Refinement Plans for Reasoned Modelling*

Maria Teresa Llano[1], Andrew Ireland[1], and Gudmund Grov[2]

[1] School of Mathematical and Computer Sciences,
Heriot-Watt University, Edinburgh, UK
{mtl4,a.ireland}@hw.ac.uk
[2] School of Informatics, University of Edinburgh, Edinburgh, UK
ggrov@inf.ed.ac.uk

We focus here on a layered style of formal modelling, where a design is developed as a series of abstract models – level by level concrete details are progressively introduced via provably correct *refinement* steps. There are two major approaches to achieving this style of formal modelling:

- firstly, within the *rule-based* approach, a user is restricted to a provably correct set of refinement steps – thereby ensuring the soundness of their development. An example of this style of development is found in [9].
- secondly, within the *posit-and-prove* approach, a user is free to "posit" models, but they are required to formally prove correctness of successive layers of abstraction. Examples of posit-and-prove approaches are VDM [8], B [1] and Event-B [2].

Our aim is to enhance the posit-and-prove approach. Specifically, we have developed a technique called *refinement plans* which automatically generates guidance for users within posit-and-prove formal modelling. Like many approaches to design, whether informal [5] or formal [3], our technique relies upon patterns. The novelty of our refinement plans is that they combine modelling and reasoning patterns, enabling us to computationally exploit the subtle interplay that exists between modelling and reasoning – what we call *reasoned modelling*. Our refinement plans are heuristic in nature, and can be applied *flexibly* during a development.

As mentioned above, a refinement plan combines both modelling and reasoning knowledge. Specifically, a refinement plan is an integration of three complementary components:

$$\textit{refinement plan} = \textit{refinement method} + \textit{proof methods} + \textit{critics}$$

A *refinement method* describes a common pattern of refinement in terms of abstract and concrete models, together with a gluing invariant pattern. For a given refinement pattern, the associated reasoning patterns are represented in terms of *proof methods*, which we borrow from *proof planning* [4]. We are particularly interested in situations where a development breaks down, but which is close enough to a known pattern for our plans to provide user guidance in terms of modelling decisions. To achieve this we use a *critics* style exception handling mechanism, analogous to proof critics [6]. Currently a critic has access to proof failures arising from the Rodin provers, and partial success in terms of the application of refinement and proof methods.

The application of refinement plans is currently a two phase process. Firstly, an evaluation of the occurrence of refinement methods with respect to a given refinement step is performed. Secondly, the results of the evaluation phase are analysed as follows:

* Thanks go to Rob Pooley, Julian Gutierrez, Alan Bundy and members of the Mathematical Reasoning Group at Edinburgh University for their feedback and encouragement with this work. The research reported in this paper is supported by EPSRC grants EP/F037058, EP/H024204, EP/E005713, EP/E035329 and BAE Systems.

- where a complete match to a refinement method is found, and there are no unproven POs, then an instance of the refinement method has been identified.
- where a partial match of a refinement method is found and/or where there are unproven POs, then the associated critics are applied in order to generate guidance as to how the given refinement can be aligned with the plan.

Note that while the failure analysis and guidance generation is automatic, the decision as to whether or not the guidance is actually applied is left to the user. Note also that the use of proof methods, and the analysis of partial success at the level of proof planning within our refinement plans represents future work.

Below we highlight a variety of scenarios where we envisage refinement plans playing a role within formal modelling:

correcting refinements: a flawed refinement step may be overcome by modifications to the abstract and/or concrete models, e.g. guard strengthening, invariant discovery.

layering refinements: an overly complex refinement step can give rise to unproven proof obligations – such failures may be overcome by the introduction of intermediate layers of abstraction.

abstracting refinements: a development which starts at too concrete a level may benefit from guidance as to how to reduce the initial complexity and open up alternative modelling choices.

suggesting refinements: at the fringe of a development, suggesting alternative refinement steps could be beneficial to a users productivity.

increasing proof automation: our refinement plans will enable us to exploit the corresponds between the structure of a refinement and the pattern of proof associated with its verification.

While the ideas that underpin reasoned modelling are generic with respect to posit-and-prove, our initial focus has been on their application within Event-B. Specifically we have implemented refinement plans that demonstrate the value of the first two scenarios highlighted above in a tool called REMO, a prototype Rodin plug-in [7].

References

1. J.-R. Abrial. *The B-Book - Assigning Programs to Meanings.* Cambridge University Press, Aug. 1996.
2. J.-R. Abrial. *Modelling in Event-B: System and Software Engineering.* Cambridge University Press, 2010.
3. J.-R. Abrial and T. S. Hoang. Using Design Patterns in Formal Methods: an Event-B Approach. In *Theoretical Aspects of Computing - ICTAC 2008, 5th International Colloquium, Istanbul, Turkey, September 1-3, 2008. Proceedings*, volume 5160 of *Lecture Notes in Computer Science*, pages 1–2. Springer, 2008.
4. A. Bundy. A science of reasoning. In *Computational Logic: Essays in Honor of Alan Robinson*, pages 178–198. MIT Press, 1991.
5. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1995.
6. A. Ireland. The use of planning critics in mechanizing inductive proofs. In *LPAR*, volume 624 of *Lecture Notes in Computer Science*, pages 178–189. Springer, 1992.
7. A. Ireland, G. Grov, M. Llano, and M. Butler. Reasoned modelling critics: Turning failed proofs into modelling guidance. *Heriot-Watt University. Technical report HW-MACS-TR-0078*, 2010. Submitted to the Journal of Science of Computer Programming.
8. C. B. Jones. *Systematic Software Development using VDM (second edition).* Prentice Hall, 1990.
9. C. Morgan. *Programming from Specifications.* Prentice–Hall, 1990.