Project DEPLOY
Grant Agreement 214158

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

DEPLOY Deliverable D31

D6.2 Methods: Tackling Industry Specific Challenges

Public Document

August 30, 2010

http://www.deploy-project.eu

Contributors:

Michael Butler
John Fitzgerald
Michael Jastram
Cliff Jones
Linas Laibinis
Manuel Mazzara
Abdolbaghi Rezazadeh
Kaisa Sere

Reviewers:

Rainer Gmehlich	Robert Bosch GmbH
Michael Jastram	University of Düsseldorf

Contents

Chapter 1

Setting the Scene

The work on "methods" in the DEPLOY project is undertaken in three work packages:

- WP6 is concerned with requirements
- WP7 addresses reuse
- WP8 focuses on dependability issues

The first report on methods (D15 or D6.2) was rather long and the reviewers urged the project to produce shorter reports that point to other published documents for detailed explanation. We have tried to respond to that request this time.

The purpose of the current report was stated in the "Description of Work" (DoW) as:

> D6.2: this intermediate deliverable will report on the needs of the industrial partners and how the research underway aims to meet those needs. Cross reference to those research ideas which are by then being worked on in WP9 [Tooling] will be included.

The "Re-focus" activity in the project was a planned step to ensure that the "academic partners" (APs) understood the evolving needs of the "deployment partners" (DPs). This activity culminated in a three day meeting in May 2009 and its outcome was reviewed at the interim review in October 2009. The largest single request from the DPs at the refocus meeting was for support for code generation.

It has been clearly stated at each review that the methodological problems coming from the applications being tackled by the DPs are "challenging". This is, of course, not surprising: each DP is experienced in systems and software development and has its own well-honed engineering practices mostly

with their own tool support. Furthermore, their core businesses are totally disjoint.

Current work in the three main work packages is described below in Chapters 2–4. As a reminder of the fact that the DEPLOY project is about the use of formal methods for the design and creation of *dependable* systems, the obvious numeric order of the three work packages has been reversed. The psychological problem of discussing dependable systems in terms of the standard [ALRL04] fault/error/failure language[1] is commented on in Chapter 2: although these issues occupy a major part in the design of any major system, DPs in DEPLOY tend to prefer terms such as "exceptions".

There are three general issues that are worth addressing before the reader turns to Chapters 2–4.

The final methods deliverable The DoW envisaged that the final (M48) deliverable on methods would cover:

> D6.3: Final research results and experience of using them in the industrial sectors — this deliverable will describe the research done in DEPLOY and will emphasize its use by the industrial partners. (M48)

It was agreed at the March 2009 review that it will be more important to deliver a document which sets out the methods actually used in the DEPLOY project in a way that will enable other potential users to benefit. Abrial's book [Abr10] is finally published and this clearly sets out the core Event-B method applying it to textbook scale examples. Chapter 5 sets out our current plans for D6.3 (D44) and points out that this is looking to be a pretty daunting task.

General method issues Some of the method issues that have come up from the applications being undertaken by the DPs do not naturally fit under any of the three method WPs. To take one illustration, Bosch need to express forms of refinement that are not natural to present in Event-B. This is not to say that they cannot be described, but the description often looks more like an encoding than a direct description. One should remember that Event-B does not offer a direct way of stating any program decomposition rules: it is necessary to express even sequencing of events by adding auxiliary variables that can be used in event guards to define sequencing. This technique works and most of the generated proof obligations are trivially discharged but it

[1] An attempt to put this terminology in a more formal framework is given in [Jon03].

does still look like an encoding of the actual design decisions. Many other examples could be listed but the general issue is that having "event refinement" as the only notion of development does sometimes lead to encodings that complicate the expression of a designer's intuition. This could, for example, result in treatments of real-time extensions that become cumbersome.

Whilst on the topic of methods in general, it is worth reiterating that the recent ACM Computing Surveys article [WLBF09] presents an authoritative and up-to-date survey of formal methods deployment. Furthermore, this will be kept current on a web site (URL to be given soon).

Constraints of the tools/philosophy Strongly related to the general methods issues is the impact of building on the Rodin-Tools; these can be fairly said to embody the core philosophy of Abrial's Event-B approach. One could add "for better or worse": it is a fundamental commitment of the DEPLOY project that tool support is seen as crucial to the adoption of formal methods. It does however mean that the path from wanting to try a new idea in the methods area can be constrained by the ability to get the tools modified. We have in fact ameliorated this problem by using the "plug-in" approach: in some cases, it is possible to put what amounts to a language extension into a plug-in.

One example where a work-around cost a DP quite a lot of time is that of records. Records –as in for example VDM– were not supported in the original Rodin tools. Stefan Hallerstede proposed an ingenious way of presenting the types and properties of constructor and selector functions but the proof obligations were rather opaque and troublesome. A plug-in has been developed but the first version featured "open" records (more akin to Z schemas) and the extension to cover "closed" records is only now becoming available. A replacement for this will come via the "Mathematical Extensions". With limited resources, it is not possible to be fully responsive to requests from DPs and –it must be added– the resource needed for "code generation" exacerbates this situation.

Another issue is that of performance. The DPs are naturally generating far larger models than in the Event-B teaching material and tool performance is seen as an inhibitor. The Düsseldorf group has invested a lot of effort to improve the performance of the model exploration tools. ETH is working on bridges to other theorem proving systems.[2]

[2]Longer term, a new UK funded project AI4FM (see www.AI4FM.org) is looking into learning from how experts tackle proofs but this project has only just started (a number of position papers have been published [BGJ09a, BGJ09b]).

Chapter 2

WP8: Dependability

Our work on achieving and demonstrating dependability, concentrated in Work Package 8, aims to provide advanced modelling techniques that allow resilience, security, reliability and safety to be incorporated into the development process. Although this necessarily involves research on the principles and foundations of modelling technology and Event-B, we have sought, where appropriate, to work with the deployment partners to provide techniques that can be used in the pilot deployments and beyond.

Work in Task 8.1, reported on Section 2.1, has focused on supporting the process of deriving specifications of resilient systems from informally expressed requirements. A notable success here has been in helping Bosch to link formal modelling techniques with the already established Problem Frames approach.

Task 8.2 (Section 2.2) addresses the formal modelling of resilience aspects. Up to 40% of requirements in the pilot and mini-pilot studies relate to resilience aspects (exceptional and faulty behaviour, especially of elements of the system environment, tolerance and recovery mechanisms). However, these requirements are typically not gathered, presented or structured well for formal modelling. In the same task, work with SAP has led to several publications addressing the Event-B modelling of faults and (time-bounded) inconsistencies in business processes. This is now being linked to automatic generation of Event-B models drived from BPMN process descriptions.

Progress in Task 8.3 (Section 2.3) has advanced the refinement-based modelling of security mechanisms in Event-B. Building on examples from SAP, models for access control have been developed, as have relatively general techniques for developing security protocols by refinement and for specifying and reasoning over ignorance properties.

Task 8.4 (reported in Section 2.4) addresses the modelling, validation and refinement of combined logical and stochastic models. Although this work

is longer-term in nature and is some way from routine industry deployment, it is being used to tackle problems of reliability assessment in the transport domain (WP2) and for the validation of properties of wear levelling in flash filestores, an industry-defined challenge problem within the Verified Software Grand Challenge.

Task 8.5 (Section 2.5), addressing real-time, is new, having been added to the project following the refocus. Activities address a range of aspects of modelling real-time properties, including tasking models and scheduling, modelling software process structures, and linking to external tools for analysis.

2.1 Specifying Fault Tolerance

The objective of Task 8.1 is to develop methods for deriving specifications of resilient systems, starting from informal requirements that are generally expressed in natural language. This original set of requirements is generally elicited from users, customers or other stakeholders. Its informality is both a strength and a weakness. It is a strength because it facilitates flexible communication between stakeholders; but it is also a weakness because consistency and redundancy checking are labour-intensive and error prone, for example. Also, it does not support easy verification that the final system is really a reflection of what was elicited in the early stages – identifying this sort of problem at later stages is indeed much more costly. This has always been true, even for "normal" desktop applications but it is even more important when resilience is an issue and applications are "critical" (real time applications, or applications involving human or animal lives). The path from informal to formal specification (passing through "semi-formal" like UML or Problem Frames) is difficult enough in itself. It is even more challenging when fault tolerance and thus degraded behaviour modes have to be taken into account. Methods developed by Task 8.1 should therefore ideally support each stage from requirements elicitation to assumption identification and from degraded behaviour layering to a final specification.

Progress has been made both on methodology and on its application to case studies. Methodological issues are presented in [Maz09b]. In [Maz09a] the specification of a simple train system is exploited as a case study to show the progress made on the method side. The process from informal/semi-formal to formal, and the structuring of specifications for expressing abnormal/degraded behaviours has been investigated from different points of view and following alternative approaches. An overview of the problems (many still open) encountered using Problem Frames for semi-formal specifications

can be found in [Jon10a]. The work done in [DM09] approached a different case study taken from Business Process Modelling to identify key issues in specifying fault tolerance and disambiguating informal specifications. The domain is very specific, but the conclusions have a wider applicability. The problem of requirements and the use of formalisms for their disambiguation has been also discussed in [MB10].

