been performed on the minipilot to add probabilities, with success. The realisation of minipilot and pilot gave us a confidence that a large scale Event-B development is feasible with the proposed process.

Another achievement is related to the integration of ProB in the data validation of CBTC controller software component which is still problematic for every deployment of CBTC systems on site. The old process based on Atelier B revealed several drawbacks, in particular with huge data properties. The motivation is therefore to automate the proof on huge data properties with alternative technologies. Siemens was interested in ProB because this tool provides services to deal with B properties in order to animate and model check B models. The success story with ProB improves significantly the data validation process at Siemens. It does not require B experts to carry out data validation. Indeed, the B experts are required only in case of problem, whereas in the former process, B experts were required in any case, for long and fastidious tasks. In addition, using ProB significantly reduces the time checking data properties: from 2 or 3 days with Atelier B to 2 or 3 hours per project.

3.2. Results of the Minipilot and Pilot Deployment

The aim of the minipilot and pilot prototypes was to bring evidence that an Event-B development at industrial scale is feasible, following the defined process.

The minipilot aim was to quickly face some modelling issues (timing, probabilities), without too much development workload. This minipilot helped to find out that probability is required for system modelling, and that timing issues related to input/output of controllers (that seems to be a very low level problem) has an impact at the higher system level.

Indeed, by developing minipilot and pilot prototypes, it appeared quite quickly that proving safety (or availability) properties without any failure is pointless, since in that case we can only prove that the system is safe, provided nothing wrong happens!

It is therefore needed to model failures. But then, some proof obligations (safety properties after a failure) cannot be discharged any more. To prove these properties, it appeared that probabilities had to be added in the model. An experiment has been performed on the minipilot to add probabilities, with success with one machine.

3.2.1. Comparison between Lifecycle with/without Event-B

An eventual adoption of Event-B life cycle at Siemens implies many changes. Below we show a comparison between lifecycle with/without Event-B.

Development:

	Classical B	Event-B
SRS/SAS document	Yes	Yes
EFS document	Yes	Yes (automaton)
Refinement Plan	No	Yes
Software specification document	Yes	No
Event-B model	No	Yes (translated from automatons + manual)
Abstract B model	Manual	translated from Event-B model
Concrete B model	generated by EdithB	generated by EdithB New EdithB rules required
Ada-PSC code	Translated (except base machines, manually written)	Translated (except base machines, manually written)

With Event-B, one document has to be written in a different manner (EFS will be mainly automatons, instead of natural language). Automatons are already used (scarcely) in EFS, so this notation is already used and understood by system engineers.

With Event-B, the software specification document is suppressed, and the refinement plan is added: in both cases, these documents (written in natural language with possibly pseudo-B notations) support the modelling work.

With Event-B, there is no need to write the abstract B model anymore (since it is translated from the Event-B model), but of course, it is required to write the Event-B model.

Validation

	Classical B	Event-B
PHA	Yes (PHA document)	Yes (PHA doc + model)
FSA on EFS	Yes (FSA document)	Yes (FSA document + model)
FSA on SWRS	Yes	No

FMEA	optional	Automatic generation, need to be reviewed
Traceability tables	Yes (manual)	Yes (automatic)
Proof of Event-B model	No	Yes
Animation at system level	impossible	Yes (no simulator required)
Abstract B model analysis	Yes	No
analysis of basic machines	Yes	Yes
Proof of classical B model	Yes	Yes
Functional test	Yes (all functions)	Yes (for functions not animated)

With Event-B, the scope of the Functional Safety Analysis on the Equipment Functional specification is larger, since it includes the analysis of the EFS and the Event-B model analysis. However the equivalent extra work at the software level (abstract B model analysis) is suppressed.

3.2.2. Event-B Deployment and Open Issues

Below is a list of open issues relating to an eventual deployment of Event-B in an industrial development of Transportation Systems.

Standards

The applicable standards at software level (CENELEC EN 50128) mention that formal methods (and in particular B) are "highly recommended" for safety critical software and specification, but no process or activities are described to define how to use formal methods. At system level, formal methods are not mentioned at all. This means that there is currently no informative or normative chapter about formal methods in Cenelec Standards, and a railway industrialist has to define a process that is acceptable to both the customer and certification bodies.

Training system engineers on Event-B

In the DEPLOY context at Siemens, Event-B models have been created by software engineers who have very little knowledge at system level. It seems that to get attention of system engineers on Event-B is still problematic. In the one hand, they do not want to change their working way. In the other word, they all think that formal method is very hard to learn.

Probabilities and Failures

As said earlier, in order to model a transportation systems, it is needed to model probabilities in the model. Our approach to model probabilities as a global variable P work well with an Event-B specification without refinement, however, it can not work with a model composed of several refinement levels. So the modelling and reasoning about probabilities are still open issues for an eventual deployment of Event-B in the transportation system development.

Visualisation of differences between Event-B machines/projects

One aspect of an Event-B development is to validate Event-B models wrt. SRS, refinement plan or EFS. We often need to validate that the changes in Event-B models respect the modifications made in SRS, RP and EFS.

Tool validation

This point is very important, indeed Rodin and its plug-ins (Event-B model decomposition, Event-B model generation, UML2B, the prover plug-ins and the associated tactics) will have to be validated in order to get confidence from industrial bodies who intend to use it in an industrial development.

3.3. Deployment of ProB at Siemens

In order to evaluate the feasibility of using ProB for checking the topology properties, Siemens sent the STUPS team at the University of Düsseldorf the models for the San Juan case study on the 8th of July 2008. There were 23,000 lines of B spread over 79 files, two of which were to be analysed: a simpler model and a hard model.

In order to handle the B models sent by Siemens, several improvements on ProB have been realised, in particular the Parser and the Type Inference. Several new data structures are implemented in ProB as well. Such improvements enable ProB being scaled up to Siemens case studies.

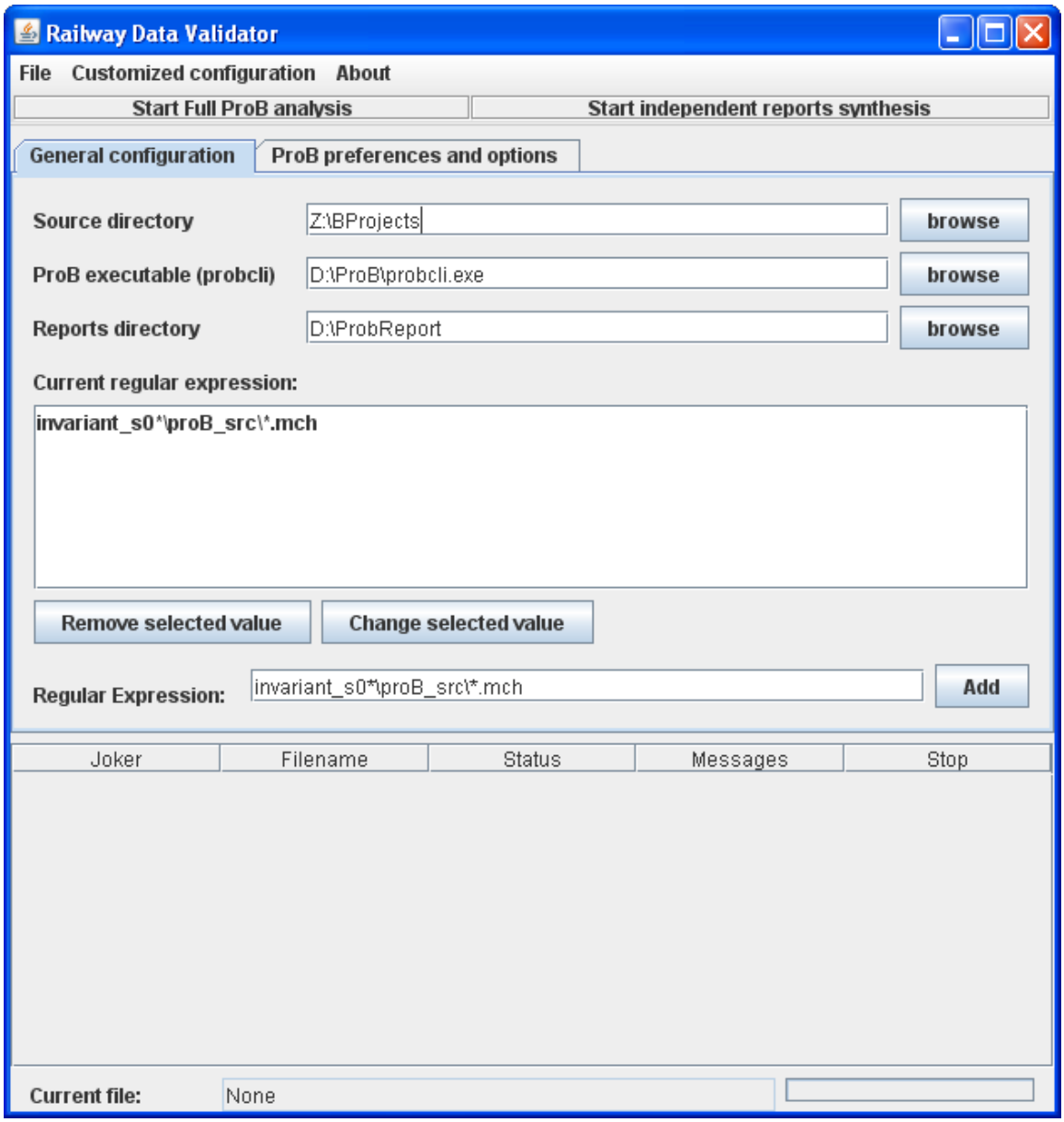
A complete animation of San Juan case study on 8th December 2008 revealed four errors that Siemens had uncovered themselves by manual inspection. Note that the STUPS team were not told about the presence of errors in the models (they were not even hinted at by Siemens), and initially STUPS believed that there was still a bug in ProB. The manual inspection of the properties took Siemens several weeks (about a man month of effort). Checking the properties takes 4.15 seconds, and checking the assertions takes 1017.7 seconds (i.e., roughly 17 minutes) using ProB 1.3.0 on a MacBook Pro with 2.33 GHz Core2 Duo.

