Parallel development of Event-B systems with agile methods*

Masoumeh Parsa[1], Colin Snook[2], Marta Olszewska[1], and Marina Walden[1]

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

Introduction. Formal methods has been used for some decades in safety critical systems such as software for metro, aircrafts and cars. Therefore, formal development is needed in industry, nowadays even to a greater extent, especially in combination with tailored software engineering practices. Due to industrial needs, software engineering practices of today embrace agile principles of flexibility and adaptability [1]. The development process should preferably be iterative and performed by a team. At each iteration step the team should have developed working software. Moreover, the requirements of the system might change over time, thus the development process should be responsive to these changes. Communication within the team and to the outside world is vital for the successful progress of the development [9].

In this paper, we are interested in an agile approach to developing formal models. The stepwise nature of the formal development process matches the agile principles very well. Moreover, we view validation via visualisation and animation as essential to an agile approach, as it allows the team members to communicate their different aspects of the system effectively. However, the development in teams also requires parallel development of models which is fundamental in the agile approach. In parallel development the model is usually decomposed into subparts [2, 7]. However, this demands a substantial initial development as a single chain and still has the disadvantage of not supporting multiple abstractions. We are investigating an alternative approach which starts modelling from several different abstractions independently and subsequently merges the developments to approach a single refined model of the complete system. Hoang et al. [8] introduce composition of different abstractions and this work inspires our contribution which is to incorporate multi-abstraction merging into a useable agile process. Yeganefard et al. [14] also proposed a method for decomposing the requirements into sub-problems; their pattern for defining the sub-problems is, however, different from ours.

We are interested in using Event-B [3] as our formalism, since Event-B is a refinement-based formal modeling language that is also used for industrial systems and has good tool support. The refinement process [5] involves making appropriate abstractions in order to be clear and hence confident that the desired properties are modeled. Currently, Event-B only supports a single linear chain of refinements. This means that only one aspect of the problem can be abstracted. Other aspects have to be incorporated at later refinements when the model is more complicated. Hence, it is a barrier to parallel development by team members needed in the agile process. Tool support is provided by Rodin [4] that is an Eclipse-based framework for developing Event-B specifications and discharging the proof obligations. Moreover, UML-B [12] was developed as a plugin for Rodin to visualize the system requirements by the help of class and state diagrams.

We illustrate our proposed agile modeling method using the Landing Gear case study of Boniol and Weils [6] which was used as a common case study by several authors in the ABZ2014 conference. In the paper by Su and Abrial [13] several modeling strategies for the case study were explored, starting from a functional approach, they moved towards a different generic approach to facilitate more automatic proofs. We observe however, that the latter approach, being more of regular design, is less readable and consequently more difficult to validate which to some extent seems to defeat the purpose of abstract modeling. Here we aim at creating a model that is both easy to validate and verify automatically.

In the case study proposed in this paper, we concentrated on dividing the system into individual subsystems based on aspects, so that we can simplify the proofs through multiple abstractions while maintaining a high degree of readability to facilitate model validation needed for agile team based development. Here we show the early stages

*Work done within the ADVICeS project, Funded by the Academy of Finland, Proj. No.: 266373

of the approach by modeling the sequential aspects of the gear extension and retraction in one abstraction and the analogical switch in another abstraction.

Modeling. In order to adapt the agile principles and allow parallel development during modeling, we focus on different abstractions of the model (system) and develop them in such a way that can keep the model simple. By identifying separate abstractions we emphasize the simplicity so that we can better explore the requirements and make a useful abstract model. Useful abstractions are often difficult to come up with [11]. The requirement specification can help in this identification process. We can note that these abstractions of the system are not necessarily system components.

Each abstraction is modeled and refined separately as UML-B statemachines that are automatically translated to Event-B. Modeling by UML-B statemachines provides us an animation of the system behavior that can be used for validation. This is a useful tool for discussing the model with customers at the same time supporting the agile principles. The correctness of the model is then proved in Event-B.

In our case study, two independent abstractions crystalised from the refinement specification; a moving gear and an analogical switch. The moving gear represents the extension and retraction of the landing gear system while the analogical switch is a hydraulic part controlling the system. The moving gear abstraction was data refined in two steps using UML-B statemachine. Each refinement step was automatically verified in Event-B. We observed a common sequence of states for the abstractions that helped us to construct the statemachines. In addition, the statemachines allowed us to animate the abstractions behavior. By focusing on one aspect at a time, our approach improves the identification of requirements. In Figure 1 we illustrate a time invariant to represent the duration time (40 sec) that the Switch should stay in the state CLOSED after each handleChange event. We can note that the invariants traditionally are used mainly for proving safety properties of the system. Here they also help to explore the details of the requirements via the abstractions.

When modeling the system the requirements revealed that the analogical switch should be closed to enable the moving gear. Hence, during the refinement process we came to a point where the abstractions would merge naturally.

