Decomposition Tool for Event-B*

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. Two methods have been identified for Event-B model decomposition: shared variable and shared event. The purpose of this paper is to introduce the two approaches and the respective tool support in the Rodin platform. Besides alleviating the complexity for large systems and respective proofs, decomposition allows team development in parallel over the same model which is very attractive in the industrial environment.

Key words: Formal Methods, Event-B, Decomposition, Shared Event, Shared Variable, Team Development, Rodin

1 Introduction

The "top-down" style of development used in Event-B [1] allows the introduction of new events and data-refinement of variables during refinement steps. A consequence of this development style is an increasing complexity of the refinement process when dealing with many events and state variables. The main purpose of the model decomposition is precisely to address such difficulty by cutting a large model into smaller components. Two methods have been identified for the Event-B decomposition: shared variable [2, 3] and shared event [4, 5]. We propose a plug-in developed in the Rodin platform [6] that supports these two decomposition methods for Event-B. Because decomposition is monotonic [4], the generated sub-components can be further refined independently. Therefore we can introduce team developments: several developers share parts of the same model and work independently in parallel. Moreover the decomposition also partition the proof obligations which are expected to be easier to be discharged in the sub-components.

This document is structured as follows: Sect. 2 defines the two styles of Event-B decomposition. Section 3 introduces in more details the tool support. Finally in Sect. 4 we draw conclusions about this work.

* Part of this research was carried out within the European Commission ICT project 214158 DEPLOY (http://www.deploy-project.eu)

2 Decomposition Styles

Consider Fig. 1 where machine M has events *e1* to *e4* and variables *v1* to *v3*. The solid lines connect variables used by the events. In Fig. 1(a), machine M is decomposed using the *shared variable approach* where events are partitioned into sub-components: events *e1* and *e2* are allocated to machine *M1* and *e3* and *e4* are allocated to machine *M2*. Consequently, variable *v1* belongs to *M1* and *v3* belongs to *M2* (*private variables*). Variable *v2* is shared between *e2* and *e3* so it is shared by both sub-components (*shared variable*). Besides the initial event allocation we introduce additional *external events* simulating how shared variables are handled in the other sub-component (*e3_ext* is added to $M1$ and *e2_ext* to $M2$). Sub-components can be refined independently but the shared variable must be present and cannot be data-refined.

Figure 1(b) depicts machine M again decomposed but using the *shared event approach* where variables are partitioned to sub-components: *v1* is placed in *M1* and *v2*, *v3* are placed in *M2*. Events using variables allocated to different sub-components (*e2* shares *v1* and *v2*) must be split. The part corresponding to each variable (*e2_1* and *e2_2*) is used to create partial versions of the non-decomposed event. The sub-components can be refined independently without constraints.

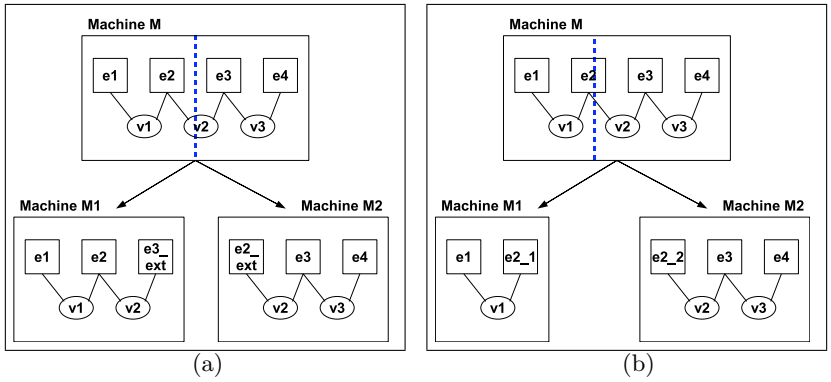


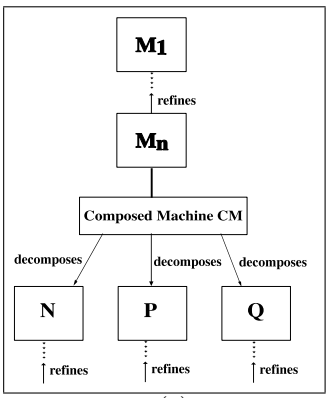
Fig. 1. Shared Variable decomposition on the left and shared event decomposition on the right

Shared event approach is suitable for message-passing distributed systems while shared variable approach is suitable for design of parallel algorithms [7].

