Decomposition Tool: Development and Usage

Renato Silva[1], Carine Pascal[2], T.S. Hoang[3], and Michael Butler[1]

[1] University of Southampton
[2] Systerel
[3] ETH Zurich
{ras07r,mjb}@ecs.soton.ac.uk
carine.pascal@systerel.fr
htson@inf.ethz.ch

Abstract. Decomposition of a formal development (in particular for Event-B models) allows the simplification and separation of logics in the initial abstract model. Specific parts of an abstract model can be partitioned and independently refined through the introduction of individual details without the "interference noise" of other non relevant sub-components.

For instance, while developing a metro system and respective safety measures, one can start the abstract model by introducing the sections of the tracks, the trains and respective carriages and a communication module that interchanges messages between the train and sections of the track. This is an abstract, coarse grain view of the entire model which is useful in a top-down development. But in order to introduce more specific details about each sub-component, we would like to isolate them while maintaining the original properties. The abstract model can be decomposed into sub-components like tracks, trains and communication and refine individually each one of them. Moreover, an advantage of following this modelling methodology is that the sub-components can be developed in parallel which allows team development. Consequently, trains can be further refined with the introduction of doors in each carriage, the definition of the conditions for opening/closing the doors and definition of emergency procedures (what happens when an emergency occurs while train is in a platform or in the middle of the track or the train is moving). In parallel, the communication model can be further specified with the introduction of a protocol for sending/receiving messages and other details.

The decomposition tool was developed to answer the previous challenge, allowing the separation of a development (abstract machine or one of the concrete machines in a refinement chain) into sub-components. The separation can be done by following either a shared variable or shared event approach. In the first approach, the user distributes the original events into sub-components. Consequently these sub-components share some original (shared) variables. Moreover (additional) external events are created to model the effect of the other sub-component on the shared variables [1]. In the second approach, variables are distributed among the sub-components according to the user's choice and as a result, sub-components share some events which are partitions of the original ones.

The first approach is suitable for development of parallel programs [2] while the second is more suitable for the development of distributed systems [3]. Both approaches are monotonic [4]: sub-components can be further refined (and eventually decomposed again) while maintaining the properties of the original non-decomposed component. An advantage of decomposition is the distribution of the proof obligations among the sub-components. Although it should be possible to continue the sub-component refinements in the original development, non-relevant sub-components interfere in a negative way when discharging the proof obligations, in particular for large developments. For a large model, the associated proof obligations for proving its correctness are usually polluted by irrelevant details, which makes the reasoning about the model consistency more difficult. Decomposition helps alleviate this issue by separating the relevant from non-relevant details into different sub-components. Here we present an overview of how the tool was developed as a plug-in for the rodin platform [5] and the changes and additions that were required to make this plug-in more robust and user friendly. We also illustrate the usage of the decomposition tool in a case study through a demo.

A challenge for the future is how to integrate the decomposition tool with other plug-ins. At the moment, only the standard Event-B elements are decomposed but ideally additional elements introduced by external plug-ins (like records) could/should also be decomposed. We need to find a solution that allows the external plug-ins to define how to decompose additional elements in the Event-B models. It seems that adding a decomposition extension-point to be implemented by plug-ins that require decomposition is a possible solution although the content of this extension-point is still vague at the moment. Further study is required and we shall explore it in the future.

Key words: decomposition, Event-B, specification, formal methods

References

1. Abrial, J.R.: Event Model Decomposition. Technical report, ETH Zurich (2009 (Unpublished))
2. Hoang, T., Abrial, J.R.: Event-B Decomposition for Parallel Programs. Abstract State Machines, Alloy, B and Z (2010) 319–333
3. Butler, M.: An Approach to the Design of Distributed Systems with B AMN. In: Proc. 10th Int. Conf. of Z Users: The Z Formal Specification Notation (ZUM), LNCS 1212. (1997) 221–241
4. Silva, R., Pascal, C., Hoang, T.S., Butler, M.: Decomposition Tool for Event-B. In: Workshop on Tool Building in Formal Methods - ABZ Conference, Orford, Quebec, Canada (February 2009)
5. Abrial, J.R., Butler, M., Hallerstede, S., Hoang, T.S., Mehta, F., Voisin, L.: Rodin: An Open Toolset for Modelling and Reasoning in Event-B. International Journal on Software Tools for Technology Transfer (STTT) (April 2010)