Visualising program transformations in a stepwise manner

Marta Pląska[1], Marina Waldén[1] and Colin Snook[2]

[1] Åbo Akademi University/TUCS, Joukahaisenkatu 3-5A, 20520 Turku, Finland
[2] University of Southampton, Southampton, SO17 1BJ, UK

1. Introduction

For designing and developing complex, correct and reliable systems formal methods are the most beneficial approach. However, a formal methodology could be difficult for industry practitioners due to its mathematical notation. Hence it needs to be supported by more approachable platform, which would give guidance both for industry and research world representatives. The Unified Modelling Language (UML) with its semi-formal notation gives the intuitive image of the system and is commonly used within the industry. In the stepwise development of our system we combine UML with the formal methods approach and refinement patterns.

For a formal top-down approach we use the Event B formalism [11] and associated proof tool to develop the system and prove its correctness. Event-B is based on Action Systems [5] as well as the B Method [1]. It is related to B Action Systems [15] for developing distributed systems in B. With the Event-B formalism we have tool support for proving the correctness of the development. In order to translate UML models into Event B, the UML-B tool [13] is used. UML-B is a specialisation of UML that defines a formal modelling notation combining UML and B.

The first phase of the design approach is to state the goals that the system needs to satisfy. Afterwards the functional requirements of the system are specified using natural language and illustrated by various UML diagrams, such as statechart diagrams that depict the behaviour of the system. The system is built up gradually in small steps using superposition refinement [4, 10]. We strongly rely on patterns in the refinement process, since these are the cornerstones for creating *reusable* and *robust* software [3, 8]. UML diagrams and corresponding Event B code are developed for each step simultaneously. To get a better overview of the design process, we benefit from the *progress diagrams* [12] that illustrate only the refinement-affected parts of the system. These diagrams are based on well-known and widely-used UML statechart diagrams. In our previous research we introduced progress diagrams, which were illustrated by a case study. Here we focus on exploring refinement patterns in view of progress diagrams, as their combination supports constructing large software systems in an incremental and layered fashion. Moreover, this combination facilitates to master the complexity of the project and to reason about the properties of the system.

2. UML, Event-B and refinement patterns.

We use the Unified Modelling Language™ (UML) [6], as a way of modelling not only the architecture of a system, but also its behaviour and data structure. UML provides a graphical interface and documentation for every stage of the (formal) development process. We focus on the statechart diagram, as it shows the possible states of the object and the transitions between them.

In order to be able to reason formally about the abstract specification, we translate it to the formal language Event-B [11]. An Event-B specification consists of a model and its context that depict the dynamic and the static part of the specification, respectively. They are both identified by unique names. The context contains the sets and constants of the model with their properties and is accessed by the model through the SEES relationship [1]. The dynamic model, on the other hand, defines the state variables, as well as the operations on these. In order to be able to ensure the correctness of the system, the abstract model should be consistent and feasible [11]. Each transition of a statechart diagram is translated to an event in Event-B. The feasibility and the consistency of the specification are then proved using the Event-B prover tool [2].

It is convenient not to handle all the implementation issues at the same time, but to introduce details of the system to the specification in a stepwise manner. Stepwise refinement of a specification is supported by the Event-B formalism. In the refinement process an abstract specification *A* is transformed into a

more concrete and deterministic system *C* that preserves the functionality of *A*. We use the superposition refinement technique [4, 10, 15], where we add new functionality, i.e., new variables and substitutions on these, to a specification in a way that preserves the old behaviour.

In order to guide the refinement process and make it more controllable, *refinement patterns* [15] are used. In this paper we are focusing on data refinement and event refinement. The example of the former is shown in the statechart diagram in Fig 1b (splitting states into substates), while examples of the latter are given in Fig 1c (adding new transitions to use refined states) and Fig 1d (separating existing transitions), when the abstract specification is as in Fig. 1a. Furthermore, we can also consider a pattern for adding the same behaviour to several states (orthogonal regions [14]) as a type of data and event refinement. The pattern types are illustrated in more detail with *progress diagrams* [12] to give an abstraction and graphically-descriptive view documenting the applied patterns in each step.

