Electronic Communications of the EASST
Volume 46 (2011)

Proceedings of the
11th International Workshop on
Automated Verification of Critical Systems
(AVoCS 2011)

Mixing Formal and Informal Model Elements for Tracing Requirements

Michael Jastram, Stefan Hallerstede, Lukas Ladenberger

15 pages

Guest Editors: Jens Bendisposto, Cliff Jones, Michael Leuschel, Alexander Romanovsky
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Mixing Formal and Informal Model Elements for Tracing Requirements

Michael Jastram[1], **Stefan Hallerstede**[2], **Lukas Ladenberger**[3]

[1] michael@jastram.de
[2] stefan.hallerstede@wanadoo.fr
[3] lukas.ladenberger@gmx.de

Institut für Softwaretechnik und Programmiersprachen
Heinrich-Heine Universität Düsseldorf, Germany

Abstract: Tracing between informal requirements and formal models is challenging. A method for such tracing should permit to deal efficiently with changes to both the requirements and the model. A particular challenge is posed by the persisting interplay of formal and informal elements.

In this paper, we describe an incremental approach to requirements validation and systems modelling. Formal modelling facilitates a high degree of automation: it serves for validation and traceability. The foundation for our approach are requirements that are structured according to the WRSPM reference model. We provide a system for traceability with a state-based formal method that supports refinement. We do not require all specification elements to be modelled formally and support incremental incorporation of new specification elements into the formal model. Refinement is used to deal with larger amounts of requirements in a structured way.

We provide a small example using Problem Frames and Event-B to demonstrate our approach.

Keywords: Requirements, WRSPM, Event-B, Rodin, ProR

1 Introduction

We describe an approach for incrementally building a formal model from structured informal specifications providing a means of requirements validation. Our approach does not require all specification elements to be modelled formally, and the resulting system description provides traceability to both formal and informal model elements. The traceability allows us to detect which requirements are affected if the system implementation changes, and vice versa. Most elements of the structured specification are still stated in natural language. Our aim is to increase the confidence that the formal model represents what has been specified, and to ensure that specification elements that do not have a formal representation are validated at a different stage of the development by informal reasoning and tracing.

We identified the WRSPM reference model [GJGZ00] as the foundation for the informal structured specification. Many concrete approaches are consistent with this reference model, e.g., [Jac01, PM95]. A specification following the WRSPM approach can still be understood by stakeholders, while providing a good foundation for formalisation. These approaches define *phenomena* which describe the state space of the system and its environment, as well as artefacts that represent constraints on the state space and the state transitions. This structure makes a traceability to a state-based formalism doable.

A distinguishing feature of our approach is the incremental modelling of the specification using refinement, which the chosen formalism must support. Once modelled formally, the potential for automated verification is high. This is particularly useful for change management and requirements evolution, which are both important aspects for real-world systems. Also, we allow specification elements without formal representation. Those elements must be justified informally using techniques suggested in [Jac01], for instance.

1.1 Structure of this Paper

In the remainder of this section, we will provide a brief foundation of requirements and specifications, as well as state-based modelling. In Section 2 we present our main thesis, the traceability between formal and informal specification and model. We deepen the aspect of formal refinement in Section 3.

In Section 4, we provide a small example to demonstrate various aspects of our approach. The example uses the Problem Frames approach and the Event-B formal method.

We are actively working on tool support and present our progress in Section 5.

After describing some of the related work in Section 6, we conclude in Section 7, which also contains an outlook on future work.

1.2 Requirements and Specification

Our approach is based on WRSPM by Gunter et. al. [GJGZ00]. WRSPM is a reference model for applying formal methods to the development of user requirements and their reduction to a behavioural system specification.

WRSPM distinguishes between artefacts and phenomena. Phenomena describe the state space (and state transitions) of the domain and system, while artefacts represent constraints on the state space and the state transitions. The artefacts are broadly classified into groups that pertain mostly to the system versus those that pertain mostly to the environment. These are:

Domain Knowledge (W) describes how the world is expected to behave.
Requirements (R) describe how we would like the world to behave.
Specifications (S) bridge the world and the system.
Program (P) provides an implementation of S.
Programming Platform (M) provides an execution environment for P.

We distinguish phenomena by whether they are controlled by the system (belonging to set s) or the environment (belonging to set e). They are disjoint ($s \cap e = \varnothing$), while taken together, they represent all phenomena in the system ($s \cup e =$ “all phenomena”). Furthermore, we distinguish them by visibility. Environmental phenomena may be visible to the system (belonging to e_v) or hidden from it (belonging to e_h). Correspondingly, system phenomena belonging to s_v are visible to the environment, while those belonging to s_h are hidden from it. These classes of phenomena are mutually disjoint.

The distinction between environment and system is an important one; omitting it can lead to misunderstandings during the development. It is sometimes regarded as a matter of taste or convenience where the boundary between environment and system lies, but it has a profound effect on the problem analysis. It clarifies responsibilities and interfaces between the system and the world and between subsystems. If we require ourselves to explicitly make that distinction, we can avoid many problems at an early stage.