Work in Task 8.1 has proceeded in close collaboration with Bosch as witnessed by deliverables D19 [DEP10a] and D15 [DEP09b]. The experiments on the use of Problem Frames for semi-formal specification of the Cruise Control led to an effective methodology to cope with informal requirements in natural language. Already in the early stages of the project, Bosch has been able to detect inconsistency and redundancy in the original requirements and reorganise them using the Problem Frames approach. The collaboration with Bosch inspired many of the results from this task, and the close cooperation was mutually beneficial.

The main challenge in Task 8.1 is represented by the holistic aim of developing a method for deriving specifications of resilient systems. This idea influenced our investigation, and our ambition to have a broader impact distracted us from focusing on the relevant details. We now think it will be better to approach more tractable subproblems, each of those separately, and then study the possible interactions between them. A reasonable starting point would be investigating discrete time systems, where fault tolerance is initially not considered, and then extending the results to more general cases.

2.2 Resilience

Task 8.2 focuses on achieving and demonstrating dependability by developing modelling and analysis patterns to support the correct application of dependability mechanisms during system development. Consequently, our work includes understanding the need for modelling faults and fault tolerance or recovery mechanisms in the deployment partners' domains[1] as well as on developing Event-B patterns for describing these features formally.

[1]Sometimes system stakeholders are reluctant to use the term "fault", often referring instead to abnormal or exceptional situations. We use the term in its technical sense of referring to causes of error states, which are deviations of internal system state from normative specifications.

2.2.1 Resilience Requirements in Pilot and Minipilot Studies

We have analysed the informal requirements descriptions produced by deployment partners for the pilot and mini-pilot studies in Deploy[2], focusing on the ways they describe error recovery (including fault tolerance and fault assumptions). These aspects always represent a substantial proportion of the requirements (sometimes as much as 35-40%). The major source of faults considered in these systems is the environment, including sensors, external networks and operators. Dealing with software design faults is never stated as a requirement, and only rarely do requirements define hardware faults (e.g. node crashes in a distributed application) and state how these need to be addressed.

The system requirements reviewed include descriptions of degraded functionality, the most typical example being system safestop. More generally we observe that the requirements always include information about how general system behaviour is affected by various abnormal situations. Unfortunately this information is rarely explicitly stated as a priority requirement (sometimes, we needed to deduce this information from other requirements).

We have found that nearly all system requirements use the concept of *operational modes* [DIRR09, IRD09] to refer to different operational conditions resulting in different functionalities provided by the system. As a consequence, descriptions of system modes and mode transitions are often intertwined with error recovery; sometimes this includes fault handling by system degradation.

A final observation is that requirements related to error recovery are not structured in a way that makes it easy for modellers to work with these issues. The relevant requirements are typically scattered over the whole requirements document and refer to issues related to different levels of abstractions. For example, none of the documents reviewed had a table of fault assumptions.

2.2.2 Modelling Resilience

Task T8.2 has included work on the resilience and consistency of business processes, using the SAP mini-pilot and pilot studies. We have considered two forms of inconsistency. The first, semantic inconsistency, is application-dependent. The second, which we term *temporary inconsistency* is introduced through an exceptional situation such as the cancellation of an order

[2]The detailed requirements documents for the case studies are largely confidential, but descriptions of the pilots are provided in public deliverables for the deployment workpackages [DEP09a, DEP10a, DEP10b, DEP10c].

and may be tolerated for a bounded time period. A pattern has been developed for modelling eventual recovery of business processes from temporary inconsistency [BFRR10]. Derived from an existing Event-B pattern for modelling time [CMR07], our new pattern models the on-time and late arrival of messages and allows for application-specific compensation for late arrival, restoring the semantic consistency of the business process.

We have designed an algorithmic translation from the Business Process Modelling Notation BPMN to Event-B [BW10]. This covers most of the commonly used BPMN features and is structurally faithful in the sense that a new machine is added to the Event-B refinement hierarchy for both the control flow and the data information of each partner in the BPMN model. This is also the case for sub-processes within a partner. A final refinement layer adds the inter-partner messages. This achieves a good level of automatic proof and localises the cause of failed proof obligations. A further feature of the work is that we model the possibility of multiple concurrent instances of the same process.

One of the possible benefits of translation from BPMN to Event-B is that a developer may annotate a BPMN diagram with application-level consistency conditions which could then be translated into Event-B and proved within the model. This possibility has been explored [BW10], although much remains to be done to make the resulting proof obligations amenable to automatic proof.

An important challenge is fine-tuning the translation to generate appropriate invariants to maximise the proportion of automatically discharged proof obligations. The challenge is increased if the designer is allowed to add application-level consistency conditions to a model. Work on this is ongoing. We also face the challenge of presenting the BPMN developer with information from failed proof attempts in an understandable way.

Other work in Task 8.2 focuses on fault tolerance (FT) modelling and its reuse [LIR10]. Based on our previous analysis of fault tolerance related requirements within the Deploy industrial partners and traces in Event-B models, we progressed with an approach to systematic integration and reuse of fault tolerance modelling. The main points of our approach here are: 1) targeting FT concerns using a separate viewpoint; 2) system abstractions as FT views; 3) iterative process of FT refinement alongside an Event-B refinement chain via detailisation templates 4) ongoing work on a transformation mechanism for Event-B models for further integration of FT modelling and 5) reuse of FT modelling decisions via libraries of templates and patterns.

Recent work focuses on two dependability means: rigorous design and fault tolerance [DIRR09, IRD09]. In order to manage the potential complexity of models of fault tolerant systems, the notion of operation mode is

introduced. Our approach formalises this notion and shows its relation to a state-based formalism using a refinement approach. We further demonstrate how to use the notions of modes and mode transitions for developing fault tolerant systems. Our results suggest that using modes in state-based modelling allows us to improve system structuring, the elicitation of system assumptions and expected functionality, as well as requirement traceability.

Work has been undertaken on the resilience of complex embedded systems, using layered control systems in space satellites as an application area [ITL$^+$10c, ITL$^+$10a]. The dynamic behaviour of such systems is typically described in terms of operational modes that correspond to the different stages of a mission and states of the components. These components are susceptible to various faults that complicate the mode transition scheme. It is often the case that to execute error recovery the system should roll back to some preceding, and usually degraded, mode. Correspondingly, all system components should be brought to that mode. Since mode changes do not occur instantaneously but rather take some time, new errors might occur during the error recovery. Such complicated mode transition schemes are prone to cascading mode transition effects and are notoriously difficult to develop and verify. Yet the success of a mission depends on the correct implementation of mode changes. Another typical problem associated with mode-rich layered systems is to ensure mode consistency of the components residing at different layers, i.e., to correctly define the mode logic and guarantee that the system faithfully implements it. A formal approach has been proposed that ensures consistency of mode changes while developing a system architecture by refinement [ITL$^+$10c]. It relies on recursive application of modelling and refinement patterns that enforce correctness while implementing modes and mode transitions. The proposed approach is exemplified by the development of an Attitude and Orbit Control System (AOCS) [DEP10b, ILT10] conducted within WP3.

2.3 Security

In Task 8.3 we pursue research on security in several directions: specifying access control systems, developing security protocols by refinement and integrating the notion of "ignorance sensitive specification" into Event-B.

2.3.1 Specifying Access Control Systems

We have proposed an approach for developing access control systems by specifying separately the insecure target system and the security authorisa-

tion [HBA09]. The two sub-systems are then combined using "Shared Event Composition" [Sil]. Temporal safety properties of systems are specified by abstract models and verified by proving that the combined system refines such abstract models.

We apply our approach to an example from SAP for compliance modelling. There are two different types of business objects: *price specification* and *sale order*, where a sale order depends on a particular price specification. Services on these business objects includes *creating*, *deleting* and *modifying*. Access control policy specifies that users can call a service on a particular business object depending on certain assigned context. Moreover, certain actions on business objects are declared to conflict with each other and should not be carried out by the same user.

Initial investigation confirms the applicability of our approach to this type of system, in particular, the separation of concerns when developing functional aspects of the insecure system and the security features specifying access control policy.

In the next steps, we plan to verify certain safety properties of the system, concentrating on the automatic generation of the abstract models corresponding to these properties. Moreover, the use of *shared event composition* means the states of two sub-systems (i.e the insecure system and the security authorisation system) need to be disjoint. This turns out to be too restricted in some situations and we plan to apply more general techniques such as *patterns* [HFA09]. Furthermore, we need to see the scalability of the approach on larger examples.

2.3.2 Developing Security Protocols by Refinement

We propose a development method for security protocols based on stepwise refinement. Our four-level refinement strategy guides the transformation of abstract security goals (Level 0) into protocols that are secure when operating over an insecure channel controlled by a Dolev-Yao-style intruder (Level 3). The intermediate refinement steps successively introduce local states (Level 1), communication channels with security properties and an intruder (Level 2), and cryptographic operations realising these channels (Level 3). The abstractions used provide insights on how the protocols work and foster the development of entire families of protocols sharing a common structure and properties. In contrast to post-hoc verification methods, protocols are developed together with their correctness proofs. We have implemented our method in Isabelle/HOL and used it to develop several entity authentication and key transport protocols.