3.3.1. Railway Data Validator

In the first experiments, ProB was used instead of Atelier B, on the same IVP (Invariant Validation Project), with great success. But the creation of IVP was still problematic, with few atomizations.

Each IVP is an encoding of a specific wayside configuration data in B; this is required in order to validate the configuration data against the formal properties in the generic B project.

The goal was to create a tool that could automatically generate the B projects (containing assertions machines), run ProB on created B projects, and collect the result in a synthesis report. In addition, this tool should not work only on ZC (Zone Controller) data, but also on CC (Carbone Controller) data which were not formally validated before the use of ProB. Indeed, in the CC software development, the topology data are contained in textfiles and loaded “on the fly” by the CC software component when needed. Therefore, in order to enable the CC data validation, the macros in definition files are derived from topology text files instead of Ada programs. Such macros are then merged with variables defined in basic invariant machines to assertion machines for the segment in question.



RDV is a new tool realized by Siemens. Via a graphical interface, RDV provides following services:

IVP Generation: this function generates an IVP for a sub-section in case of ZC or a segment in case of CC. The generated IVP is almost ready to be used by ProB or Atelier B. Indeed, we still need some manual modifications related to properties of non-function constants, however, in comparison with the former tool, it reduces significantly manual modifications on generated B machines. In addition, with the file selection function based on regular expression, RDV enables the generation of a subset of assertions machines (i.e., only machines with modifications). Moreover, the automation of CC IVP creation is a great help for safety engineers as there are about several hundreds IVP to be created (one project per segment).

ProB Launch: RDV enables users to parameterize ProB before launching it. ProB is called on each assertion machine in order to analyse the assertions contained in each machine. It does not require B experts to carry out data validation. Indeed, the B experts are required only in case of problem, whereas in the former process, B experts were required in any case, for long and fastidious tasks. In addition, using ProB significantly reduces the time checking data properties: from 2 or 3 days with old tool to 2 or 3 hours per project with RDV. The analyse of CC assertion machines with ProB reduces significantly the interaction with user as one does not need to launch Atelier B several hundreds times on created CC IVP.

Assertion-Proof Graph Generation: This function provides a graphical way to investigate the proof on an assertion. This is based on a service provided by ProB to compute values of B expressions and the truth-values of B predicates, as well as all sub-expressions and sub-predicates. This is very useful because users often want to know more about the exact source why an assertion fails. This was one problem in the Atelier B approach: when a proof fails it is very difficult to find out why the proof has failed, especially when large and complicated constants are present.

Validation Synthesis Report: The results of analysis realized by ProB on assertions machine are recorded in a set of .rp and .err files (one per B component and per sector/segment). Each rp (report) file contains the normal results of the analysis with ProB:

- Values used during initialization of machines;
- Proof details on each assertion checked;
- Result of the analysis (true, false, timeout...).

Each err (error) file contains the abnormal results of the analysis:

- Variables/Constants with incorrect type;
- Variables/Constants with multiple values, redefinition of value or missing value;
- Error during execution of ProB (missing file...).

When all report and error files have been generated, an html synthesis report is issued. This report gives the result (number of false/true assertions, timeout, unknown results...) for each sector/segment. For each B component, a hyperlink to the detailed results, error and report files give access to the results of the component (in order to know which assertion is false, for instance).

3.3.2. Deployment & Current Use of ProB at Siemens

Per agreement with our clients, we still need to use Atelier B in conjunction with ProB. But ProB has proven to be more effective and less restrictive than Atelier B for railway data validation. We have currently used it on all on-going projects.

Alger line 1 (ZC): This project has 2 sectors. ProB has been used for the last 3 versions of this project railway data. For the last version, the results are as follow:

Filename (runtime)	Predicates	TRUE	FALSE	UNKNOWN	TIMEOUT
pas_as_inv_s01.html	1174	1164	10	0	0

pas_as_inv_s02.html	1174	1162	12	0	0
Total	2348	2326	22	0	0

Each line represents the summary result for one sector. The Predicates column shows the number of assertion to be analysed, the TRUE column represents the number of assertions verified by ProB (no counter-example found by ProB). The FALSE column represents the number assertions failed by ProB (with counterexample). The UNKNOWN column represents the number of assertions that ProB does not know how to verify. The TIMEOUT column corresponds to assertions that ProB encountered a time out problem during analysis. For each sector, there were two assertions un-proved with Atelier B due to their complexity. One of which is shown below. This property has rightfully been proven wrong with ProB with the railway data for both sectors.

ran(inv_quai_variants_nord_troncon >< ((((((((((t_quai_pas <| inv_quai_adh_red_nord_rg_variant_bf_i) |> t_rg_variant_bf) \/ ((t_quai_pas <| inv_quai_ato_inhibe_nord_rg_variant_bf_i) |> t_rg_variant_bf)) \/ ((t_quai_pas <| inv_quai_mto_inhibe_nord_rg_variant_bf_i) |> t_rg_variant_bf)) \/ ((t_quai_pas <| inv_quai_atp_inhibe_nord_rg_variant_bf_i) |> t_rg_variant_bf)) \/ ((t_quai_pas <| inv_quai_arret_tete_nord_rg_variant_bf_i) |> t_rg_variant_bf)) \/ ((t_quai_pas <| inv_quai_arret_centre_nord_rg_variant_bf_i) |> t_rg_variant_bf)) \/ ((t_quai_pas <| inv_quai_arret_queue_nord_rg_variant_bf_i) |> t_rg_variant_bf)) \/ ((t_quai_pas <| inv_quai_tete_ape_nord_rg_variant_bf_i) |> t_rg_variant_bf)) \/ ((t_quai_pas <| inv_quai_centre_ape_nord_rg_variant_bf_i) |> t_rg_variant_bf)) \/ ((t_quai_pas <| inv_quai_queue_ape_nord_rg_variant_bf_i) |> t_rg_variant_bf))) /\ ran(inv_quai_variants_sud_troncon >< ((((((((((t_quai_pas <| inv_quai_adh_red_sud_rg_variant_bf_i) |> t_rg_variant_bf) \/ ((t_quai_pas <| inv_quai_ato_inhibe_sud_rg_variant_bf_i) |> t_rg_variant_bf)) \/ ((t_quai_pas <| inv_quai_mto_inhibe_sud_rg_variant_bf_i) |> t_rg_variant_bf)) \/ ((t_quai_pas <| inv_quai_atp_inhibe_sud_rg_variant_bf_i) |> t_rg_variant_bf)) \/ ((t_quai_pas <| inv_quai_arret_tete_sud_rg_variant_bf_i) |> t_rg_variant_bf)) \/ ((t_quai_pas <| inv_quai_arret_centre_sud_rg_variant_bf_i) |> t_rg_variant_bf)) \/ ((t_quai_pas <| inv_quai_arret_queue_sud_rg_variant_bf_i) |> t_rg_variant_bf)) \/ ((t_quai_pas <| inv_quai_tete_ape_sud_rg_variant_bf_i) |> t_rg_variant_bf)) \/ ((t_quai_pas <| inv_quai_centre_ape_sud_rg_variant_bf_i) |> t_rg_variant_bf)) \/ ((t_quai_pas <| inv_quai_queue_ape_sud_rg_variant_bf_i) |> t_rg_variant_bf))) = {}

Sao Paulo line 4 (ZC): This project has three sectors. ProB has been used for the last 6 versions of this project railway data. For the last version, the results are as follow:

Filename (runtime)	Predicates	TRUE	FALSE	UNKNOWN	TIMEOUT
pas_as_inv_s036.html	1465	1459	6	0	0
pas_as_inv_s037.html	1465	1460	5	0	0
pas_as_inv_s038.html	1465	1457	8	0	0
Total	4395	4376	19	0	0

ProB has detected issues with a group of properties which had to be commented in machines used with Atelier B because they were crashing the predicates prover. Here an example of one of them:

```
!(cv_o,cv_d).(((cv_d : t_cv_pas & cv_o : t_cv_pas) & cv_o :
inv_lien_cv_cv_orig_i[inv_chainage_cv_liste_i[inv_chainage_cv_deb(cv_d) ..
inv_chainage_cv_fin(cv_d)]])

& not(inv_lien_cv_cv_dest_i((t_cv_pas <| inv_lien_cv_cv_orig_i~) |>
inv_chainage_cv_liste_i[inv_chainage_cv_deb(cv_d) ..
inv_chainage_cv_fin(cv_d)](cv_o)) = cv_d)

=> inv_lien_cv_cv_dest_i((t_cv_pas <| inv_lien_cv_cv_orig_i~) |>
inv_chainage_cv_liste_i[inv_chainage_cv_deb(cv_d) ..
inv_chainage_cv_fin(cv_d)](cv_o)) : inv_cv_pas_modifiable_i~[{TRUE}]))
```

Thankfully, after analysis, we have concluded that the problem was not critical. Nonetheless, without ProB, it would have been a lot harder to find these problems. In this case, the assertion-proof graphs were useful to understand better where the problems were coming from.