W and R may only be expressed using phenomena that are visible in the environment, which is $e \cup s_v$. Likewise, P and M may only be expressed using phenomena that are visible to the system, which is $s \cup e_v$. S has to be expressed using phenomena that are visible to both the system and the environment, which is $e_v \cup s_v$.

Once a system is modelled following WRSPM, a number of properties can be verified with regard to the model, one being *adequacy with respect to S*:[1]

$$\text{FOR ALL } e\ s,\ W \text{ AND } S \text{ IMPLY } R \qquad \text{(Adequacy)}$$

This simply says that the specification constrains the world such that the requirements are realized. Obviously we are not interested in the trivial solution to (Adequacy), meaning that no e and s exist to satisfy (Adequacy).

We demonstrate our ideas using Problem Frames [Jac01], which is a concrete approach to software requirements analysis that is a manifestation of the WRSPM reference model. The central element in problem frames is the problem diagram that consists of exactly one machine domain, designed domains and given domains.

1.3 State-Based Modelling and Refinement

Our approach could be used with a wide range of formal methods for state-based modelling that have an associated notion of refinement. We find state-based formalisms such as ASM , VDM , TLA+ or Event-B particularly suited because they permit straightforward specification of state, state invariants and state transitions for modelling dynamic behaviour. In this paper, we focus on state-based modelling and provide an example using Event-B, which we introduce in Section 3.1. Event-B is suitable for discussing the example that we introduce in Section 4. Using Event-B we can also discuss limitations of requirements tracing: not all requirements can be formalised within the core Event-B formalism. Formal and informal reasoning need to be combined in a sensible way. The boundary of formalisation in the example is given by temporal and real time properties. We have intentionally chosen a boundary that could be moved by using another formal method or extending Event-B because we think it is not fixed and may change depending on project characteristics. It also serves to illustrate that the boundary may be moved as a development progresses. We think of modelling and requirements validation as an incremental process: we permit the boundary to be moved as need arises.

We take advantage of the concept of refinement which is supported by Event-B. Other notions of refinement could be used without changing the approach fundamentally. Our approach allows us to account for additional requirements at later refinement stages, thereby providing a structuring mechanism for the introduction of requirements into the formal model.

[1] We present “informal formulas” as used in WRSPM in a textual way to set then apart from “ formal formulas” as used in Event-B.

2 From Formal to Informal and Back

The requirements engineering process can be broken down into requirements specification, system modelling, requirements validation and requirements management [Wie03].

We will briefly describe these process steps (shown in Figure 1) and how they relate to the work in this paper.

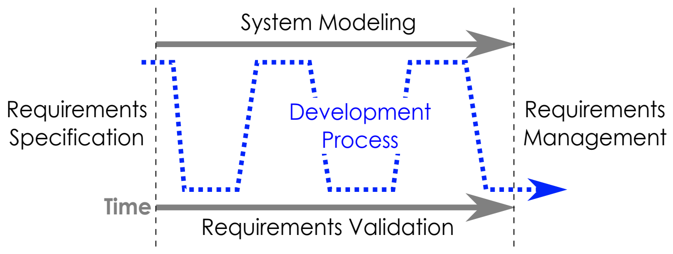


Figure 1: The Incremental Development Process

Requirements Specification – The requirements are structured according to the approach of choice, resulting in a specification that follows the WRSPM reference model. In the example that we introduce in Section 4, we use the Problem Frames approach.

System Modelling – The objective of this phase is the formal modelling of elements. Not all elements need to be modelled formally, which is one distinguishing feature of our approach. In the example in Section 4, we use Event-B.

Requirements Validation – The validation of the requirements is a central aspect of this paper and is described in detail in Section 2.1.

Requirements Management – In practice, a specification is never "done". The ongoing work includes change management and requirement evolution. These tasks are supported by our approach.

These tasks, including elicitation, analysis and negotiation, are performed in parallel. We do not want to create the impression that this is a sequential process.

2.1 Requirements Validation

System modelling provides us with partly formalised elements as described by the requirements. We think of system modelling as an incremental process where more and more is formalised. However, we do not assume that necessarily everything is formalised. The methodology we propose allows for a mixture of formal and informal proof as a means of validation. As a consequence of frequent incremental changes we need effective support for tracing requirements: formal models change as they incorporate increasing detail, requirements change as a consequence of the validation itself. The transition to requirements management is considered fluent and the same techniques of traceability are applied.

Demonstrating (Adequacy) now involves dealing with formal and informal elements. In the following, we designate by Rf the formal requirements, by Wf the formal domain properties and by Sf the formal specification elements. The difference $R \setminus Rf$ of all requirements and formal requirements gives the informal requirements Ri, similarly for informal domain properties Wi and informal specification elements Si.