We can report on progress on all fronts: we have improved our refinement

strategy and protocol modelling technique, we have extended existing and added new case studies covering additional protocol features, and we have extended our theory of refinement. Regarding the refinement strategy, we have defined simpler and more general initial models for the security goals. In particular, we have defined abstract protocol-independent initial models for secrecy as well as for weak and strong authentication. We have developed a more concise representation of the intruder events that fake messages on the channels at later levels, allowing the refinement of the intruder to rely on lemmas of a standard, recurring form. We have extended the case studies of server-based key transport to cover a range of classical protocols all derived form a common Level 1 model. These include Abadi-Needham's version of Otway-Rees, Yahalom, Needham-Schroeder, Denning-Sacco and Kerberos IV and V (the last two in progress).

Finally, we have extended our theory of refinement with a notion of observation (cf. [Abr10]). This allows us to prove theorems about the soundness of refinement as a method to establish the inclusion of observable traces and the preservation of already established properties by subsequent refinements. In our view, these properties are indispensable for a clear semantics of the refinement process.

More details about this work can be found in [SB09].

2.3.3 Ignorance Sensitive Specification in Event-B

We started our investigation on extending the underlying logic of Event-B to a logic of "knowledge" as described in the work of McIver and Morgan [Mor09, MM09, Mor06]: a modal operator is added for expressing *knowledge* about some hidden "high-security" variables. Our main aim is to start the development of *secured Event-B*, an extension of Event-B which facilitates the reasoning about protocols with ignorance properties, e.g. that an observer should not know more information about certain hidden facts than is allowed. In the first step, we introduce a notion of *security machine invariants* expressing security properties of systems and prove that our abstract systems guarantee these properties.

An attractive aspect for this line of work is that the operational semantics of the model can be "translated" to ordinary first-order logic, hence this could be introduced into Event-B without needing additional tool support. Moreover, the reasoning about traditional functional properties can be separated naturally from the reasoning about additional security properties: the security properties can be "encoded" as a refinement of the functional model. The idea is illustrated in our example of specifying the Dining Cryptographers problem [Hoa10].

The natural next step of this work is to investigate the refinement of security specifications with the main challenge being to see how to separate the functional and security aspects automatically, and how to compose these models, e.g. building larger protocols from smaller sub-protocols.

2.4 Stochastic Reasoning

The aim of work in Task 8.4 is to introduce the possibility of of stochastic and logical modelling and reasoning into Event-B. This would allow us to address several important challenges. First, it could be used to provide quantitative assessment of such important system attributes as safety, reliability and availability. Second, it is needed for proving "almost certain" termination for a range of applications such as communication protocols. The work in the task has advanced in three main directions:

- Extending Event-B to enable probabilistic discrete time-based reasoning about dependability attributes;
- Reasoning about almost certain termination;
- Reasoning based on continuous time.

2.4.1 Probabilistic Reasoning about Dependability

While refining an abstract system specification, we gradually introduce implementation decisions that initially are modelled nondeterministically. In general, nondeterminism can be resolved in several ways depending on the characteristics of the target system. For example, refinement can aim to obtain the design that guarantees the highest reliability (i.e., probability that a system functions correctly over a given period of time under a given set of operating conditions). We propose to augment Event-B models with probabilistic information required to perform the quantitative analysis of reliability so that this can form a part of the decision-making process during refinement. We have demonstrated that cyclic systems augmented Event-B models can be given a semantics of Markov processes (or Markov chains for fully probabilistic systems). We have shown how to perform reliability assessment using model checking [TTL10a, TTL10b] and have also demonstrated that such systems can be analysed algebraically [TTL10c].

2.4.2 Reasoning about Almost Certain Termination

We continue the work presented in [HH07] to achieve a practical method for qualitative reasoning, and in particular its interaction with refinement. Overall, we would like to reason about qualitative (probabilistic) termination, albeit keeping the reasoning within first-order predicate logic (hence we do not need to extend our proof system for tool support). In the work cited above, we restricted probabilistic choice to the actions of events. Currently we are experimenting with associating probability with the choice of parameters of the events. Moreover, due to the nature of refinement that allows us to reduce non-determinism, qualitative termination arguments might not necessarily be preserved through refinement. Our plans are to find suitable proof obligation rules for refinement that maintain such reasoning. We have focused on the example of the Dining Philosophers problem and have sought to model a probabilistic solution in Event-B [HH09].

2.4.3 Reasoning Based on Continuous Time

In engineering, dependability attributes are often assessed using continuous-time modelling. In [And10] we proposed an approach to extending Event-B models with event rates and giving the models the semantics of continuous-time Markov chains. We have further investigated the topic by modelling a flash filestore example – one of the examples in the Grand Challenge initiative on Verified Software. Our focus is on the analysis of wear-levelling algorithms using novel probabilistic specification and analysis techniques. The expected lifetime of a flash filestore implementing such an algorithm may be derived using probabilistic proof techniques [AMMM10][3].

The work conducted in Task 8.4 has aimed at strengthening the theoretical basis for stochastic reasoning in Event-B. Hence it has not been actively used by the deployment partners so far. However, the theory shows signs of maturity. The results of the task will be used within WP2 to assess the probability of hazardous event occurrence, i.e. to assess safety stochastically. Moreover, the work conducted in WP4 might also be augmented by probabilistic reasoning while designing protocols.

The main technical challenge is to increase usability and scalability of the developing approaches to stochastic and logical reasoning in Event-B. In particular, we are planning to consider architectural-level reasoning, work on finding powerful yet simple proof obligations to support our approaches,

[3]Following a Deploy workshop at Newcastle involving Carroll Morgan and Annabelle McIver, external funding permitted Zoe Andrews to visit the University of New South Wales to work on continuous distributions with for a month in late 2009.

as well as to conduct large case studies inspired by the tasks of deployment partners.

2.5 Real Time

After refocus, a new task **T8.5** was added to the project Description of Work:

> **T8.5 Modelling & Analysis of Real-Time Systems (M20-M48)**: T8.5 develops abstractions and patterns to permit the analysis of timing properties of formal engineering designs. This necessitates the introduction of appropriate abstractions for threads or processes, techniques for expressing modal temporal properties (such as race-freedom) and abstractions/patterns for explicit time in models (allowing analysis of system-level timing constraints in platform-specific models).

The introduction of Task 8.5 was motivated by the need to incorporate temporal properties and the associated reasoning into the Event-B development process. In particular, adding a capability to express modal temporal properties as well as explicitly stating timing constraints in Event-B models would considerably enrich the modelling language, extending the range of potential applications of the formalism.

Deployment partners T8.5 was introduced to meet needs identified by the deployment partners. In most cases the topics of study suggested by the partners were tightly connected with the notions of concurrency (between software threads/tasks/processes) and schedulability analysis. The suggested topics include:

SSF-1 expressing modal temporal properties as well as reasoning about concurrency of software processes (SSF, Bosch);

SSF-2 stating such properties as race freedom (i.e. absence of race conditions) and atomicity, which necessitates introduction of the concept of tasks or processes into Event-B (SSF);

SAP expressing temporal properties relating to recovery from inconsistent states (SAP);

Bosch-1 modelling a real-time clock that can be processed by multiple timers in Event-B (Bosch);

Bosch-2 incorporating the notion of parallel tasks or processes (especially for specifying periodic tasks) into Event-B modelling, which is subsequently used for code generation (Bosch, SSF).

Additionally, some preliminary work has been carried out on the following topic:

Real time analysis using schedulability analysis (in particular, the worst-case execution time analysis) in the formal development process.

Progress The academic partners have begun research to address to address the topics raised by the deployment partners. Åbo Akademi has focused on systematic introduction of the tasking structure and its properties (e.g. scheduling policies) into the Event-B development process (topics **SSF-1, SSF-2**). At some point of development, system events are grouped together into separate tasks, dynamic execution of which is driven by an abstract scheduler. The scheduler itself can be refined to implement different scheduling policies.

Åbo has developed a method for deriving sequential programs from Event-B specifications (topic **Bosch-2**). Bostrom [Bos10] describes the approach by first presenting a suitable Event-B semantics for development of sequential programs. A scheduling language is then introduced to describe the control flow. Throughout the paper, the algebraic method for reasoning about loops developed by Back and von Wright [BvW99] is used to analyse the models and the event scheduling. The combined approach allows us to develop and verify different scheduling patterns. The advantage of this approach is that much of the development of sequential programs can be carried out using the current tools of Event B. Only the final scheduling step introduces flow control constructs that may require generation of a few additional proof obligations. In the future, the work will be extended to model execution of parallel threads or tasks of the system.

Issues in real-time system modelling flow over into several other dependability tasks. In work related to Task 8.1, Newcastle has investigated formalising software systems that allow continuous observations, in particular, developing the timeband framework for modelling real-time systems (topics **SSF-1, Bosch-1**). In Task 8.2, Newcastle has also developed patterns for modelling time and information consistency in business information systems [BFRR10] (topic **SAP**), as discussed in Section 2.2. In a useful spin-off topic, Newcastle has also begun related work on requirements for formalisms to support modelling and analysis of dynamic reconfiguration of dependable real-time systems (topic **SSF-1**) [MB10].

Newcastle and Åbo have started to develop an approach to verifying real-time properties of Event-B models by building an intermediate *Process View* model (topics **SSF-1, SSF-2, Real time analysis**) that provides the explicit notions of process and synchronization while abstracting from functional detail. Rely-Guarantee reasoning [Jon83] is applied to define process abstractions in a compositional way. Project View models augmented with clocks and time constraints can be translated into UPPAAL to verify liveness and real-time properties. An important concern is the verification of Process View consistency and the link between Process View and Event-B models. All the relevant conditions are expressed in the way that makes them amenable to the verification in Rodin. This work is motivated by the satellite on-board software (AOCS) case study proposed by SSF.