Paris line 1 (ZC): this project has 6 sectors. ProB has been used for the last 7 versions of this project railway data. For the last version, the results are as follow:

Filename (runtime)	Predicates	TRUE	FALSE	UNKNOWN	TIMEOUT
pas_as_inv_s011html	1503	1501	2	0	0
pas_as_inv_s012.html	1503	1498	5	0	0
pas_as_inv_s013.html	1503	1496	7	0	0
pas_as_inv_s014.html	1503	1499	4	0	0
pas_as_inv_s015.html	1503	1498	5	0	0
pas_as_inv_s016.html	1503	1498	5	0	0
Total	9018	8990	27	0	0

Roissy LISA: this project has 3 sectors. ProB has been used for the last 3 versions of this project railway data. For the last version, the results are as follow:

Filename (runtime)	Predicates	TRUE	FALSE	UNKNOWN	TIMEOUT
ry_pads_as_lisa_inv_pa31.html	1038	1038	0	0	0
ry_pads_as_lisa_inv_pa32.html	957	957	0	0	0
ry_pads_as_lisa_inv_pagat.html	1038	1038	0	0	0
Total	3033	3033	0	0	0

3.3.3. ProB validation

The IVP generation tool has been empirically validated. For each project, we have verified that all important data were kept in the modified machines and that Atelier B and ProB were still type checking them. The ProB launcher and synthesis report generator were also empirically validated. Their validity is also base on the ProB validation which was carried out by Dusseldorf Team who has realised

- over 1000 manual entered unit tests at the Prolog level, which check the proper functioning of the various core predicates operating on B's data structures.

- Over 500 regression tests which are made up of B models along with saved traces. These tests are valuable in ensuring that a bug once fixed remains fixed. They are also very effective at uncovering errors in different ProB components (parser, type checker, B interpreter, ProB kernel, etc.) which are critical components for the data validation.
- Self-Model check with Mathematical Laws. The idea is to formulate a wide variety of mathematical laws and then use the model checker to ensure that no counter-example to these laws can be found. ProB now check itself for over 500 mathematical laws which covers laws for Booleans (39 laws), arithmetic laws (40), laws for sets (81), relations (189), functions (73) and sequences (61) as well as some specific laws about integer ranges (24) and various basic integer sets (7).

The above tests are complemented by a code coverage analysis which shows that all predicates and clauses of the ProB kernel are covered by tests. Some clauses cannot be covered because they are only reachable through internal errors.

3.3.4. Conclusions

Siemens has now a long experience with B development at software level. But the minipilot and the pilot development raised new technical issues that were not addressed during the past experience: in particular, the modelling of failures is something new, requiring new techniques (use of probability linked with the Event-B model).

The minipilot and the pilot resulted in an industrial process that will be followed in the future Event-B developments at Siemens.

The integration of ProB in the software safety validation process has been clearly a great success.

References

M. Leuschel, J. Falampim, F. Fritz, and D. Plagge. Automated property verification for large scale B Models. In A. Cavalcanti and D. Dams, editors, Proceedings FM 2009, LNCS 5850, pages 708-723, Springer-Verlag, 2009.

Jérôme Falampim. DEPLOY Deliverable D16. Pilot Deployment in Transportation. September 11th 2009.

Jérôme Falampim. DEPLOY Deliverable D41. Enhanced Deployment in Transportation. June 14th 2011.

Michael Leuschel, Jens Bendisposto, Sebastien Krings and Daniel Plagge. ProB Validation Report.

4. The Space Sector

4.1. Introduction

In this chapter we provide the final feedback by SSF on the results achieved by DEPLOY in the space domain. We briefly overview the course of the work, explain how the results of the development have influenced deployment directions and discuss the achievements and the open issues.

The chapter is structured as follows: in the Section 4.1 we discuss the deployment strategy. In Sections 4.2-4.4 we give a detailed feedback on the method and tools with which we have experimented. Section 4.5 provides a general assessment of FM adaptation in SSF. Section 4.6 looks into deployment in the space sector. In Section 4.5 we draw the conclusions and outline the way forward.

Deployment strategy

The greatest challenge in the space projects is to ensure traceability of the requirements and validate that requirements have been properly implemented. Software development is also significantly influenced by RAMS (Reliability, Availability, Maintainability and Safety) activities that often lead to requirements changes. The goal of DEPLOY project was to explore how formal modelling and verification can facilitate structuring requirements, deriving robust system architecture and increasing degree of development automation.

Our approach throughout the project was to experiment with the formal modelling of several typical space applications to understand how to reap the benefits of rigorous engineering and pave the path towards integrating it into the existing development practice.

Initial pilot

As a pilot, SSF chose to model a considerable subset of requirements of Bepi Colombo (Europe's first mission to Mercury, see [RD2]) SIXS (Solar Intensity X-ray and Particle Spectrometer) / MIXS (Mercury Imaging X-ray Spectrometer) OBSW (On-Board Software), the specification and implementation of which is to a large extent under the responsibility of SSF. The pilot focused on modelling telecommand processing service and verification of the processing flow.

Enhanced pilot

As a pilot SSF has chosen a system with richer set of architectural properties – Attitude and Orbit Control System (AOCS). SSF is expected to be involved into several projects concerned with AOCS in the future. To set the requirements for the pilot, SSF has summarised its design experience gained while developing GOCE (Gravity Field and Steady-State Ocean Circulation Explorer, see [RD2]) DFAC (Drag-Free Attitude and Orbit Control) to define generalised requirements for AOCS.

The work on the pilot has ignited research on formal modelling of mode-rich systems and creating techniques for systematic specification of mode-logic. A variation of AOCS – a distributed AOCS was proposed to experiment with formal verification of dynamic behaviour with model checking.

Currently research on integrating RAMS activities is getting a new spin. In cooperation with Aabo, SSF is experimenting with modelling dynamically reconfigurable systems as well as

deriving safety cases from formal models.

Abbreviations

AOCS	Attitude and Orbit Control System
CTL	Computation Tree Logic
DFAC	Drag-Free Attitude and Orbit Control
DoW	Description of Work
DSAOCSS	Distributed System for Attitude and Orbit Control for a Single Spacecraft
DSL	Domain-Specific Language
ECSS	European Cooperation for Space Standardisation
ESA	European Space Agency
ESTEC	European Space Research and Technology Centre
ETHZ	Eidgenössische Technische Hochschule Zürich
FDIR	Failure Detection, Isolation and Recovery
FM	Formal Method(s)
FMEA	Failure Modes and Effects Analysis
FP	Framework Programme
GM	Generic Mode
GOCE	Gravity Field and Steady-State Ocean Circulation Explorer
ID	Identifier
LNCS	Lecture Notes in Computer Science
LTL	Linear Time Temporal Logic
MIXS	Mercury Imaging X-ray Spectrometer
MM	Mode Manager
M*n*	n^{th} Month in DEPLOY
NASA	National Aeronautics and Space Administration
OBSW	On-Board Software
PUS	Packet Utilisation Standard
PV	Process View
QA	Quality Assurance
RAMS	Reliability, Availability, Maintainability and Safety
SIXS	Solar Intensity X-ray and Particle Spectrometer
SRD	Software Requirements Document
SSF	Space Systems Finland, Ltd.
TC	Telecommand
TM	Telemetry
T*x*.*y*	y^{th} Task in DEPLOY WP*x*
UM	Unit Manager
UML	Unified Modelling Language

4.2. Modelling Activities in the Development Process

4.2.1. Modelling of BepiColombo SIXS/MIXS OBSW Requirements

In the WP3 pilot deployment (see [RD5]), SSF created Event-B models for a considerable subset of requirements of the OBSW, with focus on handling of telecommands and on production of telemetry packets, i.e., on application of PUS (Packet Utilisation Standard, see [RD1]). While modelling in Event-B we had to clarify certain requirements. As a result of spotting certain ambiguities the requirements were changed. Though it is hard to assess whether the thorough requirements inspection would have had the same impact, we can certainly say that DEPLOY has facilitated requirements engineering of the OBSW.

After the initial modelling several other modelling attempts were undertaken (see [RD6]). At that phase, no new requirement was modelled and no feedback to requirement authors got produced. Instead, the focus was on trying to construct better models by utilising the modular and decompositional extensions (see [RD7]) of the Event-B language. Modelling with the modular extension was done by SSF, whereas modelling with the decompositional extension was done by Southampton. The work on the modular modelling produced a useful tool feedback that allowed the developers to improve the tools as well as analyse different approaches to using modularisation in Event-B development.