For the formal elements we can formally verify that

$$\forall e\ s \cdot Wf \land Sf \Rightarrow Rf\ , \tag{1}$$

assuming that sufficient of W and S have been formalised to cover Rf. For informal elements we allow informal arguments, for instance, of the kind used in the problem frames approach [Jac01] or not formalised mathematical proofs. Doing this, we show:

$$\text{FOR ALL } e\ s,\ \ W \text{ AND } S \text{ IMPLY } Ri\ . \tag{2}$$

We permit also using formal elements in the antecedent of (2) but only formal elements in the antecedent of (1). As many critical requirements as possible should be validated formally, giving high assurance of their satisfaction. Relying on formally verified facts in informal justification will also improve their quality.

2.1.1 Formal Tracing

To formalise artefacts A they need to be of a form that can be "translated" into a formula F so that we can state

$$A \text{ EQUIVALES } F\ . \tag{3}$$

This makes tracing from F to A and vice versa trivial. Formal proofs of (1) can provide information about which formal artefacts are used in order to validate specific requirements. Among others, this has been implemented in the proof support of the Rodin tool [ABH+10]. If formal artefacts $F_1, \ldots, F_k$ have been used to prove formal requirement Rf_n from $Wf \land Sf$, then we know that a change of the informal requirement R_n that equivales Rf_n affects the informal artefacts A_1, $\ldots$, A_k. The formal model provides a way to validate requirements rigorously and an efficient way to trace dependencies between informal artefacts. The latter is crucial for the maintenance of large numbers of requirements occurring in industrial practice. Support by proof tools means that this tracing can be automated to a large degree.

2.1.2 Informal Tracing

Artefacts that are not formalised can still be traced but the dependencies can only be checked manually by inspecting informal arguments. Changes of involved artefacts require corresponding human intervention. A known technique to limit the impact of changes is the identification of a *satisfaction base* [KJ10] for each informal artefact of Ri. A satisfaction base for a requirement R_n consist of those artefacts from S and W that are sufficient to justify it. Using the concept of a satisfaction base, (2) can be rephrased as

$$\text{FOR ALL } e\ s,\ \ SB(R_n) \text{ IMPLY } R_n\ . \tag{4}$$

where $SB(R_n)$ is a subset of W and S, representing a satisfaction base for the given requirement. The satisfaction base is used in the informal justification and for tracing dependencies, similarly to formal tracing. However, possibilities for automation are very limited. Also note that there may be multiple satisfaction bases.

3 Formal Refinement

Formula (1) can grow very large for a complex model. This can make it very difficult to verify any interesting property but also to compute a sufficiently small set of formal artefacts that are used to verify specific formal requirements Rf_n. Formal refinement alleviates this problem by introducing parts of the overall model in small increments. The original WRSPM approach sketch a notion of implementation based on the program P and the programming platform M:

$$\text{FOR ALL } e\ s,\ W \text{ AND } P \text{ AND } M \text{ IMPLY } R\ . \tag{5}$$

This can be achieved by relying on implication for implementation (e.g., [HH98, GJGZ00]),

$$\text{FOR ALL } e_v\ s,\ P \text{ AND } M \text{ IMPLY } S \tag{6}$$

providing a simple notion of refinement in a predicative specification style. Instead of formalising the refinement notion (6) we prefer a notion based on discrete transition systems that permits more direct specification of dynamic aspects of a model. For the purposes of this article we do not consider details of M such as the targeted programming language. We consider S as a collection of invariants and transitions of a discrete transition system which we specify by means of Event-B [Abr10].

3.1 Event-B Machines

Event-B is a state-based modelling method whose models are characterised by *proof obligations*. Proof obligations serve to verify properties of the models.

We only provide a brief summary of simplified Event-B in terms of proof obligations. A complete description can be found in [Abr10]. Variables v define the state of a machine. They are constrained by invariants $I(v)$. Possible state changes are described by means of events.

3.2 Event-B Proof Obligations

In Event-B two main properties are proved about formal models: consistency and refinement.

Consistency means that the invariant $I(v)$ is established by the initialisation and maintained by all other events of the machine.

Refinement links abstract events to concrete events aiming at the preservation of properties of the abstract event when it is replaced by the concrete event.

Proof obligations can now be created that ensure that invariants are never violated, both in the machine where they appear and in refinement. Additional proof obligations for well-definedness or deadlocks can be constructed mechanically.

3.3 Tracing of Requirements with Event-B

The Event-B model contains formal artefacts as indicated by (3). The domain properties Wf and specification elements Sf can be represented by means of events and invariants. By consistency and refinement we get a collection of invariants IA that are preserved by all events EA. We can now partition events and invariants according to the artefacts they represent: $IA = IW \cup IS$ and $EA = EW \cup ES$. Making this distinction is standard in the Event-B method.[2] To fit into the shape

[2] Using model decomposition we could now decompose the two parts and focus on the refinement of the sub-model consisting of IS and ES. The interface to the other sub-model would act as a contract guaranteeing overall consistency.

of WRSPM adequacy we consider the before-after predicates of all events and identify $Wf = IW \wedge BA(EW)$ and $Sf = IS \wedge BA(ES)$, where $BA(EE)$ yields the disjunction of the before-after predicates of the events EE. In formal refinement Pf the formal program is usually considered a subset of Sf that is being gradually constructed during refinement. After some refinement steps we have $Pf = IS \wedge IP \wedge BA(EP)$ where the events EP are refinements of the events ES. Hence, $Pf \Rightarrow Sf$ by choosing suitable witnesses, obtaining the formal counterpart of (6). We have identified the formal domain properties Wf, specification element Sf and program Pf.