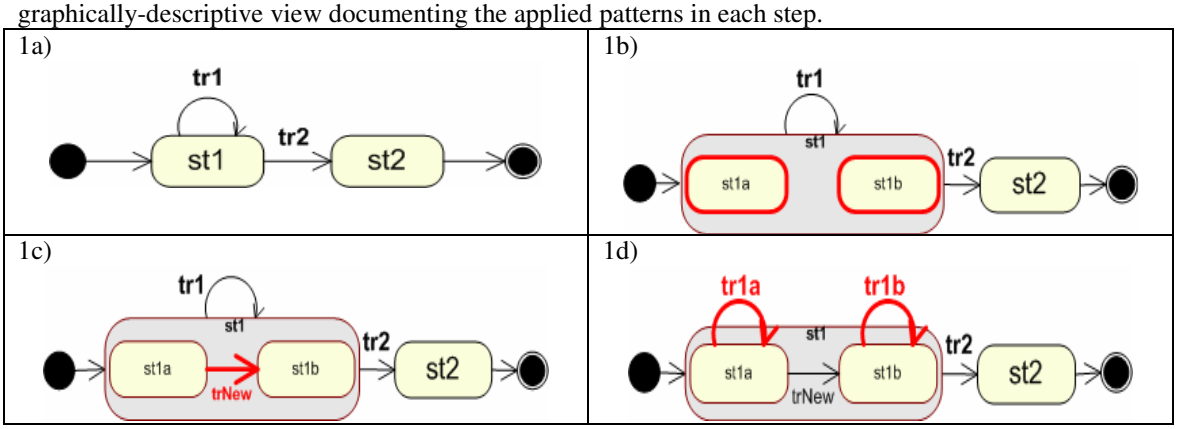


Fig. 1. Refinement patterns

3. Modelling refinement steps – progress diagrams

In most cases it is difficult to get an overview of the refinement process, since the size of the system grows during the development. Although refinement patterns help the designers and developers to refine the system, we combine them with the idea of progress diagrams [12]. The progress diagram is a table that is divided into a description part and a diagram part. The tabular part briefly describes the features and design patterns in the system of the current development step. It also depicts how states and transitions (initiated, refined or anticipated) are refined, as well as new variables that are added with respect to these features. The diagram part gives a supplementary view of the current refinement step and is in fact a fragment of the statechart diagram. Hence, with the progress diagrams we give a more detailed documentation of the refinement patterns and visualise the refinement step, showing the most important features and changes. Our approach results in a clear and legible graphical view of the system.

In order to illustrate the idea of *progress diagrams* in combination with *refinement patterns*, we use the abstract system (shown in Fig.1a) consisting of two states (*st1* and *st2*) and two transitions (*tr1* and *tr2*). We refine it in one step to the concrete system shown in Fig.1c, where the state (*st1*) is partitioned into substates (*st1a* and *st1b*) and the anticipating transition *trNew* is added between the new substates. The progress diagram of this sample *refinement step* is depicted in Fig.2. Here we also assume that a new variable *yy* is added to the refined system, as illustrated in the rightmost column of the progress diagram.

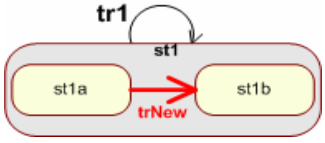
Description	States	Ref. States	Transitions	Ref. Transitions	New Var.
1st refinement step: • creating hierarchical substates (in state *st1*) – data ref. • adding new transition concerning the substates (*trNew*) – event ref.	*st1*	*st1a* *st1b*	-	*trNew*	yy
					

Fig. 2. Example of a progress diagram

4. Conclusion

This paper presents a compact approach to documentation of the stepwise system development focusing on the design patterns. Formal methods and verification techniques are used to ensure that a development is correct. Our approach uses the B Method as a formal framework and allows us to address modelling at different levels of abstraction. The use of progress diagrams during the incremental construction of large software systems provides legible and accessible documentation, whereas the refinement patterns facilitate the (complex) development process. We benefit from the progress diagrams, as we concentrate only on the refined part of the system. The combination of descriptive and visual approaches enables us to focus on the details we are most interested in. Furthermore, it helps to better understand the refinement steps and patterns used in the development, giving a clear overview of the development.