Southampton has worked on modelling time constraints (delay, deadline, duration) for software tasks or processes (topics **SSF-1, Bosch-1**). Also, Southampton has proposed an intermediate specification notation for modelling concurrent software processes, which allows direct translation into executable code [EB08, Edm10] (topic **Bosch-2**). Work on exploring links between Event-B and UPPAAL has also been undertaken (topic **Real time analysis**).

Future work The main challenges for the research in this area are explicit and consistent modelling of real time and its duration in Event-B as well as quantitative verification of extended Event-B models. Moreover, a possible bridging with external tools supporting real time verification (e.g. UPPAAL) should be explored more thoroughly. These two directions are main future activities in the task. The main risk is that some of the initiated research will not reach sufficient maturity to be fully deployed before the end of the project.

Chapter 3

WP7: Productivity through Reuse

During the reported period, WP7 research has advanced in the following directions:

- Developing Event-B design patterns (ETH);
- Extending Event-B language with generic modules (Newcastle, Åbo);
- Proposing generic templates (module interfaces) for mode-managing components (Newcastle, Åbo);
- Reasoning about inter-module dependencies (Åbo);
- Integrating fault tolerance mechanisms using proposed refinement patterns (Newcastle).

In addition to this work, a number of the plug-ins for the Rodin platform that are relevant to WP7 have been developed or extended. These include the modularisation plug-in (Newcastle), the decomposition plug-in (Systerel and Southampton), and the refactoring plug-in (Southampton).

We describe below these latest developments in more detail.

Design patterns ETH has investigated further the connection between the choreography models used in SAP and the application of Event-B patterns. The input for the pattern application is generated from matching of choreography models, while the generated output is the corresponding Event-B model. Earlier results of this work were published in [HFA09].

The main idea is to have a pattern represented by a simple choreography model (in the same way as in the actual problem). Application of patterns

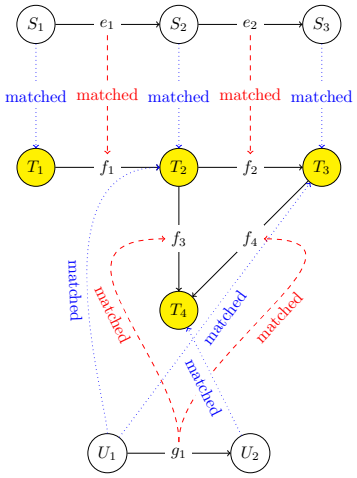

Figure 3.1: Matching choreography models

amounts to just "matching" the two choreography models. In general, more than just one pattern can be applied to a problem, and, as a result, the problem choreography model can be matched with several pattern models. From these matching choreography models, the information about pattern applications in Event-B can be derived. This initial idea has been reported in the deliverable D21.

The approach is illustrated (at a very high level) in Figure 3.1, where the problem choreography model (indicated by nodes $T_1 \ldots T_4$) is matched with two patterns (denoted by nodes $S_1 \ldots S_3$ and $U_1 \ldots U_2$). In the figure, the developer needs to match the events of the patterns with the events of the problem. Here e_1 is matched with f_1, e_2 is matched with f_3 and g_1 is matched to both f_3 and f_4 (indicated by dashed arrows). The matching of the states (dotted arrows) is a consequence of the matching of the events. The input for the application of patterns hence can be generated automatically.

As possible continuations of this work, ETH sees developing a tool support for linking between choreography models, and, more importantly, validating the approach with more case studies, starting with the SAP pilots described in [DEP10c].

Generic modules Newcastle and Åbo have extended the Event-B language and tool support with a possibility to define modules [ITL+10b, ROD10] – components containing groups of callable operations. Modules can have their own (external and internal) state and the invariant expressing properties on this state. The important characteristic of modules is that they can be developed separately and then composed with the main system during its

formal development.

A module description consists of two parts – *module interface* and *module body*. Let M be a module. The module interface MI is a separate Event-B component. It allows the user of the module M to invoke its operations and observe the external variables of M without having to inspect the module implementation details. The module interface consists of module interface description MI and the context MI_Context. The context defines the constants c, and sets s. The interface description correspondingly consists of the external module variables w, the external module invariant M_Inv(c, s, w), and a collection of module operations, characterised by their pre- and postconditions, as shown below.

INTERFACE MI =
  SEES MI_Context
  VARIABLES w
  INVARIANT M_Inv(c, s, w)
  OPERATIONS
    res ← op_1(par) =
      PRE M_$Guard_1$(c, s, par, w)
      POST M_$Post_1$(c, s, par, w, w', res')
  ...
END

Figure 3.2: Interface component

A module development always starts with the design of an interface. Once an interface is formulated and declared final, it cannot be altered in any manner. This ensures correct relationships between a module interface and its body, i.e. that the specification of an operation call is recomposable with an operation implementation. A module body is an Event-B machine, which implements each operation described in the module interface by a separate group of Event-B events. Additional proof obligations generated to prove correctness of a module guarantee that each event group faithfully implements the given pre- and postconditions of the corresponding interface operation.

Module instantiation allows us to create several instances of the same module. Obviously, different instances of a module operate on disjoint state spaces. Via different instantiation of generic parameters the designers can easily accommodate the required variations when developing components with similar functionality. Hence module instantiation provides us with a powerful mechanism for reuse.

Developing generic templates for mode-managing components This is joint work by Newcastle, Åbo and SSF, which can be seen as application

of the Event-B modularisation extension described above for modelling space satellite systems. Dynamic behaviour of such systems is typically described in terms of operational modes that correspond to the different stages of a mission and states of the components. Components are susceptible to various faults that complicate mode transition schemes. Yet the success of a mission depends on the correct implementation of mode changes.

The proposed formal approach ensures consistency of complex mode transitions while developing a system architecture by refinement. The approach relies on recursive application of modelling and refinement patterns that enforce correctness while implementing mode and mode transitions. The proposed approach is exemplified by the development of the Attitude and Orbit Control System (AOCS) [DEP10b, ILT10] suggested by SSF.

The achieved results were reported in a conference publication [ITL+10c]. The paper also proposes a generic pattern for specifying components of layered mode-rich systems. This pattern essentially defines a generic module interface that can be instantiated by component-specific data and behaviour. The paper demonstrates that layered mode-rich systems can be developed in an incremental modular style by recursive instantiation of the proposed pattern. On the architectural level, the approach can be seen as a stepwise unfolding of architectural layers.

The Event-B framework extended with modularisation capabilities is used as the modelling language. Availability of automatic tool support – the Rodin platform and the modularisation plug-in [ROD10] – makes the approach relevant in industrial practice.

Inter-module dependencies In a separate line of research, Åbo is developing an approach for reasoning about inter-module dependencies. The approach is based on the action system formalism and the notion of module/component contracts. This work is also inspired by the SAP deployment.

An earlier version of this approach (published in [NDS09]) has focused on service-oriented systems. A service might collect information from several sources and also deduce new information out of these. Whether being an intermediate service or an elementary source of information, the provided service needs to be considered as a black-box from the utilising system point of view, i.e. as a component. Moreover, the utilising system might depend on this information and halt until it is available.

To reason, systematically and rigorously, about the service dependencies they are formalised within the action system framework. The reactive nature of action systems provides means for reasoning about the information in a service-like distributed manner. The focus is on examining the consequences

to the action system when applying uni- and bidirectional dependencies.

Refinement patterns for fault tolerance Recently Newcastle has developed a new approach, focusing on modelling fault tolerance (FT) and its reuse. The approach is based on the performed analysis of fault tolerance related requirements obtained from the Deploy industrial partners. It enables systematic integration and reuse of fault tolerance modelling. The main points of the approach are the following:

- Expressing the FT concern as a separate model view;
- Presenting system abstractions as FT views;
- Defining an iterative process of FT refinement (alongside the Event-B refinement chain), while using special *detalisation templates*;
- Specifying a transformation mechanism for Event-B models, aimed at further integration of FT modelling (ongoing work);
- Reusing FT modelling decisions via libraries of detalisation templates and refinement patterns.

The work was reported in the SERENE workshop [LIR10].

Plug-in development A number of Rodin plug-ins that are relevant to WP7 have been developed or enhanced during the reported period:

Modularisation plug-in Developed by Newcastle. Allow definition of generic modules containing groups of callable operations. Introduces a new component type – a module interface. The interface can have a generic parameters that are instantiated by concrete values during module composition. Essentially, a generic module interface stands for a separate parameterised Event-B development.

Decomposition plug-in Developed by Systerel and Southampton. Allows to decompose a model into sub-models. Supports two types of model decomposition – by shared variables (A-style) and shared events (B-style).

Refactoring plug-in Developed by Southampton. Supports renaming of different Event-B elements in such a way that the refactoring does not change the semantics of any of the affected elements/files.

Chapter 4

WP6: Requirements

The objectives of this Work Package are to investigate useful techniques for managing and validating requirements for complex dependable systems. The main goal is to develop techniques for tracing requirements throughout the entire Event-B development process from the abstract specification to its implementation. These techniques should also support requirements evolution that occurs during system development. Another goal is to adopt some techniques for integrating requirements obtained from RAMS (reliability, availability, maintainability, safety) activities into formal system modelling and analysis.