We can now turn to the formal requirements Rf, formal adequacy (1) and the formalised (5) not taking account of the programming platform M:

$$\forall e\ s \cdot Wf \wedge Pf \Rightarrow Rf\ . \tag{7}$$

Assuming we already have verified (1), adequacy of the implementation (7) follows by the discussion of the preceding paragraph, using

$$\forall s \cdot Pf \Rightarrow Sf\ . \tag{8}$$

Refinement allows this to be applied incrementally to deal with small more manageable sets of artefacts at each refinement step. Gradually, the set of satisfied refinements is extended until all requirements are covered,

$$\varnothing = Rf^0 \subseteq Rf^1 \subseteq Rf^2 \subseteq \ldots \subseteq Rf^n = Rf\ , \tag{9}$$

where the Rf^i correspond to the refinement steps of the model. Most of these refinement steps will involve the domain properties and specification elements:

$$\forall e\ s \cdot Wf^{i+1} \wedge Sf^{i+1} \Rightarrow Wf^i \wedge Sf^i\ . \tag{10}$$

Refinement steps for implementing the program will usually be less related to requirements. The refinement method, however, does not make a particular distinction between the two uses of refinement. Each refinement step can be used to verify adequacy of the specification gradually:

$$Wf^i \wedge Sf^i \Rightarrow Rf^i \setminus Rf^{i-1}\ . \tag{11}$$

Refinement theory guarantees that adequacy validated in earlier refinement steps is preserved. After n refinement steps (1) is verified.

Formula (11) suggests a method of stepwise tracing of requirements following the refinements. Often requirements can be identified with invariants, event guards or actions. In this case (11) holds trivially. Sometimes theorems can be stated [HL09] that are implied by the invariants. In this article we limit tracing to this level. However, this is not a fundamental limitation of the approach. For instance, one could also permit temporal formulas derived from $Wf \wedge Sf$ as supported by TLA+. Some of TLA+ is also implemented in the ProB tool [LB08] that has been integrated with the Rodin tool that we use. But for this article we contend ourselves with a less expressive notation relying only on invariants and possible transitions.

Problem frame diagrams do not use refinement, but techniques of decomposition like projection. They serve for structuring large sets of requirements. They correspond to the last refined model just before turning to implementation (by means of P). The problem frame diagram will always contain the entire set of formal and informal requirements R. We do not intend to extend the idea of refinement from Event-B to problem frames in this paper.

4 Example: A Traffic Light Controller

We are going to demonstrate the approach presented here by creating the model of a traffic light system that allows pedestrians to cross a street. We already introduced this example in [JHLJ10]. The system consists of two traffic lights for pedestrians (one on each side of the street), two corresponding traffic lights for the cars, and push buttons for the pedestrians to request a green light for crossing the street.

We consider this example useful, because it is simple enough to understand, but complex enough to be interesting. Further, the example concerns state (which we model formally) as well as real-time (which we specify informally), allowing us to demonstrate the mixing of formal and informal modelling elements.

4.1 Requirements Specification

Following our approach, we would apply a specification approach of choice, in this case Problem Frames. This may lead to a the problem diagram shown in Figure 2. The Problem Frames diagram is incomplete. For instance, information regarding the temporal properties of the system are missing. This is by design, as the problem diagram only depicts the contextual aspects of the model and their relationships in the form of shared phenomena. The textual representation is still the central repository for all information regarding the system. This leads to a new natural language specification, with a vocabulary that is managed in a separate glossary.

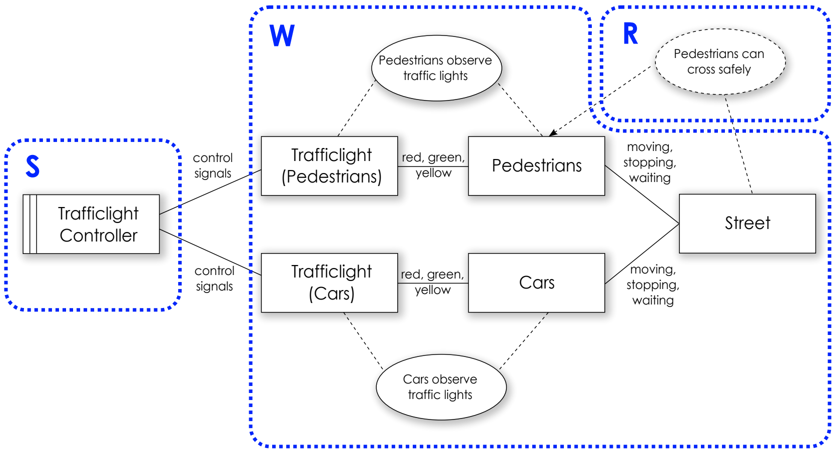

W
R
S
Pedestrians observe traffic lights
Pedestrians can cross safely
control signals
Trafficlight (Pedestrians)
red, green, yellow
Pedestrians
moving, stopping, waiting
Trafficlight Controller
Street
control signals
Trafficlight (Cars)
red, green, yellow
Cars
moving, stopping, waiting
Cars observe traffic lights