Development of OBSW has demonstrated that for some systems modelling of dynamic behaviour poses the main challenge. For instance, OBSW does not have complex safety constraints represented by the corresponding invariants. This was not surprising for modellers at SSF because a typical software requirement expresses what the software should do in a given situation. Events alone suffice for modelling such requirements. Invariant-like higher-level requirements may exist but may consider equipment in a way that the implications to the software are “shallow” or too abstract or beyond the scope of software requirements. Formal modelling of the systems similar to OBSW should be more focused on reasoning about liveness-type of properties. For instance, at the second phase of the project we have experimented with using UPPAAL tool to analyse dynamic behaviour and timing aspects of OBSW. Modelling of Attitude and Orbit Control Systems

For the enhanced deployment phase, SSF has decided to experiment with a system that has a richer set of safety properties. The requirements for the case study were written in the Ada programming language. In that way, a pre-tested specification with a precise semantics got produced from scratch in less than one month. SSF’s Event-B activities (see [RD5]) on this AOCS were carried out within a couple of months and got completed in January 2010. SSF’s Event-B models considered in [RD5] are statement-by-statement models of the Ada presentation. Many people in the Event-B community seem to be categorically against such a way of modelling, but many of the used arguments are unconvincing in the sense that source code modelling has for many years been a main-stream approach in computer-aided verification. In January 2010, SSF decided not to continue statement-by-statement modelling, but only after the whole Ada presentation had already been modelled and primarily just because too much work was needed for proving the invariants of interest. Also note that some of those invariants were proven indeed, and for each remaining invariant it seemed possible to construct a proof.

As with BepiColombo, formal modelling has resulted in several changes of the Ada representation of the requirements. Aabo and Newcastle have created several Event-B models “inspired by AOCS”. Some of these models are considered in the papers [RD12], [RD13] and [RD15]. The goal of this work was to propose a generic method for modelling mode-rich systems

in Event-B. In particular, the system specification was structured according to architectural layers and consistency conditions between mode logic at different layers were defined. Originally, AOCS is a centralised control system. A dedicated component – manager of the global mode logic is responsible for system mode transitions. A natural extension of the work on modelling mode-rich systems is to experiment with verification of mode-logic of distributed systems. In December 2010, Aabo and Newcastle suggested that SSF would specify and model a distributed AOCS. A corresponding specification is now available as [RD23]. A mode synchronisation protocol designed from scratch is in a sense the "core" of the specification. SSF's formal method activities on the distributed AOCS have so far solely concentrated on the problem of whether the protocol is correct for its purpose. Various trial-and-error attempts to prove certain associated invariants have unsuccessfully been made using Rodin platform. Some evidence for correctness in cases where the number of modes is 3 and the number of protocol partners is 2 has been obtained (see [RD24]) using ProB [RD18]. The case of 3 modes and 3 partners somehow seems to be beyond the capabilities of explicit-state model checkers such as ProB, so a few weeks have been spent in trying to find counterexamples in that case by means of bounded model checking, using the symbolic model checker NuSMV [RD17].

4.3. Assessment of Event-B and Rodin platform

Event-B is essentially a guarded command style language where events and guards can be specified using a simply typed first-order logic. Since the language is Event-Based, it is eminently suited for modelling behaviour that can be captured well with state machines. Capturing algorithmic computation requires more effort.

An Event-B model has two parts: the context and the machine. Roughly speaking the context captures static definitions such as types while the machine models the behaviour. Refinement can be used to split the modelling into several successive steps where each model provably is a behavioural refinement of the more abstract machines. This allows for some limited management of model complexity, but experience has shown that it mostly functions as an aid to manage proof complexity.

The Rodin platform is the main tool used for Event-B modelling and proof activities in DEPLOY. Though Event-B in principle is independent of tools, the implementation in Rodin platform is dominating in the sense that there is no competing Event-B tool.

Serious lack of scalability discourages industrial use of Rodin platform, no matter how much effort is put on improving the competence of users. Very high consumption of memory without any really good reason is perhaps the worst concrete form of this lack of scalability and has side effects such as non-response (e.g. due to thrashing in use of virtual memory) and "disappearing auto-provability" (i.e. failure in automated discharging of a proof obligation that under earlier circumstances has got automatically discharged). Such consumption occurs in quite "medium size" models and typically does not seem to be due to "inherent complexity".

A lot of time in interactive proving tends to get spent in selecting of needed hypotheses and in deselecting of unneeded hypotheses. Rodin platform tends to be unable to draw any obvious conclusion when there are many selected hypotheses.

The Event-B proof methodology in Rodin platform questionably ignores the problem of inconsistent combinations of axioms and/or guards. In cases of inconsistency, even a very careful

user may produce arbitrarily many proofs without noticing the inconsistency, as it is not necessarily likely for an inconsistent combination of axioms and/or guards to get used in a single proof. Rodin platform should definitely have a proper interface for proving consistency among axioms and guards.

Despite certain improvements during DEPLOY, the type system in Event-B is uncomfortable if not inadequate. For example, there is no proper support to abstract data types, i.e. types defined by means of inductive axioms, as it is strikingly difficult to produce any inductive proof in Rodin platform. (We would like to note here that we did not evaluate the theory plugin recently developed in the project.)

In plain Event-B, lack of modularity is not only a readability problem but also a source of arbitrarily much work that could be avoided with proper utilization of modularity. Due to work done by Newcastle and Aabo in WP3, a modular extension of Event-B and an associated plug-in of Rodin platform now exist and are considered in detail in Section 4.4.

A trained person can construct a medium complexity Event-B model in a few weeks. However, spending a few weeks on writing an Event-B model tends to be more frustrating than spending the same amount of time on writing a program in high-level programming language. One explanation is that any high-level programming language inherently provides more "interesting variation" than Event-B.

Let us end the assessment with some positive observations:

- A proof on paper can often be mechanized using Event-B and the Rodin platform.
- Many things can be expressed in Event-B much more compactly than in many other formal languages.
- Ensuring correctness of proofs and increasing the degree of automation in proofs have had a high priority in the development of the Rodin platform.
- Many useful plug-ins of the Rodin platform have been created in DEPLOY.
- The latest version of the Rodin platform is definitely much more convenient to use than the version that was available in the beginning of DEPLOY.

4.4. On the Use of Methods and Tools

Here we consider methods and tools “beyond what is default in Rodin platform”.

4.4.1. Model Checking

It is often the case that a property of interest is expressible as a temporal logic formula but not as an invariant. Such properties are beyond the scope of proper Event-B proof methodology but not necessarily beyond the scope of Event-B. ProB [RD18] and the ProB plug-in [RD7] provide model checking of LTL and CTL formulas such that the syntax of atomic formulas is the same or almost the same as the syntax of predicate expressions in Event-B.

In preliminary work for pilot deployment in 2008, SSF used ProB in search for deadlocks in a “root machine” of an Event-B model of BepiColombo SIXS/MIXS OBSW requirements. A

deadlock was found indeed, and the machine was revised accordingly. While using ProB for more detailed machines we have found it difficult to compute consistent instances of the used Event-B contexts.

In Autumn 2011, SSF used ProB in checking of LTL formulas that concern the "3 modes and 2 partners" case of the mode synchronisation protocol designed for a distributed AOCS. Except for the formulas, the protocol descriptions were written in Event-B using Rodin platform. The ProB plug-in was used for producing machines for the stand-alone ProB that was used for the actual model checking. Though the plug-in itself has model checking facilities, SSF preferred to use the stand-alone tool in order to avoid unnecessary feature interaction and in order to utilise late improvements that were available for the stand-alone tool but not for the plug-in.

The Event-B description of the mode synchronisation protocol got revised several times as ProB reported counterexamples to expected properties. The found errors were modelling errors, i.e. mismatches between the verbal specification and the Event-B description. (The verbal specification itself was later revised due to an error that was found in inspection without any tool.)

Some attempts were made in order to use ProB in the "3 modes and 3 partners" case. However, ProB tended to run out of memory almost regardless of the intended model checking activity. A support to memory efficient state space generation actually got implemented in ProB soon after the developers had been informed about the problem.

In model checking, it is often wise to consider several formalisms and tools. For the period December 2011 – January 2012, SSF has used model checking outside Event-B, still working on conjectured invariants that had originally been written to be proven in Rodin platform. Most of these experiments have been done using bounded model checking in the symbolic model checker NuSMV [RD17]. Several model checking processes have been run in several processors for several weeks. So far all results confirm that a desired property holds in all states that are reachable from the initial state within an expressed number of steps. Though NuSMV has options for concluding whether a check is complete w.r.t. the state space, it may be impossible to reach such a conclusion. In some processors, some checks have already got stopped due to running out of memory. The archive [RD25] contains one of the used protocol descriptions in NuSMV modelling language, some formulas that have been considered and some associated script and log files on NuSMV experiments.

The model checking plug-ins for the Rodin platform provide a useful way of debugging models, and in some cases even proving model properties. Although problems have been encountered, the model checking plug-in have developed and matured at a rapid pace.

4.4.2. Real-Time

Design of space software is often dominated by real-time requirements. The idea to reuse Event-B models to verify real-time requirements is explored in the report [RD10] that also contains a small case study for demonstrating the approach. This research is also described in Section 4.2 in the deliverable [RD6]. Process View (PV) modelling with explicit notions of processes and their synchronisation was suggested as the modelling approach, and verification of properties not expressible as invariants was suggested to be handled by model checking, via a translation to the model checking tool UPPAAL [RD22] that is dedicated to real-time aspects.

4.4.3. FMEA in the Development of Layered Mode-Rich Control Systems

In our work on ensuring mode consistency of AOCS we have relied on the mode logic that has been defined a priori. However, often mode logic has to be defined by the system developers. To facilitate this process we proposed a structured approach to defining fault tolerance part of mode logic, e.g. backward mode transitions.