This Work Package is structured under two sub-tasks according to the priorities identified by the deployment partners and the expertise of technology-providing partners. These sub-tasks are T6.1: **Requirements tracing and validation** and T6.2 **Requirements evolution**.

The scope of T6.1 is to develop techniques for tracing requirements throughout the entire Event-B development process from the abstract specification to its implementation. Building on previously available methods some methods are provided that facilitate construction of formal specifications from informal requirements [Ili07]. This includes providing some patterns for structuring informal requirements and guidelines for incorporating requirements into the refinement process. We have developed techniques supporting early requirements validation via graphical modelling, model animation and testing. The ultimate goal of these approaches is to allow seamless integration of dependability-related requirements obtained via RAMS activities into Event-B system engineering. To ensure that developed systems achieve the desired degree of dependability, we have developed guidelines for validating dependability-related requirements in the Event-B engineering process. The results feed into the DEPLOY tooling Work Package WP9 by developing a basic plug-in to support requirement management and traceability.

T6.2 concerns changes in the initial requirements imposed on a system during its construction due to evolution of knowledge about the system behaviour. Also we have to consider changes in the environment in which the system will operate or changes in the business processes. Every deployment partner perceives requirements evolution as one of the major challenges in modern system engineering. To ensure successful deployment of formal engineering methods, we aim to create techniques to deal with requirements evolution efficiently and effectively. We develop approaches supporting an iterative style of system development that simplifies capturing new or altered requirements. In this work we investigate how to cope with changing or emerging requirements via the use of generic specification and refinement patterns. Changes to requirements will result in changes to models and proofs and any developed techniques should help to minimise the effort of revising models and re-proof.

To summarise, the outcome of this Work Package should result in producing reusable specification and refinement patterns for capturing requirements and provide some guidelines for requirements traceability and validation. It has to provide some methods for dealing with requirements evolution in Event-B formal development. In the following section we provide an overview of what has been achieved since the last report.

4.1 The Link between Problem Frames and Event-B

This section reports on further application and elaboration of the Problem Frames and Event-B link, which is mainly done in conjunction with Bosch. Linking Problem Frames and Event-B was the main interest of Bosch under WP1.

The initial approach has been described in Deliverable D15. It provides guidelines on requirements structuring, mapping to Event-B and traceability between requirements and Event-B models. The approach has been applied in the Cruise Control deployment in WP1 and described in Deliverable D19. Here we outline what has been achieved since the publication of D15. The starting point in the Bosch pilot was requirements elicitation and engineering. The outcome of this initial phase was a requirements document in natural language which describes the functional requirements of the system. Based on observation made during the first two years, the gap between the textual requirements and Event-B models was considered very big. This could hinder the validation process between the Event-B models and the initial

requirements. To overcome this problem Bosch decided to introduce a semi-formal step in the requirements engineering process. The Problem Frames method [Jac01] is chosen for this purpose. As described in D19, Bosch extended the existing Problem Frame approach with the notions of elaboration and decomposition and called it extended Problem Frames. These extensions support a smooth translation of a Problem Frame model into Event-B. They allow decomposition of a given problem into different sub-problems which can be handled separately and later recombined in a subsequent phase. The following approach is taken to produce a formal model of the cruise control system in Event-B from semi-formal requirements:

- Each *problem diagram* is modelled as a separate machine with its associated context;
- *Elaborations relation* between different abstraction levels of PFs is realised in Event-B by *refinement* of the machine and its associated context;
- *Projections* of a problem diagram into two or more sub-problems are realised in Event-B by *shared-variable decomposition* [Abr09] with some changes;
- Each *phenomenon* defined in a problem diagram is modelled either as a *constant* or a *variable* in Event-B;
- Abstract phenomena which need elaboration in subsequent stages are realised in Event-B using *records*;
- Elaborations of *phenomena* in problem diagrams are realised in Event-B using *data refinement*;
- *Action and behaviour* stated in problem diagrams are realised in Event-B by *events* and/or *invariants*.

Bosch holds the view that without introduction of the extended Problem Frames approach, they would have difficulty in order to validate the Event-B models against the initial textual requirements. They believe that with the introduction of the Problem Frames approach they have two simpler validation problems: the first one is to validate whether the Problem Frames model is correct with respect to the requirements document and the second one is to validate whether the Event-B model is correct with respect to the Problem Frames model. From the viewpoint of Bosch these two validation problems are easier to solve than the initial one. In addition, having the semi-formal requirements in the form of Problem Frames makes the mapping into

Event-B very promising. This is because of the similarity between the Event-B refinement notation and elaboration in the Problem Frames approach. An overview of some devised guidelines for the translation is presented earlier though there are still some challenges left, i.e. the problem of tracing the implementation of requirements in the Event-B specification.

4.2 Linking Natural Language Requirements and Event-B

This section reports on further advances in establishing traceability between natural language requirements and Event-B, mainly done in conjunction with the University of Düsseldorf. There are two aspects to this work. We continued to establish a theoretical foundation for traceability by adapting the WRSPM reference model. Secondly, we drove tool support forward by building a tool platform for requirements engineering. As of this writing, this platform is still being developed and has not yet been been presented to the Deploy project partners.

The work on the prototype requirements plug-in developed in Year 1 has been abandoned. While it was useful for exploring the issues associated with requirements management, the practical and architectural needs changed so much that it made more sense to start over.

4.2.1 Traceability between Natural Language Requirements and Event-B using WRSPM

Establishing traceability between textual requirements in a natural language and the Event-B models is a prime goal of this work [JHLJ10]. To achieve this goal, a set of intermediate constructs based on WRSPM [GGJZ00] is adopted to structure the initial requirement and resulting Event-B models. The WRSPM reference model provides a reliable and systematic interface between the informal requirements and the Event-B models.

We describe an approach for building a formal model from natural language requirements. Our aim is to increase the confidence that the formal model represents the desired system, by explaining how the requirements are "realised" in the formal model. The relationship "realises" between requirements and formal models is kept informal. Justifications are maintained with each requirement and element of a formal model that are linked by "realizes", tracing requirements into the model, providing the sought explanation. Hence, the technical problem we have to solve is how to trace requirements

into a formal model.

Requirements traceability provides a justification for a formal model with respect to the requirements. It is a difficult problem [Bjø08, GF94, Jas09]. Furthermore, it is a cross-disciplinary problem connecting requirements engineering and formal methods. The benefits of the use of formal methods during requirements engineering has long been recognized. For instance, [Ber02] quantifies the impact of formal methods in requirements engineering based on industrial case studies.

A key contribution is the traceability between natural language requirements and the Event-B model which is structured according to WRSPM. It allows us to cope with changes in the model and changes in the requirements. Our approach distinguishes the following three types of traces:

Evolution Traces: As the model evolves over time, there is traceability from one iteration to the next. This is particularly useful for the stakeholders to verify that changes to the requirements reflect their intentions. This can be done by exploring the requirement's evolution over time, allowing the stakeholder to compare the original requirement to the modeler's revision.

Explicit Traces: Each non-formal requirement is explicitly linked to at least one formal statement. These traces are annotated with a justification that explains why the formal statement corresponds to the non-formal requirement.

Implicit Traces: There is implicit traceability within the Event-B model. Those traces can be discovered via the model relationships (e.g. refinement relationships, references to model elements or proof obligations). For instance, a guard that ensures that an invariant holds is implicitly linked to that invariant via a proof obligation.

Tracing an element of the formal model to an original requirement may require following a chain of traces.

4.2.2 Dealing with Changes in Requirements and Models

We assume that the requirements and the formal model need to be changed frequently and assume that the requirements are incorporated incrementally into the model. In the process, the requirements may have to be rewritten, corrected, clarified or split. The formal model may have to be modified correspondingly as the requirements become better understood [HL09].

We consciously limit the scope of our approach. We assume that we start with a set of "reasonable" user requirements, but do not provide a method for eliciting them because good elicitation methods exist [Jac01, DDMvL97].

4.2.3 Tool Support

We created a few case studies to explore the usefulness of our ideas and realised that tool support is crucial for even medium-sized projects. However, we didn't want to pursue tool development (or the extension of the existing tool) without a solid theoretical foundation. We have now reached a point where the theory is mature enough to start building tool support.