Figure 2: A simplified Problem Frames diagram for the traffic light problem

Note that it can be useful to introduce an informal notion of refinement already in the textual description of the system to structure it. We see that in the description of the traffic light states, that are sometimes referred to abstract as the abstract *stop* and *go*, and sometimes as the concrete colours *red*, *yellow* and *green*. We can take advantage of this in the modelling phase by establishing abstract properties that are simple and easy to trace. The refinement concept of Event-B allows us to introduce the concrete colours later on, while preserving the original properties (assuming correct data refinement), as demonstrated in Section 4.3.

The new specification is already more precise than the original requirements, while still comprehensible by the stakeholders. We already identified items as R, W and S. This makes it easier to reason about the model. It also allows us to identify the proper role for validating or justifying each artefact: Stakeholders are concerned with R, domain experts with W and designers with S.

R-2.1	*Pedestrians* can cross safely. They are crossing when they are not *waiting*.
W-2.1	*Pedestrians* observe the traffic lights (tl_{peds}). This means that they may move (*moving*) when the traffic lights allows them to *go*. Upon indicating *stop*, they finish moving (*stopping*) and then wait (*waiting*).
S-2.2	tl_{cars} must wait for a certain time ($delay_{cars}$) before switching to *go* after tl_{peds} turned to *stop*.
S-2.3	$delay_{peds}$ is 3 seconds (± 100ms).

4.2 System Modelling

We decided to use the Event-B formalism (Section 3.1), making it easier to model some aspects of the model and more tricky to model others. In particular, it is easy to express safety properties like R-2.1, more difficult to express state transition properties like S-2.2, and almost impossible to express real-time properties like S-2.3.

Following the incremental approach described in Section 2, we start with the safety requirement R-2.1, for which a state-based formalism like Event-B is well-suited.

$$Pedestrians \neq waiting \Rightarrow Cars = waiting \tag{12}$$

Not all properties can be modelled as easily as R-2.1. For instance, the behaviour of pedestrians (W-2.1) cannot be represented by an invariant. Instead, we can model it according to the approach described in Section 3.3 by representing it as a before-after predicate of an event. The property W-2.1 doesn't have the proper granularity for this approach, so we rewrite it to specify each transition separately. This rewrite is part of the incremental specification process, and the result must be validated with the domain experts.

W-2.1a	*Pedestrians* that are *moving* can only change their state to *stopping*.
W-2.2b	*Pedestrians* that are *stopping* can only change their state to *waiting*.
W-2.1c	*Pedestrians* that are *waiting* can only change their state to *moving*.
W-2.1d	*Pedestrians* may only change to *moving* if tl_{peds} indicates *go*.
W-2.1e	If tl_{peds} indicates *stop*, then *Pedestrians* must change to *stopping* if they are *moving* and change to *waiting* if they are *stopping*.

Rewritten like this, it can be modelled in Event-B as follows:

```
Event  $peds\_moving\_to\_stopping \widehat{=}$
    when
        $\mathtt{W-2.1a:}\ Pedestrians = moving$
    then
        $\mathtt{W-2.1a:}\ Pedestrians := stopping$
Event  $peds\_stopping\_to\_waiting \widehat{=}$
    when
        $\mathtt{W-2.1b:}\ Pedestrians = stopping$
    then
        W−2.1b : $Pedestrians := waiting$
Event   $peds\_waiting\_to\_moving \mathrel{\widehat{=}}$
    when
        W−2.1c : $Pedestrians = waiting$
        W−2.1d : $tl\_peds = go$
    then
        W−2.1c : $Pedestrians := moving$
```

Note how we could establish a clear traceability according to (3). The exception is W-2.1e, which is difficult to model in Event-B. Event-B allows us to enforce that something does *not* happen (via a guard), but difficult to guarantee that something does happen (implying that all events except one are disabled). The missing traceability to W-2.1e reminds us that this property must be justified outside this formal model. This could be done by reasoning, testing, or with a different formalism like temporal logic.

This justification may be invalidated if the source or target of the traceability relationship changes. Thus, it has to be verified after each such change. A tool may support this by invalidating that relationship if either of the elements involved changes.

The reader may have noticed that the above represents a state machine. It could be useful to develop an approach specific to state machines.

4.3 Data Refinement

In Section 3.2, we described how consistency is maintained across refinement levels. We will demonstrate this concept by showing how the traffic light states *stop* and *go* are transformed via data refinement into *red*, *yellow* and *green*.