Embedded control systems typical for space sector are often developed in a layered fashion, which provides the designers with a convenient mechanism for structuring the system behaviour according to the identified architectural layers. Each software component in layered mode-rich systems can be viewed as a mode manager. Let us assume that the scenario defines how to bring a system from the non-operational mode to the most advanced mode. Such a forward mode transition scenario is usually given in the system requirements document.

The implementation of the mode transition scenario can be interrupted either by transitional errors (i.e., errors that appear during a mode transition step) or unit usability errors (i.e., errors that occur when a unit performs below its required level). When the mode manager (MM) chooses a new target mode, it initiates the corresponding mode transitions in the lower layer unit managers (UMs). If an error is detected, the corresponding unit manager assesses the error and either initiates error recovery by itself or propagates the error to MM. MM, in its turn, makes a decision how to handle such an error. This decision usually involves rolling back to some less advanced (i.e., degraded) mode.

To systematically define the rollback procedures for each mode, we proposed to conduct Failure Modes and Effects Analysis (FMEA). FMEA is a well-known inductive safety analysis technique. For each system function or component, it defines possible failure modes, local and system effects, as well as detection and recovery procedures. The information is collected in a table form. The traditional FMEA allows us to discover and structure failure modes of components. We propose to conduct FMEA of each operational mode. Therefore, we tailor FMEA to fit our purposes. In our work [RD20] (see also [RD16] and [RD19]) we have proposed a systematic approach to deriving the fault tolerance part of a mode logic using FMEA. Also, we have formalised the required conditions of mode consistency and demonstrated how to ensure them while developing a system by refinement in Event-B. Verification by theorem proving and stepwise refinement have allowed us to undertake a formal development of a complex control system — Attitude and Orbit Control System (AOCS). Hence our approach shows a good scalability.

4.4.4. Modularisation

Rodin platform's modularisation plug-in [RD7] (see also [RD11]), designed by Newcastle and Aabo as a part of WP3, provides facilities for structuring Event-B developments into logical units of modelling, called modules. The module concept is very close to the notion of classical B imports. However, unlike a conventional development, a module comes with an interface. An interface defines the conditions on how a module may be incorporated into another development (that is, another module). The plug-in follows an approach where an interface is characterised by a list of operations specifying the services provided by the module. An integration of a module into a main development is accomplished by referring operations from Event-B machine actions using an intuitive procedure call notation.

There are at least the following reasons to split a development into modules.

- Structuring large specifications: it is difficult to read and edit a large model; there is also a

limit to the size of model that the Rodin platform may handle comfortably and thus decomposition is an absolute necessity for large scale developments.

- Decomposing proof effort: splitting helps to split verification effort. It also helps to reuse proofs: it is not unusual to return back in refinement chain and partially redo abstract models. Normally, this would invalidate most proofs in the dependent components. Model structuring helps to localise the effect of such changes.

- Team development: large models may only be developed by a (often distributed) developer team.

- Model reuse: modules may be exchanged and reused in different projects. The notion of interface makes it easier to integrate a module in a new context.

A modularisation plug-in experiment on BepiColombo SIXS/MIXS OBSW requirements was carried out at SSF in August – September 2010 and was focused on a few of the requirements that had been considered in earlier non-modular experiments.

The original goals of the experiment were as follows:

- Systematic isolation of activity details and related conditions to modules in such a way that the machines using the modules do not replicate much of what is expressed inside the modules.

- Precision of descriptions of the considered behaviour about as accurate as in the most detailed available non-modular Event-B model.

- Avoidance of massive atomic activities. Long chains of atomic activities do not realistically model concurrency.

- To deal with "module integration invariants". Such an invariant refers to variables of more than one module.

- Reasonable total proof effort (including time spent in "iterative optimisation") without compromising the above goals.

The final Event-B project of the experiment can be understood to sufficiently meet all the above-mentioned goals, except possibly the proof effort reasonability goal. However, the proof effort was to a certain extent more reasonable than in some earlier Rodin platform experiments.

The problems encountered in the tools during the experiment have been solved in later releases of the plug-in. Since October 2010, the plug-in has been in a good shape with respect to the features used in the experiment.

By following certain modelling conventions it is possible to significantly improve the usability of modularisation plug-in. Designers of the plug-in recommend the following conventions.

- Avoiding very large and complicated operation post conditions, especially involving existential quantifiers to simplify proofs. In general, complex post-conditions can be simplified by introducing additional module variables and invariant properties on these variables.

- Refraining from using operation calls to model returned exceptions. To achieve the same effect one might strengthen preconditions of calling events by checking external module variables. Rather than using returned composite values, which can include the status indicating success or a particular occurred exception, additional external module variables storing such a status of the latest call can be introduced.

- Avoid generating new values supplied by the environment inside of operation post conditions. The problem can be circumvented by introducing additional local variables in a calling event and then forwarding the value of these local variables as extra parameters of an operation call. Alternatively, module processes can be used for modelling such input from the environment.

Even when these guidelines are followed, usability of the plug-in suffers from a ``macro-style'' approach in the sense that in interactive proving, the user deals with the output of a translator and is assumed to understand that output as if it were the original form. However, solving this problem would likely involve a lot of work, whereas e.g. the record type plug-in [RD7] of Rodin platform has essentially the same problem.

4.4.5. Decompositional Approaches

Atomicity Decomposition

In atomicity decomposition [RD8], coarse-grained atomicity is refined to more fine-grained atomicity. New events introduced in a refinement step are viewed as hidden events not visible to the environment of a system and are thus outside the control of the environment. Any number of executions of an internal action may occur in between each execution of a visible action.

Model Decomposition

The notion of model decomposition [RD8] covers machine and context decomposition. The entry point for the decomposition of a model is a machine and its whole hierarchy of seen contexts. The resulting sub-models are independent of each other, and the partition of the models includes the allocation of variables, invariants, events and even context elements like sets, constants and axioms to subparts.

Model decomposition has two main styles. In shared variable style, events are partitioned into sub-components in different machines, variables associated with the original events get shared by the machines, and the sub-components can be refined independently but only in such a way that shared variables are present and not data-refined. In shared event style, variables are partitioned into sub-components in different machines, events associated with the variables get split accordingly, and the sub-components can be refined independently without constraints.

Experience at University of Southampton

University of Southampton maintains a decomposition plug-in [RD7] and, as documented in [RD8], has used the plug-in for applying the above-described decompositional approaches to BepiColombo SIXS/MIXS OBSW. SSF's non-decompositional Event-B projects have been used as de facto primary specifications in this experiment that has turned out useful in development of decompositional practices.

SSF has no relevant experience about the decomposition plug-in but sees the above-described approaches being motivated by a several decades long history of process-algebraic approaches.

4.4.6. Code Generation

Several code generation plug-ins [RD7] for Rodin platform are being developed. SSF is mostly interested in C code generation and has a few times informed the plug-in developers about aspects that are important from SSF's point of view. SSF has so far not performed extensive evaluation of the plug-ins.

4.5. Overall Assessment of Internal FM Adoption

This section is scoped to SSF. Dubravka Ilić, Timo Latvala and Kimmo Varpaaniemi have had the role of formal method experts. Thomas Långbacka specified the system considered in the WP3 minipilot. Pauli Väisänen did most of the AOCS specification work for the pilot. All these people have a research background on formal methods, though only one has pre-DEPLOY experience on Event-B. Kimmo Varpaaniemi is the main author of almost every Event-B project released by SSF. Matti Anttila, Timo Latvala, Thomas Långbacka and Tero Vihavainen attended the Event-B training in Zürich in April 2008. Laura Nummila and Tuomas Räsänen spent several months inside SSF in 2010 on getting familiar with Event-B and Rodin platform. This internal training is documented in the report [RD21] and in the deliverable [RD6] Section 4.6.

DEPLOY is so far the only project at SSF where Event-B has been used, even though SSF has had prior to DEPLOY other similar also academic projects. Consequently, the tradition of using formal methods exists to some extent, but it is not utilised in any commercial way. DEPLOY showed us approximate costs and risks involved in training personnel without any prior experience in formal methods, but also revealed the most common limitations to wide adoption of FMs. The most arguable of these is the justification of the effort of using FMs (adoption, learning curve, application) vs. benefits in the real commercial project. Even if some potential exists as Rodin tools (platform + numerous useful plug-ins) showed, more easy-to-use, comprehensive and automated tools are needed, as recognized by most of the SSF staff who participated in the trainings. We can freely say that almost one third of the company had some hands-on the tools and methodology resulted within DEPLOY but as outlined above, currently there is not one commercial project where this experience would be directly applied. This does not mean, however, that DEPLOY achievements are diminished — on the contrary — we believe that the experience gained in this project showed us greater perspective for improving some other development practices we already have.

4.6. Strategies on Space Deployment of Formal Methods

Even though space domain specific applications were of the most interest in the project and our initial key requirements for using FMs were drawn to satisfy these initial needs, practice and experience gained during the project lifetime showed that the domain specific requirements could be generalized and are as valid and applicable as those in other fields. As our experience of FMs grew larger, unforeseen needs started driving our further project engagement. For instance, SSF would benefit from code generation, even though this was not of big interest during the project lifetime. With now more mature experience on methods and tools usage, we can look forward to new improvements like incorporation of FMEA results etc. Modularization plug-in as well seems to bring a substantial potential when it comes to developing mode-oriented systems, quite typical not only in the space domain but other domains within SSF's line of business as well.