We considered using the existing requirements plug-in as a starting point, but eventually abandoned the idea. There were technical and methodological

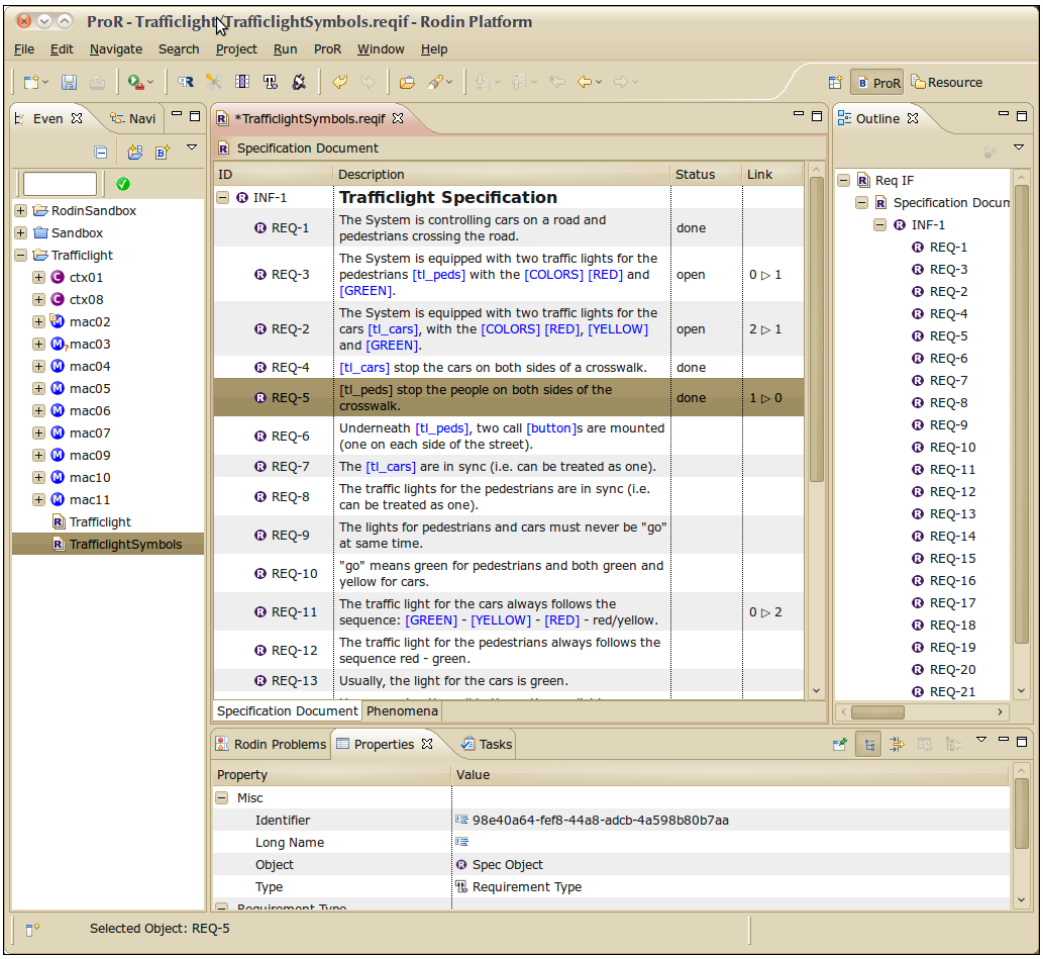

Figure 4.1: Screenshot of the ProR Requirements Plug-In for Rodin

reasons for this.

On the architectural side, we wanted to take advantage of the Eclipse Modeling Framework (EMF). It is a code generation facility for building tools and other applications based on a structured data model. EMF provides tools and runtime support to produce a set of Java classes for the model, along with a set of adapter classes that enable viewing and command-based editing of the model, and a basic editor. Through the Rodin EMF plug-in, we can build on a technology that is already used within Rodin.

Furthermore, we want to be able to interoperate with existing processes of the industrial partners. Especially in the automotive industry, the ReqIF exchange format for requirements [Req10] is starting to be used more and more. A number of tools support this format, including the market leader in the automotive field, IBM Doors. ReqIF provides a clean, powerful, generic data model for requirements that we can leverage to build specialised tools that support our method, while keeping interoperability with industry processes intact.

On the methodical side, there were a large number of features missing from the existing requirements plug-in that were necessary for our method. Specifically, the synchronisation architecture wasn't powerful enough. The plug-in associates requirements in various formats with the Event-B model. Synchronisation has to be triggered manually. In contrast, the EMF notification mechanism allows us to hook in fine-grained listeners in both the Event-B and the requirements model to react to changes as appropriate.

We expect to present our tool (Figure 4.1), together with a proof-of-concept Rodin integration in autumn 2010. Progress can be tracked at [Pro10].

4.2.4 An Overview of Achievements

In this work, we presented an approach for building a formal model from natural language requirements. With our approach, the boundary between informal requirements and formal model is clearly defined by annotated chains of traces, which keep track of model evolution and explicit and implicit links.

In addition to the explicit traceability between requirements and model, we take advantage of the implicit traceability within the formal model to support us in verifying the model against the requirements. In particular, we take advantage of traceability through proof obligations: when all proof obligations are discharged, we know that the model is consistent. If we trust our traceability, then we have confidence that our requirements are consistent as well. Common identifiers can be used in the informal requirements and formal model. A supporting tool could assist the user by pointing out

matching identifiers.

Our approach has proven successful with a number of small projects. In the near future, we will tackle bigger case studies; we will incorporate ongoing research like decomposition.

As of this writing, the effort for building tool support within the Rodin platform is well under way.

4.3 Evaluation of Cookbook Guidelines

This section looks at the evaluation of a specific "cookbook approach" by formal modelling of a simplified cruise control system in Event-B.

As a part of D15, a set of guidelines in the form a cookbook has been developed for specification and refinement of control problems in Event-B. The Event-B formal method is used for system-level modelling by defining states of a system and events which act on these states. It also supports refinement of models.

This cookbook is intended to systematise the process of modelling and refining a control problem system by distinguishing environment, controller and command phenomena. In [YBR10] the usefulness and effectiveness of this cookbook is investigated and evaluated by following it throughout the formal modelling of a cruise control system found in cars. The outcomes are identifying the benefits of the cookbook and also giving guidance to its future users.

The model of the cruise control system represented in this work contains the platform, the environment and the software application. Separation of these concepts through decomposition in later steps of design allows us to derive a specification of the control system, since the software and the hardware are being separated. In addition, other aspects of an embedded system are usually analysed and modelled through different techniques to formal methods.

4.4 Event Refinement Diagrams

This section looks at using "Event Refinement Diagrams" to structure and incorporate requirements into formal models.

Modelling of large and complex systems can result in large and complex models and difficult proofs [AH07]. Refinement techniques can address this complexity by developing the final model in a stepwise manner. In an Event-B specification it is common to represent a desired outcome as an abstract

atomic event and then decompose it into more concrete events in the subsequent refinement levels. This is called atomicity decomposition. If the abstraction gaps between refinement levels are well-devised, we only need to add relatively small details in each refinement level and proof obligations would be relatively easy to discharge. Most of proof obligations are related to consistency between refinement levels, so with the small gaps between these levels, discharging proof obligations should be a straightforward task.

Despite the advantages outlined above, the Event-B refinement method does not explicitly represent all refinement connections between abstract and concrete events. The implicit refinement relation between an abstract event and some of its refined counterparts makes it difficult to comprehend complex Event-B models. As a direct consequence of this, devising an ordering or control structure between refined events appears as a challenging issue. To overcome this problem a graphical notation called *Atomicity decomposition diagrams* is introduced in [But09]. This approach, which is adopted from JSD notation, [Jac83] provides a structuring mechanism that can direct the refinement process. The atomicity decomposition technique also helps to structure requirements and incorporate them into the Event-B refinement chain in a systematic way. It is intended to make the standard refinement rules clearer and their application more systematic. The diagrammatic notation of atomicity decomposition shows relationships between refinement levels. In this approach usually a single event shows the goal in the abstract level, and then it is decomposed to sub-events in the subsequent refinements.

The atomicity decomposition technique is investigated by applying it to a media channel case study [FB10]. At the abstract level only service requirements of the media channel system are considered. These include three phases of establishing, modifying and closing a channel.

Using hierarchical diagrams proposed by the atomicity decomposition approach, the control structure is introduced into event-B models in an incremental approach. This work demonstrates how service-level requirements can be refined into protocol requirements employing the benefits of atomicity decomposition techniques in structuring requirements.

Applying this technique to the multi media system partly shows that the technique can help overcome some of the complexity problems. Based on our experience in this case study, we believe that atomicity decomposition can be scaled to complex systems. The atomicity decomposition technique makes the standard refinement more systematic and visual. Sequential relations between levels of refinement during incremental modelling of a system are structured in atomicity decomposition diagrams. In this approach we start with a more global view of the intention of the protocol and then use atomicity decomposition to arrive at models that include all protocol details.

4.5 Deriving Formal Specifications

This section consists of an overview of the work done at Newcastle University on requirements. As mentioned in Section 2.1, Newcastle's effort on requirements focused on the specification of dependable systems and how to derive formal specifications starting from informal requirements which are generally expressed in natural language. Particular attention has been also paid to methodological aspects as discussed in [Maz09a] and [Maz09b].

Deliverables D19 [1] and D15 [2] document the close collaboration with Bosch in the development and application of both preliminary ideas as well as more established ones. In particular, D19 describes Newcastle's approach in Chapter 6 with an early discussion about linking Problem Frames and Event-B. The HJJ research strand has been also discussed as a "method" for deriving specifications of systems which interact with the physical world [JHJ07].

Further progress has been made since D19. First, what we know and what we do not know about the use of Problem Frames for semiformal specification of systems has been clarified [Jon10a]. Second, the process from informal to formal specifications for modelling and analysis purposes has been investigated by means of different case studies inspired by different domains like Business Process Modelling [DM09] or real-time dependable systems [MB10]. Finally, the role of layers of abstraction has been further discussed in [Jon10b] and a detailed treatment on using rely/guarantee conditions in developing a delicate concurrent program has been presented in [JP10].