3 Decomposition Tool

The decomposition tool is implemented as a plug-in in the Rodin platform. The decomposition style used depends on the inputted system and on the end-user's

preference. The decomposition originates sub-components according to the configuration (events/variables partition). That configuration is stored persistently in a composed machine [8] for future reuse or editing as seen in Fig. 2. "Replaying" the decomposition might require additional storing mechanisms. We intend to address this issue in the future.



(a)

COMPOSED MACHINE CM
TYPE SHARED EVENT
REFINES M_n
INCLUDES
N
P
Q
EVENTS
evt_1 **REFINES** M_n.evt_1
Combines Events N.evt_1 ‖ P.evt_1
...
evt_n **REFINES** M_n.evt_n
Combines Events P.evt_n ‖ Q.evt_n
END

(b)

Fig. 2. Decomposition tool diagram for a machine M_n and composed machine CM using the shared event approach

The input for the decomposition is a machine of a given Rodin project selected by the end-user. After the selection of the style and decomposition configuration, the tool generates the sub-components automatically. Summarising these are the steps to be followed in order to decompose (we decompose M_n in Fig. 2(a)):

1. End-user selects a machine M_n to decompose.
2. End-user defines sub-components to be generated: N, P, Q ....
3. End-user selects the decomposition style to use:
   Shared Variable: end-user selects the events to be allocated to sub-components. The tool automatically decomposes the rest of the model according to the event partition (shared/private variables, external events).
   Shared Event: the end-user selects the variables to be allocated for each sub-component. The rest is done automatically.
4. The end-user can opt to decompose the seen contexts into the sub-components similarly to the machine decomposition.
5. Sub-components are fulfilled according to the decomposition configuration.
6. The decomposition configuration is stored as a composed machine.
7. Sub-components N, P, Q ... can be further refined.

The configuration is performed through the Rodin's Graphical User Interface as it seems more suitable for the end-user.

3.1 Limitations

For the shared variable decomposition the partition of events is always possible since the possible restriction (variables) can be shared between sub-components. On the other hand, that decomposition might be less significant despite being possible: a further refinement may be more complex and not benefit the development. The decomposition should have a final goal: a misleading decomposition may harm a system instead of helping. For the shared event decomposition, the partition of variables is not always possible for all developments. Predicates using variables allocated to different sub-components need user's interaction in order to simplify the separation since such operation cannot be done automatically. The tool automatically flags situations requiring user's intervention.

4 Conclusion

This paper presents the decomposition of Event-B models and tool support in the Rodin platform. Decomposition can advantageously be used to decrease the complexity and increase the modularity of large systems, especially after several refinements. Main benefits are the distribution of proof obligations over the sub-components which are expected to be easier to be discharged and the further refinement of independent sub-components in parallel introducing team development of a model which is attractive for the industry. Shared variable and shared event decomposition are supported in the same tool: the former seems suitable when designing concurrent programs while the latter seems particularly suitable for message-passing distributed programs. However the end-user chooses a decomposition style depending on specific systems and on its modelling preferences. The decomposition configuration is stored persistently for replaying/editing although further study is still required for this matter. A visualisation view for decomposition seems intuitive and we intend to explore it using GMF [9].

References

1. Métayer, C., Abrial, J.R., Voisin, L.: Event-B Language. Technical report, Deliverable 3.2, EU Project IST-511599 - RODIN (May 2005)
2. Abrial, J.R., Hallerstede, S.: Refinement, Decomposition, and Instantiation of Discrete Models: Application to Event-B. Fundam. Inf. **77**(1-2) (2007) 1–28
3. Abrial, J.R.: Event Model Decomposition. Technical report, ETH Zurich (2009 (Unpublished))
4. Butler, M.: Synchronisation-based Decomposition for Event-B. In: RODIN Deliverable D19 Intermediate report on methodology. (2006)
5. Butler, M.: Decomposition Structures for Event-B. Integrated Formal Methods iFM2009 (February 2009)
6. Rodin: RODIN project Homepage. `http://rodin.cs.ncl.ac.uk` (September 2008)
7. 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
8. Silva, R., Butler, M.: Parallel Composition Using Event-B. `http://wiki.event-b.org/index.php/Parallel_Composition_using_Event-B` (July 2009)
9. GMF: Graphical Modeling Framework. `http://www.eclipse.org/modeling/gmf/` (September 2008)