In general, the experience gained in the project creates a great base for further following up the tools and methods improvement especially those that are of greater need for SSF and which were not covered by this project alone. As we currently see it, FMs have significant potential to be used in the space domain but require much stronger verification power and higher automation level. Expressivity and ergonomy are important but not necessarily dominating factors. (For example, a useful model checker with an uncomfortable language is better than a less useful model checker with a comfortable language.)

4.7. Conclusions

Originally starting involvement in DEPLOY project with only very few people familiar with formal methods, evolved into having a community with improved competence on formal methods capable of applying their experience in a specific space domain and wider. The interest was mutual. Feedback from SSF has had a certain impact on development of methods and tools in DEPLOY. Specific project needs originated from SSF's line of work droved development of additional plug-ins and made somewhat significant improvement of the platform as well through feedback on various case studies.

Even though SSF's focus has slightly changed throughout the project, the main achievements do fulfil initial expectations and form a solid base for possibly further application and exercise of DEPLOY results within company's other businesses as well.

References

Applicable Documents

[AD1] Information Society Commission of the European Communities, Information Media Directorate-General, and Communication Technologies. *Seventh Framework Programme Grant Agreement No 214158, Industrial Deployment of Advanced System Engineering Methods for High Productivity and Dependability. Collaborative Project*. December 2007.

[AD2] Grant Agreement [AD1] *Annex I – Description of Work*. Approved by EC in November 2007.

[AD3] DEPLOY Project. *Industrial Deployment of Advanced System Engineering Methods for High Productivity and Dependability. Consortium Agreement*. May 2008.

[AD4] DEPLOY Project. *DEPLOY Deliverable D18 – D12.3 Project Refocus – Amended Description of Work*. August 2009.

Reference Documents

[RD1] ECSS Secretariat, ESA-ESTEC, Requirements & Standards Division. *Space Engineering: Ground Systems and Operations – Telemetry and Telecommand Packet Utilisation (ECSS-E-70-41A)*. http://www.ecss.nl/, January 2003.

[RD2] ESA. *GOCE Earth Explorers*. http://www.esa.int/esaLP/LPgoce.html, January 2012.

[RD3] ESA Media Center, Space Science. *Factsheet: BepiColombo*.

http://www.esa.int/esaSC/SEMNEM3MDAF_0_spk.html, January 2012.

[RD4] DEPLOY Project. *DEPLOY Deliverable D5 JD1: Report on Knowledge Transfer*. http://www.deploy-project.eu/pdf/fv-d5-jd1-reportonknowledgetransfer.zip, January 2009.

[RD5] DEPLOY Project. *DEPLOY Deliverable D20 D3.1 – Report on Pilot Deployment in the Space Sector*. http://www.deploy-project.eu/pdf/D20-pilot-deployment-in-the-space-sector-final-version.pdf, January 2010.

[RD6] DEPLOY Project. *DEPLOY Deliverable D39 D3.2 – Report on Enhanced Deployment in the Space Sector*. http://www.deploy-project.eu/pdf/D39-Enhanced%20Deployment%20in%20the%20Space%20Sector.pdf, August 2011.

[RD7] Event-B and Rodin platform Documentation Wiki. *Rodin Plug-ins*. http://wiki.Event-B.org/index.php/Rodin_Plug-ins, January 2012.

[RD8] Asieh Salehi Fathabadi, Abdolbaghi Rezazadeh, and Michael J. Butler. *Applying Atomicity and Model Decomposition to a Space Craft System in Event-B*. In Mihaela Bobaru, Klaus Havelund, Gerard J. Holzmann, and Rajeev Joshi (Eds.), *NASA Formal Methods, Third International Symposium, NFM 2011, Pasadena, CA, USA, April 18–20, 2011, Proceedings*, LNCS 6617, pp. 328–342, Springer, http://www.springerlink.com/content/d8q6028518110157/, April 2011.

[RD9] Alexei Iliasov, Linas Laibinis, Elena Troubitsyna, and Alexander Romanovsky. *Formal Derivation of a Distributed Program in Event B*. In Shengchao Qin and Zongyan Qiu (Eds.), *Formal Methods and Software Engineering, 13th International Conference on Formal Engineering Methods, ICFEM 2011, Durham, UK, October 26–28, 2011, Proceedings*, LNCS 6991, pp. 420–436, Springer, http://www.springerlink.com/content/4454051k14384878/, October 2011.

[RD10] Alexei Iliasov, Linas Laibinis, Elena Troubitsyna, Alexander Romanovsky, and Timo Latvala. *Augmenting Event B Modelling with Real-Time Verification*. Technical Report 1006, Turku Centre for Computer Science, http://tucs.fi/research/publication-view/?pub_id=tIlLaTrRoLa11a, April 2011.

[RD11] Alexei Iliasov, Elena Troubitsyna, Linas Laibinis, Alexander Romanovsky, Kimmo Varpaaniemi, Dubravka Ilić, and Timo Latvala. *Supporting Reuse in Event B Development: Modularisation Approach*. In Marc Frappier, Uwe Glässer, Sarfraz Khurshid, Régine Laleau, and Steve Reeves (Eds.), *Abstract State Machines, Alloy, B and Z: Second International Conference, ABZ 2010, Orford, Québec, Canada, February 22–25, 2010, Proceedings*, LNCS 5977, pp. 174–188, Springer, http://www.springerlink.com/content/1246876j83576368/, February 2010.

[RD12] Alexei Iliasov, Elena Troubitsyna, Linas Laibinis, Alexander Romanovsky, Kimmo Varpaaniemi, Dubravka Ilić, and Timo Latvala. *Developing Mode-Rich Satellite Software by Refinement in Event B*. In Stefan Kowalewski and Marco Roveri (Eds.), *Formal Methods for Industrial Critical Systems: 15th International Workshop, FMICS 2010, Antwerp, Belgium, September 20–21, 2010, Proceedings*, LNCS 6371, pp. 50–66, Springer, http://www.springerlink.com/content/w8281922t2471515/, September 2010.

[RD13] Alexei Iliasov, Elena Troubitsyna, Linas Laibinis, Alexander Romanovsky, Kimmo Varpaaniemi, Pauli Väisänen, Dubravka Ilić, and Timo Latvala. *Verifying Mode Consistency for On-Board Satellite Software*. In Erwin Schoitsch (Ed.), *Computer*

Safety, Reliability, and Security: 29th International Conference, SAFECOMP 2010, Vienna, Austria, September 14–17, 2010, Proceedings, LNCS 6351, pp. 126–141, Springer, http://www.springerlink.com/content/7hv48n2h38128363/, September 2010.

[RD14] Dubravka Ilić, Timo Latvala, Kimmo Varpaaniemi, Pauli Väisänen, Elena Troubitsyna, and Linas Laibinis. *Evaluating a Control System Architecture Based on a Formally Derived AOCS Model*. In *DASIA 2010: DAta Systems In Aerospace, Budapest, Hungary, June 1–4, 2010, Proceedings*, ESA SP-682, European Space Agency, June 2010.

[RD15] Ilya Lopatkin, Alexei Iliasov, and Alexander Romanovsky. *Rigorous Development of Dependable Systems using Fault Tolerance Views*. In *Proceedings of ISSRE 2011, 22nd Annual International Symposium on Software Reliability Engineering, Hiroshima, Japan, November 29 – December 2, 2011*, IEEE, http://deploy-eprints.ecs.soton.ac.uk/343/, December 2011.

[RD16] Ilya Lopatkin, Alexei Iliasov, Alexander Romanovsky, Yuliya Prokhorova, and Elena Troubitsyna. *Patterns for Representing FMEA in Formal Specification of Control Systems*. In *Proceedings of HASE 2011, 13th IEEE International High Assurance Systems Engineering Symposium, Boca Raton, FL, USA, November 10–12, 2011*, IEEE, http://deploy-eprints.ecs.soton.ac.uk/347/, November 2011.

[RD17] NuSMV project. *NuSMV: a New Symbolic Model Checker*. http://nusmv.fbk.eu/, January 2012.

[RD18] ProB project. *The ProB Animator and Model Checker*. http://www.stups.uni-duesseldorf.de/ProB/index.php5/Main_Page, January 2012.

[RD19] Yuliya Prokhorova, Elena Troubitsyna, Linas Laibinis, Kimmo Varpaaniemi, and Timo Latvala. *Deriving Mode Logic for Fault-Tolerant Control Systems*. In *Proceedings of NODES 2011, 5th Nordic Workshop on Dependability and Security, Copenhagen, June 27–28, 2011*, Technical Report of Technical University of Denmark, June 2011.

[RD20] Yuliya Prokhorova, Elena Troubitsyna, Linas Laibinis, Kimmo Varpaaniemi, and Timo Latvala. *Derivation and Formal Verification of a Mode Logic for Layered Control Systems*. In *Proceedings of APSEC 2011, 18th Asia Pacific Software Engineering Conference, Ho Chi Minh City, Vietnam, December 5–8, 2011*, IEEE, December 2011.

[RD21] Tuomas Räsänen and Laura Nummila. *DEPLOY Training Evaluation Document*. http://deploy-eprints.ecs.soton.ac.uk/314/, September 2010.

[RD22] UPPAAL project. *UPPAAL*. http://www.uppaal.org/, January 2012.