[1] "Pilot Deployment in the Automotive Sector", January 2010

[2] "Advances in Methods", July 2009

Chapter 5

Plans for D44 (D6.3)

As agreed at the March 2009 review, the DEPLOY project will attempt to produce a final methods deliverable which describes a range of approaches to developing systems. The core method is obviously that of Event-B but we will also aim to describe other approaches that have been used by the DPs. Note that the previous sentence is a deliberate restriction: we are not even considering writing a "complete" review of formal methods for dependable systems.

It must be remembered that this is an ambitious objective that is likely to commit us to a book length document. Before the next review in early 2011, we will take a position on whether it would be more prudent to aim for a web information structure.

Current ToC We have obviously started discussions on the proposed document: what follows is our current view of a likely table of contents.

- Introduction
  - Notion of formal description
  - History (brief)
  - Concept of modelling
- Event-B
  - Notation
  - Semantics
- Modelling
  - Building up a specification ("horizontal refinement")

  - Structuring data
  - Examples
  - Animating models
- Reification[1] ("vertical refinement")
  - Formal rules and discharging proof obligations
  - Constructing gluing invariants
  - Examples
  - Animating at multiple levels
  - Code generation
  - Handling concurrency
- Developing dependable systems (handling exceptional behaviour)
  - Modelling dependability requirements
  - Describing exceptional behaviour
  - Fault tolerance in design
  - Security
- Getting from requirements to models
  - UML-B
  - problem frame approach
- Handling large applications
  - Decomposition
  - Modules
  - Patterns
- Tackling things which are not obviously formulated as "events"
  - Fault-tolerance as a generic concern
  - Concurrency
  - Time

[1] This is the VDM term which may or may not be used – but we clearly need to move away from "vertical/horizontal".

  - Continuous behaviour
  - Probabilities
- Other approaches
  - ...
  - ...

Bibliography

[Abr09] Jean-Raymond Abrial. Event model decomposition. Technical report, ETH Zürich, 2009.

[Abr10] Jean-Raymond Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, May 2010.

[AH07] Jean-Raymond Abrial and Stefan Hallerstede. Refinement, decomposition, and instantiation of discrete models: Application to Event-B. *Fundamenta Informaticae*, 77(1-2):1–28, 2007.

[ALRL04] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, 2004.

[AMMM10] Zoe Andrews, Annabelle McIver, Larissa Meinicke, and Carroll Morgan. Probabilistic Aspects of Flash Filestores. Technical Report CS-TR-1202, School of Computing Science, Newcastle University, 2010. Accepted for the *Verified Software: Tools and Experiments* Workshop at VSTTE 2010, Edinburgh, August 2010.

[And10] Zoe Andrews. Towards a Stochastic Event-B for Designing Dependable Systems. In M. Jastram, L.Laibinis, F. Lösch, and M. Mazzara, editors, *Proceedings of the First Deploy Technical Workshop*. Technical Report CS-TR-1187, School of Computing Science, Newcastle University, 2010.

[Ber02] D. M. Berry. Formal Methods: The Very Idea – Some Thoughts About Why They Work When They Work. *Science of Computer Programming*, 42(1):11–27, 2002.

[BFRR10] J. W. Bryans, J. S. Fitzgerald, A. Romanovsky, and A. Roth. Patterns for Modelling Time and Consistency in Business Infor-

mation Systems. In *Proc. 15th IEEE Intl. Conf. on Engineering of Complex Computer Systems*, pages 105–114, March 2010.

[BGJ09a] Alan Bundy, Gudmund Grov, and Cliff B. Jones. Learning from experts to aid the automation of proof search. In Liam O'Reilly and Markus Roggenbach, editors, *AVoCS'09 – PreProceedings of the Ninth International Workshop on Automated Verification of Critical Systems*, Technical Report of Computer Science CSR-2-2009, pages 229–232. Swansea University, 2009.

[BGJ09b] Alan Bundy, Gudmund Grov, and Cliff B. Jones. An outline of a proposed system that learns from experts how to discharge proof obligations automatically. In Jean-Raymond Abrial, Michael Butler, Rajeev Joshi, Elena Troubitsyna, and Jim C. P. Woodcock, editors, *Dagstuhl 09381: Refinement Based Methods for the Construction of Dependable Systems*, pages 38–42, 2009.

[Bjø08] D. Bjørner. From Domain to Requirements. In *Concurrency, Graphs and Models: Essays dedicated to Ugo Montanari on the Occasion of his 65th Birthday*, pages 278–300. Springer, 2008.

[Bos10] P. Boström. Creating Sequential Programs from Event-B Models. Proceedings of *Integrated Formal Methods 2010 (IFM 2010), Nancy, France*, Lecture Notes for Computer Science, Springer, October 2010.

[But09] Michael Butler. Decomposition structures for Event-B. In *IFM '09: Proceedings of the 7th International Conference on Integrated Formal Methods*, pages 20–38, Berlin, Heidelberg, 2009. Springer-Verlag.

[BvW99] R. Back and J. von Wright. Reasoning algebraically about loops. *Acta Informatica*, 36:295–334, 1999.

[BW10] J. W. Bryans and W. Wei. Formal Analysis of BPMN models using Event-B. In *Proceedings of 15th International Workshop on Formal Methods for Industrial Critical Systems (FMICS)*, 2010. To appear.

[CMR07] Dominique Cansell, Dominique Méry, and Joris Rehm. Time Constraint Patterns for Event B Development. In Jacques Julliand and Olga Kouchnarenko, editors, *B 2007: Formal Specification and Development in B, 7th Intl. Conf. of B Users, Besançon, France,*

January 17-19, 2007, volume 4355 of *Lecture Notes in Computer Science*, pages 140–154. Springer, 2007.

[DDMvL97] R. Darimont, E. Delor, P. Massonet, and A. van Lamsweerde. GRAIL/KAOS: An Environment for Goal-Driven Requirements Engineering. In *Proc. of the 19th int. conf. on Software engineering*, pages 612–613. ACM, 1997.

[DEP09a] DEPLOY. D2.1 Pilot Deployment in Transportation. FP7 ICT DEPLOY Project, September 2009. Deliverable D16 – Online at `http://www.deploy-project.eu/`.

[DEP09b] DEPLOY. D6.1 advances in methods. FP7 ICT DEPLOY Project, 2009. Deliverable D15 – Online at `http://www.deploy-project.eu/`.

[DEP10a] DEPLOY. D1.1 Pilot Deployment in the Automotive Sector. FP7 ICT DEPLOY Project, January 2010. Deliverable D19 – Online at `http://www.deploy-project.eu/`.

[DEP10b] DEPLOY. D3.1 Report on Pilot Deployment in the Space Sector. FP7 ICT DEPLOY Project, January 2010. Deliverable D20 – Online at `http://www.deploy-project.eu/`.

[DEP10c] DEPLOY. D4.1 report on pilot deployment in business information sector. FP7 ICT DEPLOY Project, January 2010. Deliverable D21 – Online at `http://www.deploy-project.eu/`.

[DIRR09] F. L. Dotti, A. Iliasov, L. Ribeiro, and A. Romanovsky. Modal Systems: Specification, Refinement and Realisation. In *Formal Methods and Software Engineering. 11th International Conference on Formal Engineering Methods*, volume 5885 of *Lecture Notes in Computer Science*, pages 601–609. Springer, December 2009.

[DM09] N. Dragoni and M. Mazzara. A Formal Semantics for the WS-BPEL Recovery Framework - The Pi-Calculus Way. In *WS-FM'09*, volume 6194 of *Lecture Notes in Computer Science*. Springer, 2009.

[EB08] A. Edmunds and M. Butler. Linking Event-B and Concurrent Object-Oriented Programs. *Electronic Notes in Theoretical Computer Science*, 214:159–182, 2008.

[Edm10] A. Edmunds. *Providing Concurrent Implementations for Event-B Developments*. PhD thesis, University of Southampton, March 2010.

[FB10] Asieh Salehi Fathabadi and Michael Butler. Applying Event-B atomicity decomposition to a multi media protocol. In *FMCO Formal Methods for Components and Objects.*, 2010.

[GF94] O. Gotel and A. Finkelstein. An Analysis of the Requirements Traceability Problem. In *Proc. of the First Int. Conf. on Requirements Engineering*, pages 94–101, 1994.

[GGJZ00] Carl A. Gunter, Elsa L. Gunter, Michael Jackson, and Pamela Zave. A reference model for requirements and specifications. *IEEE Softw.*, 17(3):37–43, 2000.

[HBA09] Thai Son Hoang, David Basin, and Jean-Raymond Abrial. Development of Access Control Systems in Event-B. Technical Report 624, Department of Computer Science, ETH Zurich, June 2009. `http://www.inf.ethz.ch/research/disstechreps/techreports`.

[HFA09] Thai Son Hoang, Andreas Fürst, and Jean-Raymond Abrial. Event-B patterns and their tool support. In Dang Van Hung and Padmanabhan Krishnan, editors, *SEFM*, pages 210–219. IEEE Computer Society, 2009.

[HH07] Stefan Hallerstede and Thai Son Hoang. Qualitative Probabilistic Modelling in Event-B. In Jim Davies and Jeremy Gibbons, editors, *IFM 2007: Integrated Formal Methods, Proceedings of the 6th International Conference*, volume 4591 of *Lecture Notes in Computer Science*, pages 293–312, Oxford, U.K., July 2007. Springer Verlag.