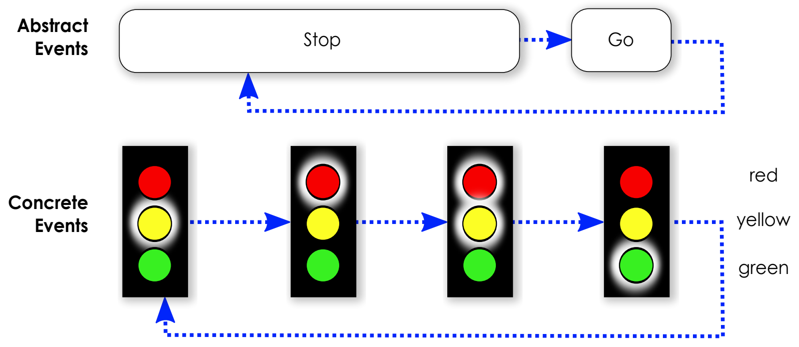


Figure 3: Data Refinement of the Traffic Light States

Data refinement allows us to state abstract properties in a concise way, while the implementation details are addressed later. This allows us to reason about some fundamental properties. Consider S-2.1 as an example of such a property. By arguing simply about *stop* and *go*, the safety property can be stated in a very concise way. The detail on how *stop* and *go* are realised (through colours), can be provided later. Carrying the notion of refinement to the requirements allows us to write more concise requirements: In this case, we can separate the safety requirement from the actual representation of traffic light states, which is also a requirement, but a different one.

We can model S-2.1 formally as follows:

$$\neg(tl_{peds} = go \land tl_{cars} = go) \tag{13}$$

The definition of *stop* and *go* in terms of colours leads to the following gluing invariant that can be introduced in a new refinement:

$$tl_{peds} = go \Leftrightarrow colors_{peds} = \{green\} \tag{14}$$

Introducing (14) into the model results in non-discharged proof obligations, as the newly introduced gluing invariant will be violated without any further modifications. The abstract events that control the traffic light's *stop* and *go* states must also be refined into concrete events that cycle through the corresponding colour states, as shown in Figure 3.

The refinement will take on a similar form as the Event-B shown in Section 4.2, where each state transition corresponds to one event. The proof obligations will ensure that the safety requirement (13) is not violated once they are all discharged, assuming that the gluing invariant (14) is modelled correctly. Discharging all proof obligations will require additional guards.

4.4 Adding Requirements with Refinement

Another application of refinement is the gradual inclusion of formal requirements into subsequent refinements, as hinted at in (9). In the traffic light example, this can be demonstrated by adding a push button for the pedestrians, allowing them to request crossing the street.

Table 1 shows the structured requirements and their formal representations:

R-2.2	*Pedestrians* can *request* to cross any time.
S-2.6	Upon switching of *tl_peds* from *go* to *stop*, the *request* is reset.
S-2.7	*Pedestrians* must not wait longer than 60 seconds for permission to cross after issuing the *request*.

Table 1: Requirement and Specification for allowing Pedestrians to Request Crossing the Street

These two properties can be incorporated into the model in a separate refinement with a new event and the extension of an existing event with a straightforward traceability, as shown in the following:

```
**Event** request_crossing ≙
    **when**
        R−2.2 : request := TRUE
    **end**
**Event** set_tl_peds_go ≙
**extends** set_tl_peds_go
    **when**
        S−2.6 : request := FALSE
```

The requirement S-2.7 cannot be modelled formally as stated. This informal artefact simply has to be verified outside the formal model. We could break down S-2.7 further to model some aspects formally (e.g. by introducing a "tick" interval). Our approach could handle this, but we omitted this for brevity.

5 Tool Support

A tool supporting this approach would have to provide a mechanism to mark informal artefacts to be marked as "justified", and a place to write this justification down. Further, all *Ri* would have to be marked for re-justification, as soon as any *W* or *S* changes.

We developed a platform for requirements engineering called ProR[3] [JG11]. While the tool can be used stand-alone, we designed it with the goal to ease the integration of natural language requirements and formal models.

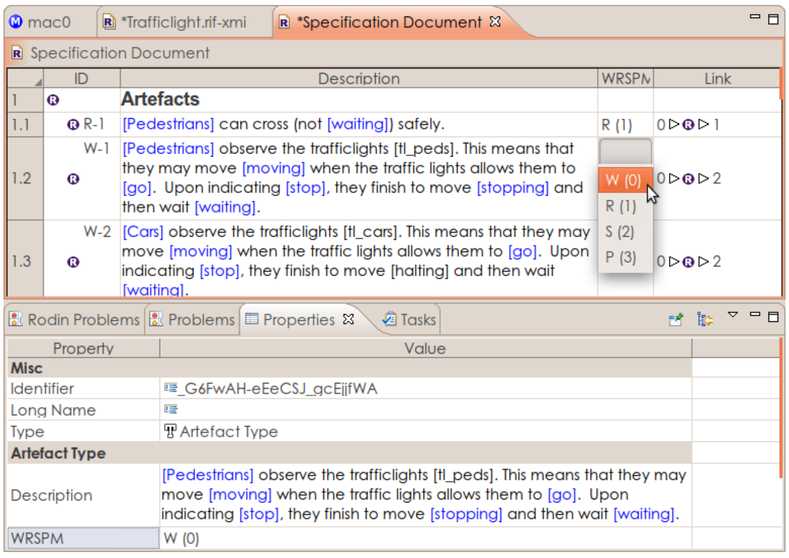
ProR is based on the Requirements Interchange Format RIF/ReqIF [OMG11]. RIF was created in 2004 by the "Herstellerinitiative Software", a body of the German automotive industry that oversees vendor-independent collaboration. In 2010, the Object Management Group (OMG) took over the standardisation process and released a new version of the standard under the name ReqIF. Our tool environment is currently based on RIF 1.2, support for ReqIF 1.0 is planned.