[RD23] Kimmo Varpaaniemi. *DEPLOY Work Package 3 Software Requirements Document for a Distributed System for Attitude and Orbit Control for a Single Spacecraft*. DEP-RP-SSF-R-006, Issue 1.3, http://deploy-eprints.ecs.soton.ac.uk/309/, October 2011.

[RD24] Kimmo Varpaaniemi. *Event-B Projects DSAOCSSv002 and DSAOCSSV003 with Special Files for ProB Classic*, http://deploy-eprints.ecs.soton.ac.uk/331/, October 2011.

[RD25] Kimmo Varpaaniemi. *Some NuSMV Experiments on the Mode Synchronization Protocol in DSAOCSS*, http://deploy-eprints.ecs.soton.ac.uk/362/, January 2012.

[RD26] Kimmo Varpaaniemi. Rodin platform bug tracker items still open in January 2012: 3151805, 3154319, 3206302 and 3385671. http://sourceforge.net/tracker/?atid=651669&group_id=108850&func=browse.

[RD27] Kimmo Varpaaniemi. Rodin platform feature request tracker items still open in January 2012: 2060085, 2075318, 2130277, 2392103, 3152959, 3155514 and 3417899. http://sourceforge.net/tracker/?atid=651672&group_id=108850&func=browse.

[RD28] Kimmo Varpaaniemi. ProB defect report tickets still active in January 2012: #102, #184 and #185. http://cobra.cs.uni-duesseldorf.de/trac/report.

5. The Business Information Sector

In this chapter we provide the final feedback by SAP on the DEPLOY work for the business information sector. This is largely based on the experience from the second part of the project, such that the latest developments and improvements of the platform are taken into account. The necessary concepts and tooling we created in this second part of the project have been described in Deliverable D4.2 [D42]. Similar to Deliverable D29 [D29] which reports on the assessment of the initial pilot deployment (for the first half of the project), in this document we will provide the final assessment focussing on the enhanced pilot deployment.

The chapter is structured as follows. The question *What are the strategies on deployment in the Business Sector?* is discussed in Section 5.1 Section 5.2 continues by describing the concrete tooling we created during the enhanced deployment in order to answer the question *Which artefacts have you developed?*. Section 5.3 describes our effort in evaluating the enhanced deployment in an industrial setting and answers the question *What deployment have you achieved?* The following Section 5.4 discusses *How well is the enhanced pilot adopted?*. In Section 5.5 we describe *What is the assessment of the Event-B method?* and *Which advances of the methods have been useful and in which way?* Finally, in Section 5.6 we explain *our view on the tool platform*. A discussion regarding *the role of training material and its quality* is left out, since the deployment strategy we followed (hiding the formalism from users) did not rely on training material of the DEPLOY project.

5.1. Introduction

Business software consists of any software which helps companies improve their business. In contrast to typical areas of application of Formal Methods, this area has some, e.g. safety critical, aspects which have been significant incentives to make the use of Formal Methods attractive, as e.g. in the transportation industry. On the other hand business software is often highly mission critical. As a result, customers expect high qualitative software which is efficiently developed — goals, which Formal Methods could be helpful to achieve. Nevertheless Formal Methods are (to our knowledge) not used routinely in the development of business software.

In this deliverable we are reporting on the evaluation of our activities towards a successful deployment of Formal Methods developed in the EU-FP7 project DEPLOY in the business sector. As there are specialised established domain-specific languages for the development of this software in place, switching to non-domain-specific languages and working with mathematical syntax is typically considered not to be feasible.

Our approach throughout the project therefore was to make business application developers benefit from formal verification techniques by hiding the mathematical formalisms, such as Event-B, behind the languages normally used in business software development and to rely on the high automation of today's verification tools, so that developers do not have to directly interact with the formalism.

Initial Pilot

For the initial pilot deployment, reported in deliverable D4.1 [D29] we focussed on service choreography modelling and integration testing. By providing a domain-specific modelling approach (MCM), including automatic transformation to Event-B and automatic verification and test generation utilizing the Rodin platform, we were able to demonstrate the feasibility of hiding

advanced formal verification techniques from software developers.

In order to demonstrate how to apply these techniques on the implementation level as well, we decided to enhance the described solution with the capability to check consistency between our domain-specific language (DSL) and the already integrated architectural artefacts, thus guiding, and driving the implementation process of business software.

Enhanced Pilot

After concluding the work on the pilot for service choreography, we went into an internal evaluation. The aim was to identify the options for an enhanced deployment [D29]. After being able to convince our stakeholders about the general applicability of utilizing Formal Methods in the business software development process, the focus was on identifying the area with the highest potential for a broad productive use.

During the discussions it became very clear that formal techniques should be based on already adopted modelling languages to increase acceptance and lower the need for learning. Furthermore, the automatic test generation based on design artefacts turned out to be the key factor for creating interest in adopting our concepts in the development organization.

As process modelling is by far the most common and mature way of describing the overall scope of business software, we decided to concentrate on the system level for the enhanced pilot deployment instead of extending the previous work on the comparatively lower abstract service layer. This choice was further encouraged by the fact that system-level testing activities are in fact most decisive for business software. This is due to the fact that companies are usually able to correct inconsistent data due to faulty software, but are highly dependent on being capable to execute their standard processes.

Our aim was further to leverage not only our experience from the first piloting phase, but also concrete concepts and implementations where applicable. For the modelling of business processes we again targeted an automated model transformation and verification approach in order to hide complexity from the user, which allowed us to reuse our previous work on formal property checking (e.g. deadlock freedom), and further enhance the approach (e.g. by data consistency).

Also, on the testing side we were able to incorporate much of the previous work by continuing to utilise the Event-B transformation for creating test cases with ProB. This gave us more freedom to work on the maturity of the general testing framework and to punctually improve its building blocks. As the dependence on a specific tool provider was perceived as a major obstacle for adopting the solutions of the first piloting phase, we put our major effort into the generalization of the concepts. As reported in Deliverable D4.2 [D42] we designed a phased model transformation approach that allowed us to choose between different test generators in a unified way. Furthermore, we enhanced the existing test optimization and integrated our solution into the testing framework at SAP.

5.2. Conducted Work

In order to realize the envisioned deployment, we integrated various components into a productively used testing framework at SAP. In Figure 5.1 the main blocks of this framework including our components are presented. The *Test Environment* offers UI-based keyword-driven testing capabilities through a *Scenario Editor*, which allows to assemble captured test scripts and

to visualize the generated executable scenarios (obtained from test cases). The scripts can be recorded through the *Script Recorder* component, which is connected to the System under Test (SUT) for this purpose. Besides the capturing of user interactions on the SUT, the *Script Recorder* offers replay functionality, which is also utilised for the stepwise execution of scenarios. Together with the *SUT* and the *Backend Repository* it assembles the original setup.

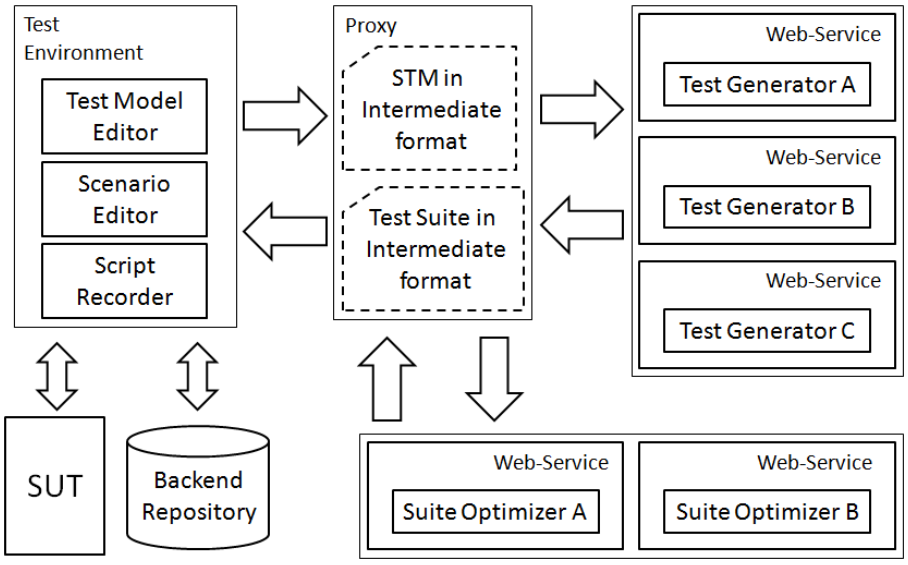


Figure 5.1. Architecture of the MBT environment

We extended the test environment by creating and integrating the *Test Model Editor*, which allows the creation and editing of process-based test models. It further enables the triggering of the test generation and visualization of the resulting test suite.

In order to mitigate the risk of dependency from one single test generation technology, our goal was to integrate multiple tools and vendors, which we achieved by providing transformations from process-based test models (TM) to abstract state transition machines (STM), which can be transformed further into vendor-specific input formats in a straight forward manner. Due to the lack of an existing intermediate format, we created a proprietary STM. However we published its concepts and are active in contributing its concepts to the various emerging standardization initiatives. A proxy is set up for routing the test generation requests in order to obtain a single communication partner, which allows to add and update generator components without additional configuration of the test environment.

General-purpose MBT tools rely on varying strategies to reduce the large initial test suites they produce during test generation. Therefore we decided to offer unified test suite optimization independent of the chosen test generator. This further allows us to consider custom requirements for the enterprise software domain. The different optimization procedures are wrapped in another set of web-services and can be used in the following way. After the test generation succeeded, the resulting traces are transformed into an intermediate test suite format and sent to the proxy component, which forwards it to an appropriate *Suite Optimizer*.