[HH09] Stefan Hallerstede and Thai Son Hoang. Qualitative Reasoning for the Dining Philosophers – Extended Abstract. In J. Abrial, M. Butler, R. Joshi, E. Troubitsyna, and J.C.P Woodcock, editors, *Dagstuhl Seminar 09381 – Refinement Based Methods for the Construction of Dependable Systems*, number 09381 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2009. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany. `http://drops.dagstuhl.de/portals/09381/`.

[HL09] Stefan Hallerstede and Michael Leuschel. How to Explain Mistakes. In Jeremy Gibbons and José Nuno Oliveira, editors, *Teaching Formal Methods, Second International Conference, TFM 2009, Eindhoven, The Netherlands, November 2-6, 2009. Proceedings*, volume 5846 of *LNCS*, pages 105–124. Springer, 2009.

[Hoa10] Thai Son Hoang. Specification of the Dining Cryptographers Problem. DEPLOY project repository: `http://deploy-eprints.ecs.soton.ac.uk/215/`, April 2010.

[Ili07] Duvravka Ilic. Deriving formal specifications from informal requirements. In *COMPSAC '07: Proceedings of the 31st Annual International Computer Software and Applications Conference*, pages 145–152, Washington, DC, USA, 2007. IEEE Computer Society.

[ILT10] A. Iliasov, L. Laibinis, and E. Troubitsyna. An Event-B model of the Attitude and Orbit Control System, 2010. DEPLOY repository: http://deploy-eprints.ecs.soton.ac.uk/.

[IRD09] A. Iliasov, A. Romanovsky, and F.L. Dotti. Structuring specifications with modes. In *2009 Fourth Latin-American Symposium on Dependable Computing, LADC 2009*, pages 81–88. IEEE Computer Society, September 2009.

[ITL+10a] A. Iliasov, E. Troubitsyna, L. Laibinis, A. Romanovsky, K. Varpaaniemi, D. Ilic, and T. Latvala. Developing Mode-Rich Satellite Software by Refinement in Event-B. In *Proceedings of 15th International Workshop on Formal Methods for Industrial Critical Systems (FMICS 2010)*. Springer, September 2010.

[ITL+10b] A. Iliasov, E. Troubitsyna, L. Laibinis, A. Romanovsky, K. Varpaaniemi, D. Ilic, and T. Latvala. Supporting reuse in Event-B development: Modularisation approach. In *Proceedings of Second International Conference on Abstract State Machines, Alloy, B, and Z (ABZ 2010)*, volume 5977 of *Lecture Notes in Computer Science*, pages 174–188. Springer, 2010.

[ITL+10c] A. Iliasov, E. Troubitsyna, L. Laibinis, A. Romanovsky, K. Varpaaniemi, P. Väisänen, D. Ilic, and T. Latvala. Verifying Mode Consistency for On-Board Satellite Software. In SAFECOMP 2010, The 29th International Conference on Computer

Safety, Reliability and Security, September, Vienna, Austria. Lecture Notes for Computer Science, Springer, 2010.

[Jac83] M. A. Jackson. *System Development*. Prentice Hall, 1983.

[Jac01] Michael Jackson. *Problem frames: analyzing and structuring software development problems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[Jas09] M. Jastram. Requirements Traceability. Technical report, U. Southampton, 2009.

[JHJ07] Cliff B. Jones, Ian J. Hayes, and Michael A. Jackson. Deriving specifications for systems that are connected to the physical world. In Cliff B. Jones, Zhiming Liu, and Jim Woodcock, editors, *Formal Methods and Hybrid Real-Time Systems: Essays in Honour of Dines Bjørner and Zhou Chaochen on the Occasion of Their 70th Birthdays*, volume 4700 of *Lecture Notes in Computer Science*, pages 364–390. Springer Verlag, 2007.

[JHLJ10] Michael Jastram, Stefan Hallerstede, Michael Leuschel, and Aryldo G Russo Jr. An approach of requirements tracing in formal refinement. In *VSTTE*. Springer, 2010.

[Jon83] C. B. Jones. Specification and design of (parallel) programs. In *Proceedings of IFIP'83*, pages 321–332. North-Holland, 1983.

[Jon03] Cliff B Jones. A formal basis for some dependability notions. In Bernhard K. Aichernig and Tom Maibaum, editors, *Formal Methods at the Crossroads: from Panacea to Foundational Support*, volume 2757 of *Lecture Notes in Computer Science*, pages 191–206. Springer Verlag, 2003.

[Jon10a] Cliff B. Jones. From problem frames to HJJ (and its known unknowns). In Bashar Nuseibeh and Pamela Zave, editors, *Software Requirements and Design: The Work of Michael Jackson*, chapter 16, pages 357–372. Good Friends Publishing Company, 2010.

[Jon10b] Cliff B. Jones. The role of auxiliary variables in the formal development of concurrent programs. In Cliff Jones and Bill Roscoe, editors, *Reflections on the work of C. A. R. Hoare*. Springer, 2010. in press.

[JP10] Cliff B. Jones and Ken G. Pierce. Elucidating concurrent algorithms via layers of abstraction and reification. *Formal Aspects of Computing*, online first, 2010.

[LIR10] I. Lopatkin, A. Iliasov, and A. Romanovsky. On Fault Tolerance Reuse during Refinement. In *Proceeding of the 2nd International Workshop on Software Engineering for Resilient Systems (SERENE 2010), London*. IEEE Computer Society, 2010.

[Maz09a] M. Mazzara. Deriving specifications of dependable systems: toward a method. In *Proceedings of the 12th European Workshop on Dependable Computing (EWDC 2009)*, 2009.

[Maz09b] M. Mazzara. Different perspectives for reasoning about problems and faults. Technical Report CS-TR No. 1151, School of Computing Science, University of Newcastle, April 2009.

[MB10] M. Mazzara and A. Bhattacharyya. On Modelling and Analysis of Dynamic Reconfiguration of Dependable Real-Time Systems. In *Proceedings of DEPEND 2010 – Third International Conference on Dependability*, pages 173–181, 2010.

[MM09] Annabelle McIver and Carroll C. Morgan. *Sums and Lovers:* case studies in security, compositionality and refinement. In Ana Cavalcanti and Dennis Dams, editors, *FM 2009: Formal Methods, Second World Congress, Eindhoven, The Netherlands, November 2-6, 2009. Proceedings*, volume 5850 of *Lecture Notes in Computer Science*, pages 289–304. Springer, 2009.

[Mor06] Carroll Morgan. *The Shadow Knows:* refinement of ignorance in sequential programs. In Tarmo Uustalu, editor, *MPC*, volume 4014 of *Lecture Notes in Computer Science*, pages 359–378. Springer, 2006.

[Mor09] Carroll Morgan. The Shadow Knows: Refinement and security in sequential programs. *Sci. Comput. Program.*, 74(8):629–653, 2009.

[NDS09] M. Neovius, F. Degerlund, and K. Sere. Service Inter-Dependency in the Action System Formalism. In *Proceedings of Third Workshop on Formal Languages and Analysis of Contract-Oriented Software, FLACOS'09, Research Report No. 385*, pages 45–53. University of Oslo, Department of Informatics, 2009.

[Pro10] ProR. ProR Requirements Engineering Platform, 2010. Documentation at http://pror.org.

[Req10] ReqIF. ReqIF 1.0 Request for Comments (Object Management Group), 2010. ReqIF 1.0 RFC at http://www.omg.org/cgi-bin/doc?mantis/10-03-07.pdf.

[ROD10] RODIN. RODIN modularisation plug-in, 2010. Documentation at http://wiki.event-b.org/index.php/Modularisation_Plug-in.

[SB09] Christoph Sprenger and David Basin. Developing security protocols by refinement. Technical Report 625, Department of Computer Science, ETH Zurich, June 2009. `http://www.inf.ethz.ch/research/disstechreps/techreports`.

[Sil] Renato Silva. Parallel Composition using Event-B. `http://wiki.event-b.org/index.php/Parallel_Composition_using_Event-B`.

[TTL10a] Anton Tarasyuk, Elena Troubitsyna, and Linas Laibinis. Augmenting Formal Development of Control Systems with Quantitative Reliability Assessment. In *Int. Workshop on Software Engineering for Resilient Systems - SERENE*. ACM, 2010.

[TTL10b] Anton Tarasyuk, Elena Troubitsyna, and Linas Laibinis. From Formal Specification in Event-B to Probabilistic Reliability Assessment. In *3rd International Conference on Dependability (DEPEND 2010)*. IEEE Computer Press, 2010.

[TTL10c] Anton Tarasyuk, Elena Troubitsyna, and Linas Laibinis. Towards Probabilistic Modelling in Event-B. Technical report, October 2010.

[WLBF09] J. Woodcock, P. G. Larsen, J. Bicarregui, and J. Fitzgerald. Formal Methods: Practice and Experience. *ACM Computing Surveys*, 41(4), Oct 2009.

[YBR10] Sanaz Yeganefard, Michael Butler, and Abdolbaghi Rezazadeh. Evaluation of a guideline by formal modelling of cruise control system in Event-B. In César Muñoz, editor, *Proceedings of the Second NASA Formal Methods Symposium (NFM 2010), NASA/CP-2010-216215*, pages 182–191, Langley Research Center, Hampton VA 23681-2199, USA, April 2010. NASA.