Design patterns in UML and B have been studied previously. Chan et al. [7] work on identifying patterns at the specification level, while we are interested in refinement patterns. The refinement approach on design patterns was presented by Ilič et al. [9]. They focused on using design patterns for integrating requirements into the system models via model transformation. This was done with strong support of the Model Driven Architecture methodology, which we do not consider in this paper. Instead we provide an overview of the development from the patterns.

Our approach is not only helpful for the developers, but also for those that later will try to reuse the exploited features of the system. A clear and compact form of progress diagrams in combination with refinement patterns is appropriate both for industry developers and researchers. Since progress diagrams do not involve any mathematical notation, they are useful for presenting the development steps to non-formal methods colleagues.

As future work tool support will be designed to generate a full refinement chain automatically from the initial specification and a series of progress diagrams in a stepwise manner. Although progress diagrams already appear to be a viable graphical view of the system development, further experimentation on complex case studies is envisaged leading to possible enhancements of the progress diagrams.

References

[1] J.R. Abrial. *The B-Book: Assigning Programs to Meanings*. Cambridge University Press, 1996.

[2] J.R. Abrial, S. Hallerstede, F. Metha, C. Metayer and L. Voisin. Specification of Basic Tools and Platform, RODIN Deliverable 3.3 (D10), http://rodin.cs.ncl.ac.uk/deliverables/rodinD10.pdf, 2005

[3] J. Arlow and I. Neustadt. *Enterprise Patterns and MDA: Building Better Software with Archetype Patterns and UML.* Addison-Wesley, 2004.

[4] R.J.R. Back and R. Kurki-Suonio. Decentralization of process nets with centralized control. In: *Proc. of the 2nd ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pp. 131-142, 1983.

[5] R.J.R. Back and K. Sere. From modular systems to action systems. *Software - Concepts and Tools,* 13, pp. 26-39, 1996.

[6] G. Booch, I. Jacobson, and J. Rumbaugh. *The Unified Modeling Language - a Reference Manual*. Addison-Wesley, 1998.

[7] E. Chan, K. Robinson and B. Welch. Patterns for B: Bridging Formal and Informal Development. In *Proc. of 7th International Conference of B Users (B2007): Formal Specification and Development in B*, LNCS 4355, pp. 125-139, 2007. Springer.

[8] E. Gamma, R. Helm, R. Johnson and J. Vlissides. *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series, 1995.

[9] D. Ilič and E. Troubitsyna. A Formal Model Driven Approach to Requirements Engineering. TUCS Technical Report No 667, Åbo Akademi University, Finland, February 2005.

[10] S.M. Katz. A superimposition control construct for distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(2):337-356, April 1993.

[11] C. Metayer, J.R. Abrial and L. Voisin. *Event-B Language*, RODIN Deliverable 3.2 (D7), http://rodin.cs.ncl.ac.uk/deliverables/D7.pdf (May 2005)

[12] M. Pląska, M. Waldén, C. Snook. Documenting the Progress of the System Development, In *Proc. of Workshop on Methods, Models and Tools for Fault Tolerance,* Oxford, UK, July 2007.

[13] C. Snook and M. Butler. U2B - a tool for translating UML-B models into B. In *UML-B Specification for Proven Embedded Systems Design*, chapter 5. Springer, 2004.

[14] C. Snook and M. Waldén. Refinement of Statemachines using Event B semantics. In *Proc. of 7th International Conference of B Users (B2007): Formal Specification and Development in B*, Besançon, France, LNCS 4355, January 2007, pp. 171-185. Springer.

[15] M. Waldén and K. Sere. Reasoning About Action Systems Using the B-Method. *Formal Methods in Systems Design 13(5-35),* 1998. Kluwer Academic Publishers.