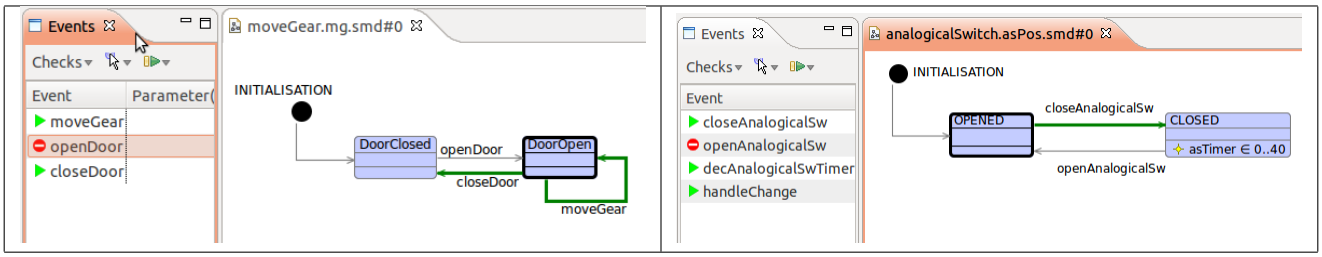

Figure 1: (a) Moving gear and (b) Analogical switch as UML-B statemachines

Merging. In order to get a complete model of the system, the separate abstractions should be merged to a single concrete model. This could take place in any stage of the refinement, which adds flexibility (and agility) to the model development. In order to merge two abstractions in our approach, we make a new refinement for one of the abstraction and then merge in the most concrete model from the other abstraction. This involves copying in elements from the other model taking into consideration common events and/or data that are modeling the same properties (e.g. the event handleChange in abstraction analogicalSwitch and event moveHandle in moveGear were merged to a single event). The relations corresponding to these events in the merged model should also be updated.

When the model statically checks ok, we consider whether there is additional modeling to be added because of the merge. This is due to the fact that one aspect may have an impact on the other one in the merge. It could for example be strengthening the guards of events to take into account the new features of the other aspect (the analogical switch being closed is a condition for any gear/door event). In this way, when we prove that the merged model is a correct refinement of its separate parts, we do not merely prove superposition refinement. Hence, the merge can be seen as a refinement of the separate aspects that were merged. Figure 2 shows how the refinement chains of two different aspects merge into one refined model.

In order for the merge process to be feasible, we would need tool support. We will investigate using, and possibly extending, the composition plug-in [10] for this purpose.

In our case study, the analogicalSwitch model was copied into the moveGear model to create a merge. We added guards (asPos = CLOSED) on all transitions in the merged statemachine. This adds the new requirement that nothing can happen without the analogical switch being closed. A similar statemachine was created to refine the analogicalSwitch model. Hence, we have two functional aspects developed from separate abstractions into a single refinement.

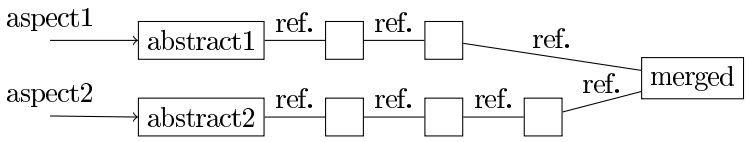

Figure 2: Merging the refined models of two abstractions

A more significant example of the merging would be if we find that one aspect A contains an abstraction of some data in the other aspect B. In this case we need to perform data refinement of the abstract data on the events that came from A and prove the refinement in A's project.

Conclusions. Formalizing the critical part of industrial systems requires adopting agile principles to the formal development method. Simplicity is emphasized in the model via the abstractions to better be able to react to unexpected changes in the development. To achieve this goal, we focus on identifying abstractions of the system by partitioning and exploring the requirements. The abstractions can then be modeled and refined in parallel supporting the idea of team work. The verification and validation of the model can be kept simpler in this way. In order to get a complete model of the system we merge the different abstractions into a single model as a refinement step. The merged model is proved to be a refinement of both the abstractions. We illustrate our approach by using parts of the landing gear system as a case study. In our future work we would like to define guidelines to assist in identifying the abstractions of the system and also study further the merging of the abstractions.

We model the abstractions as UML-B statemachines that are automatically translated to Event-B. In this way we have tool support for the development. Via the UML-B statemachines the different steps of the development can be animated and validated. The parallel development of simpler parts of the system together with the animation facilities provided by the tool support takes the agile principles into account and makes the formal development process more adaptive and flexible. We anticipate that the formal development of critical systems can benefit from incorporating agile principles.

References

[1] Guide to agile practices. `http://guide.agilealliance.org`.

[2] Jean-Raymond Abrial. Event model decomposition. Technical report, ETH Zurich, 2009.

[3] Jean-Raymond Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, New York, NY, USA, 1st edition, 2010.

[4] Jean-Raymond Abrial, Michael Butler, Stefan Hallerstede, Thai Son Hoang, Farhad Mehta, and Laurent Voisin. Rodin: an open toolset for modelling and reasoning in Event-B. *Int. J. Softw. Tools Technol. Transf.*, 12(6):447–466, November 2010.

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

[6] Frederic Boniol and Virginie Wiels. The landing gear system case study. In *ABZ 2014: The Landing Gear Case Study*, pages 1–18. Springer, 2014.

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

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

[9] Marta Olszewska and Marina Walden. FormAgi - A Concept for More Flexible Formal Developments. Technical report, Department of Computer Science, Åbo Akademi. To appear.

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

[11] Colin Snook. Exploring the barriers to formal specification, PhD Thesis, 2001.

[12] Colin Snook and Michael Butler. UML-B: Formal modeling and design aided by uml. *ACM Trans. Softw. Eng. Methodol*, pages 92–122, 2006.

[13] Wen Su and Jean-Raymond Abrial. Aircraft Landing Gear System: Approaches with Event-B to the Modeling of an Industrial System. In *ABZ 2014: The Landing Gear Case Study*, ABZ '14, pages 19–35. Springer International Publishing, 2014.

[14] Sanaz Yeganefard and Michael Butler. Problem decomposition and sub-model reconciliation of control systems in Event-B. In *IEEE 14th International Conference on Information Reuse & Integration, IRI 2013, San Francisco, CA, USA, August 14-16, 2013*, pages 528–535. IEEE, 2013.