ProR is part of the Requirements Modeling Framework, which is an official Eclipse Project.

ProR can be installed directly into Rodin. A tight integration can be achieved with plugins that access both the Rodin and ProR data structures.

We created a plugin that allows us to manage the vocabulary of the natural language requirements as Event-B models. A user would then be empowered to work on requirements and Event-B models in the same environment. The plugin tracks all names in the model (variables, constants, etc.) and allows them to be highlighted, as shown in Figure 4. Traces between the model

[3] http://www.pror.org



	ID	Description	WRSPM	Link
1		**Artefacts**		
1.1	R-1	[Pedestrians] can cross (not [waiting]) safely.	R (1)	0 ▷ Ⓡ ▷ 1
1.2	W-1	[Pedestrians] observe the trafficlights [tl_peds]. This means that they may move [moving] when the traffic lights allows them to [go]. Upon indicating [stop], they finish to move [stopping] and then wait [waiting].	W (0)	0 ▷ Ⓡ ▷ 2
1.3	W-2	[Cars] observe the trafficlights [tl_cars]. This means that they may move [moving] when the traffic lights allows them to [go]. Upon indicating [stop], they finish to move [halting] and then wait [waiting].		0 ▷ Ⓡ ▷ 2

Property	Value
Misc	
Identifier	_G6FwAH-eEeCSJ_gcEjjfWA
Long Name	
Type	Artefact Type
Artefact Type	
Description	[Pedestrians] observe the trafficlights [tl_peds]. This means that they may move [moving] when the traffic lights allows them to [go]. Upon indicating [stop], they finish to move [stopping] and then wait [waiting].
WRSPM	W (0)

Figure 4: Integration of WRSPM-structured artefacts and formal Event-B elements.

and the requirements can be created via drag and drop. These can then be typed and annotated. In addition, model elements can be shown in-line with the requirements, allowing the user to see the requirement and the associated model element side-by-side. This gives users the flexibility to adapt the traceability and its presentation to their work flow.

The user can then establish the traceability manually, while the colour highlighting in the requirements text helps to relate the requirement to the linked model elements. Once an iteration is complete, the user can perform a manual validation and record the status of the validation as an annotation in the trace.

The plugin is built using the Eclipse EMF technology [4]. This allows us to "hook" code into the models to perform various tasks. Depending on the specification approach used, we could provide validators to ensure consistency according to the approach taken.

The application of the tool is shown in Figure 4, where the elements from the formal model are highlighted in the requirement text. We also see how a classification of elements can be performed, in this example following WRSPM. The desired artefact type is selected from a drop-down directly in the editor.

The *Properties View* in the lower pane shows additional information regarding the selected element.

The right column shows the number of incoming and outgoing links, providing a quick summary of each element's traceability. These links can be unveiled.

For links, the rightmost column contains the link target. Selecting it shows the target's properties in the Property View. In the screenshot we see that the link target is the event *stopping_peds*. As it is selected, the Property View shows its attributes, including the event itself. This is a reference to the model, not a copy of the event.

The tool is currently in a prototypical state and is actively developed. Specifically, it currently support the manual creation of links and colour highlighting. Also, the Event-B integration at this time is relatively coarse (serving as a proof of concept), as it relies on the EMF-support in Rodin. This means that no traces to sub-expressions are possible with the current prototype.

We envision a tool that identifies unaccounted requirements and model elements, and that invalidates traces when related model elements change, as well as change impact analysis.

6 Related Work

The issue of traceability has been analysed in depth by Gotel et. al. [GF94]. Our research falls into the area of post-requirements specification traceability.

Abrial [Abr06] recognises the problem of the transition from informal user requirements to a formal specification. He suggests to construct a formal model for the user requirements, but acknowledges that such a model would still require informal requirements to get started. He covers this approach in [Abr10].

The WRSPM reference model [GJGZ00] was attractive because it allowed us to discuss the specification in general terms, while still being meaningful in the context of a specific approach like Problem Frames [Jac01] or the functional-documentation model [PM95].

There have been successful attempts in applying Problem Frames and Event-B together. In [LGG+10], the authors show how these are being applied to an industrial case study. In contrast

[4] Eclipse Modelling Framework, http://www.eclipse.org/emf/

to our approach, only requirements that were actually modelled formally were included in the specification in the first place.