The test reduction is implemented on the intermediate format for test suites. Therefore a further transformation of the results in the *Suite Optimizer* is not necessary. The proxy takes the reduced test suite and routes it back to the *Test Model Editor* where it will be used to create the concrete test suite, containing executable scenarios.

Besides being able to seamlessly integrating the *Test Model Editor*, detaching the test environment from the test generation services brings the following advantages:

- Re-Use: Utilizing a generic input and output format, we are able to hide the complexity of the specific model transformations into the input format of concrete test generators, thus making MBT accessible as a service to other potential test environments.
- Performance: Decoupling computational expensive functionality like test generation and test suite reduction promises better system performance and does not block front-end users. Replication of the web-services and the introduction of load balancing to the proxy further increases scalability.
- Maintenance: The service-based decoupling in combination with a proxy further allows to maintain and upgrade test generation components in a non-persuasive way.

5.3. Deployment Evaluation

After prototyping, creation and integration of the components described in the previous section, we consulted one of SAP's major product areas in order to negotiate an evaluation strategy. It was agreed to set up a case study with 7 development teams, which were asked to apply model-based testing (MBT) in their scenario testing activities for a specific internal release. During their activities, the team members where asked to collect requirements and report bugs they encountered. After concluding the case study further interview session were carried out with the participants, in order to get their overall assessment of the tool as well as information about their productivity and the experienced learning effort. As information about product and development activities have to be handled with great discretion (especially in the case of quality-related information), we will report in a more general way about the findings of the case study. However, we feel that this is also in the interest of the reader as it abstracts from the product and company specific context.

Case Study Participants

The participants did not have a background in Formal Methods but knew the basic concepts of business process modelling and were familiar with the concrete business process they wanted to cover with different scenario tests. They were further trained and experienced to use the proprietary testing environment, but did not have any knowledge about MBT. In the beginning of the case study, they received a 2 hour workshop on the additional modelling and testing concepts, necessary to understand and operate our tool-extension as well as additional documentation and guiding samples. Further all participants could rely on remote expert support for any tooling, technology or process related question. On average these support activities aggregated to about one additional hour per participant.

Requirements Analysis

Over the course of the case study, the participants collected 47 different requirements and prioritized them regarding their importance from 1 (absolute showstopper) to 5 (nice to have). In

the 2 months period of evaluation we were able to incorporate all requirements of priority 1 and 2 and most of priority 3. The remaining requirements mainly concerned the automation of addition steps in the test generation process, which do not directly relate to the test generation and had been manual steps in the original process as well (e.g. the linking of created test cases with test plans) or even addressed issues in the original test environment. Overall, only one requirement concerned the enhancement of the test generation functionality, while the remaining mainly dealt with usability issues.

Interview Sessions

Each participant was interviewed after the case study. All stated that the maturity of the tool improved dramatically during the evaluation phase and agreed on the verdict that both tool and testing process is mature enough for a broad use inside the organization. Further it was confirmed that the learning effort was very limited and the approach quite intuitive. As expected, the usability of the test model editor still left room for improvement, but was comparable to other internally used tools. It was a common observation that the MBT approach demanded greater care in the script recording and test data definition activities. However this was generally perceived as a positive side-effect.

Conclusion

Based on the requirements analysis and interview sessions, sufficient confidence has been gained to go for a phased roll-out of model-based scenario testing to the whole product area. This roll-out will be accompanied by the hand-over of responsibility for the maintenance and further improvement of the tooling that was created in the context of DEPLOY from our research unit to an operations team.

5.4. Level of Adaptation

As described in the previous section, the results of DEPLOY are going to be used productively in the future. Since we based all modelling on existing domain-specific model types and translations to Event-B, developers are able to create and maintain modelling content without a deep understanding of Formal Methods. The Event-B language as well as the utilised concepts and tools embedded in Rodin are completely hidden from the users.

In this way we hope that a large group of developers will be able to make use of Formal Methods in the future. On the other hand the chosen deployment approach demands highly skilled maintenance workers which are not only deeply familiar with the Formal Methods used but also with the technical framework hiding it. At the moment this special maintenance is still taken by researchers and it remains unclear whether it will be possible to hand it over to development in the future as well.

Nevertheless we achieved to build up confidence in the development community that Formal Methods can be used routinely and beneficially in the development process. Thus, we feel that model-based testing could be a kind of a eye-opener as it showcases some of the many advantages of using Formal Methods. Further, we are currently in promising negotiations with sales representatives and consultants in order to make our tools available as product offerings to SAP partners and customers.

5.5. Assessment of Event-B Method

The Event-B language can be used comfortably to describe event-triggered concurrent models (such as message choreography models) and state machine based models (such as implementation models for business objects). This is because each activity, as represented as an Event-B event, is independent of other activities, and only controlled by environmental stimuli (like messages) or control states, which can be effortlessly modelled in event guards. On the contrary, when it comes to sequential models like business processes, extra care has to be taken for a token-based mechanism to express sequentiality, especially when multiple instances of a same activity can exist and run in parallel. This does not only increase modelling complexity, but also put additional burdens on model verification. With the Flow plug-in, sequential properties (not models) can be intuitively expressed in graphic terms. However, it is yet to be evaluated whether the plug-in can reduce the difficulty of verification.

An advantage of Event-B is its refinement paradigm, which provides a reasonably easy way to formally capture relationships of models at different abstract levels. This is important because, using refinement, we are able to prove that models are consistent with each other throughout the whole development cycle. However, from our practical experiences, while it is relatively easy to transform model consistencies even automatically into formal refinement specifications, it is usually very difficult to validate such refinements using either theorem proving or model checking techniques. Without exceptions, a large amount of manual efforts is indispensable to figure out additional and often hidden relationships between models in order to increase provability. Although we see potential solutions to automatically discover such auxiliary relations, there is still a huge gap between what we can model using Event-B, and what we can prove using the Formal Methods that support Event-B.

Event-B comes with powerful automated theorem provers that can handle integers quite easily. Various auto-/post-tactics can be selected and combined to automatically discharge a large number of proof obligations. A recent introduction of the Relevance Filter plug-in uses heuristics to select relevant hypotheses in each proving step. Using this plug-in, we witnessed a high increase in the number of automatic PO discharges for some of the models that we experimented with. While all of these are quite promising, we are still in great expectation for more powerful automated provers and for better user supports in manual proofs. The integration of the Isabelle prover is deemed an important step towards this goal.

Different decompositions are available to reduce the complexity of modelling large software models that can be naturally divided into separate sub-components. These decomposition techniques are supported by plug-ins, which allow the enforcement of certain relations before and after decompositions. However, all decomposition techniques have their limits and are suitable only for certain scenarios. For instance, shared-event and shared-variable decompositions support only top-down development. For a service-oriented world where service compositions and orchestrations make up a large portion of software development, bottom-up design is very common but only supported by modularisation. In fact, any sophisticated software development involves both top-down and bottom-up designs, and using one and only decomposition technique is unlikely to suffice. Combining all decomposition techniques in one model development will be an interesting topic to explore.

Business software models use strings and structured data types very commonly, which can be better supported now using the Theory plug-in. Collaborative modelling started to get attention, and the Teamwork plug-in is a welcomed first step. However, our evaluation shows that it is not mature enough to be applied in real life model development. We would especially like to see

support for collaborative proving.

5.6. Assessment of Rodin

Rodin is a well developed software environment having different plug-ins (ProB, AtelierB-Provers, Anim-B, UML-B, etc.). It provides a good support for Event-B language. The tool is sufficiently well documented and has an intuitive and user-friendly interface.

Even though Rodin is a well-developed, stable and simple-to-use application, the large footprint of the Eclipse framework hinders its deployment inside of a web-service. Fortunately the ProB plug-in was also made available as a standalone tool, so that we could realize our envisioned client-server architecture for a seamless integration into the currently used test framework, as described in the previous sections.

Another problem is due to the incompatibilities among plug-ins. For a realistic development, it is necessary to take advantage of the power of several plug-ins. This is however not supported by Rodin and its plug-ins. There is also no framework or methodology to ensure plug-in compatibilities. For instance, we need modularisation to decompose a software model into several sub-component, each having its public interface and private interface hidden from other components. Then, when we use ProB to check the model, ProB is unaware of the structure imposed by modularisation, and thus fails.

References

[D29] Andreas Roth, Vitaly Kozyura, Wei Wei, and Sebastian Wieczorek. DEPLOY Deliverable D29: Initial Assessment results. Technical report, FP7-DEPLOY Project, EU, 2010. published at http://www.deploy-project.eu/.

[D21] Andreas Roth, Sebastian Wieczorek, Vitaly Kozyura, Wei Wei, and Sebastian Wieczorek. DEPLOY Deliverable D4.1 D21: Report on Pilot Deployment in Business Information Sector. Technical report, FP7-DEPLOY Project, EU, 2010. published at http:// www.deploy-project.eu/.

[D42] Sebastian Wieczorek, Vitaly Kozyura, Wei Wei, and Andreas Roth. DEPLOY Deliverable D4.2 D42: Report on Enhanced Deployment in Business Information Sector. Technical report, FP7-DEPLOY Project, EU, 2011. published at http://www.deploy-project.eu/.