There are approaches spanning from requirements to formal model, a well-known one being KAOS [DDML97]. But rather than allowing informal elements that are omitted from the formal model, it provides so-called “soft-goals” that are broken down into requirements that can still be modelled formally.

Reveal [Pra03] is an engineering method based on Michael Jackson’s “World and the Machine” model. There are a lot of similarities to our approach, including the acknowledgement of requirements that are not part of the formal model. However, Reveal is more of a process description of the overall requirements engineering process. Therefore it could be quite attractive to apply the Reveal process with the approach described here.

7 Conclusion

In this paper, we presented an approach for incrementally building a formal model from structured informal requirements. Our approach supports partial formal modelling and provides traceability for both formal and informal specification elements. This approach allows us to take advantage of the formal model regarding automated verification, while providing a systematic (albeit manual) approach to validation of the remaining specification elements.

We demonstrate our ideas on a specification and model of a traffic light system. While this is arguably a teaching example, it contains examples of specification elements that are challenging in formal modelling and demonstrates how these can be addressed.

We believe that tool support is a crucial element for such an approach to work and presented an integration of the ProR platform for requirements engineering and the Rodin platform for Event-B modelling to support our approach.

Future Work. We will continue investigating different specification methods. While we find WRSPM useful, it is a reference framework that is not intended to be applied as is. We have experimented with Problem Frames, which are useful but does not match well with our approach to refinement (based on Event-B).

We will explore the suitability of Event-B for modelling bigger specifications with our approach, if possible real-world examples.

Last, we will continue our work on tool support.

Acknowledgements. The work in this paper is partly funded by Deploy[5]. Deploy is a European Commission Information and Communication Technologies FP7 project.

Bibliography

[ABH$^+$10] J.-R. Abrial, M. J. Butler, S. Hallerstede, T. S. Hoang, F. Mehta, L. Voisin. Rodin: An Open Toolset for Modelling and Reasoning in Event-B. *STTT* 12(6):447–466, 2010.

[Abr06] J. Abrial. Formal methods in industry: achievements, problems, future. In *Proceedings of the 28th international conference on Software engineering*. Pp. 761–768. 2006.

[5] http://www.deploy-project.eu

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

[DDML97] R. Darimont, E. Delor, P. Massonet, A. van Lamsweerde. GRAIL/KAOS: An Environment for Goal-driven Requirements Engineering. In *Proc. of the 19th int. conf. on Software engineering*. Pp. 612–613. ACM, 1997.

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

[GJGZ00] C. A. Gunter, M. Jackson, E. L. Gunter, P. Zave. A Reference Model for Requirements and Specifications. *IEEE Software* 17:37–43, 2000.

[HH98] C. A. R. Hoare, He Jifeng. *Unifying Theories of Programming*. Prentice Hall, 1998.

[HL09] S. Hallerstede, M. Leuschel. How to Explain Mistakes. In Gibbons and Oliveira (eds.), *Teaching Formal Methods, Second International Conference, TFM 2009, Eindhoven, The Netherlands, November 2-6, 2009. Proceedings*. Lecture Notes in Computer Science 5846, pp. 105–124. Springer, 2009.

[Jac01] M. Jackson. *Problem frames: analysing and structuring software development problems*. Addison-Wesley/ACM Press, 2001.

[JG11] M. Jastram, A. Graf. Requirements, Traceability and DSLs in Eclipse with the Requirements Interchange Format (RIF/ReqIF). In *Tagungsband des Dagstuhl-Workshop MBEES*. fortiss GmbH, München, 2011.

[JHLJ10] M. Jastram, S. Hallerstede, M. Leuschel, A. G. R. Jr. An Approach of Requirements Tracing in Formal Refinement. In *VSTTE*. Springer, 2010.

[KJ10] E. Kang, D. Jackson. Dependability arguments with trusted bases. In *Requirements Engineering Conference (RE), 2010 18th IEEE International*. P. 262–271. 2010.

[LB08] M. Leuschel, M. Butler. ProB : an automated analysis toolset for the B method. *International Journal on Software Tools for Technology Transfer* 10(2):185–203, 2008.

[LGG$^+$10] F. Loesch, R. Gmehlich, K. Grau, C. Jones, M. Mazzara. Report on Pilot Deployment in Automotive Sector. Technical report D7, DEPLOY Project, 2010.

[OMG11] OMG. Requirements Interchange Format (ReqIF) 1.0.1. 2011.
http://www.omg.org/spec/ReqIF/

[PM95] D. L. Parnas, J. Madey. Functional documents for computer systems. *Science of Computer programming* 25(1):41–61, 1995.

[Pra03] Praxis. Reveal – A Keystone of Modern Systems Engineering. Technical report, 2003.

[Wie03] K. Wiegers. *Software Requirements: Practical Techniques for Gathering and Managing Requirements throughout the Product Development Cycle*. Microsoft Press, Redmond Wash., 2nd edition, 2003.