Event-B Model Decomposition

Carine Pascal[1] and Renato Silva[2]

[1] Systerel
[2] University of Southampton

Abstract. Two methods have been identified in the DEPLOY[3] project for Event-B model decomposition: the shared variable decomposition (called A-style decomposition), and the shared event decomposition (or B-style decomposition). Both allow the decomposition of a (concrete) model into several independent sub-models which may then be refined separately. The purpose of this paper is to introduce the Event-B model decomposition, from theory (A-style vs. B-style, differences and similarities) to practice (decomposition plug-in of the Rodin [1] platform).

Key words: formal methods, Event-B, decomposition, A-style, B-style, team development, Rodin

1 Introduction

The "top-down" style of development used in Event-B[2] 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 having to deal with many events and many state variables. The main purpose of the model decomposition is precisely to address such difficulty by cutting a large model into smaller components. Decomposition cuts a model MC into sub-models $MC_1, ..., MC_n$, which can be refined independently and more comfortably than the whole. This solution gives an interesting perspective for the team development of a model: the possibility for several developers to share parts of a model and to work independently and possibly in parallel, on them.

1.1 Definition and constraints

A model contains contexts, machines or both. The notion of model decomposition covers machine and context decomposition.

[3] DEPLOY - Industrial deployment of system engineering methods providing high dependability and productivity - FP7 Integrated Project supported by the European Commission (Grant 214158), under Strategic Objective IST-2007.1.2.
See `http://www.deploy-project.eu`.

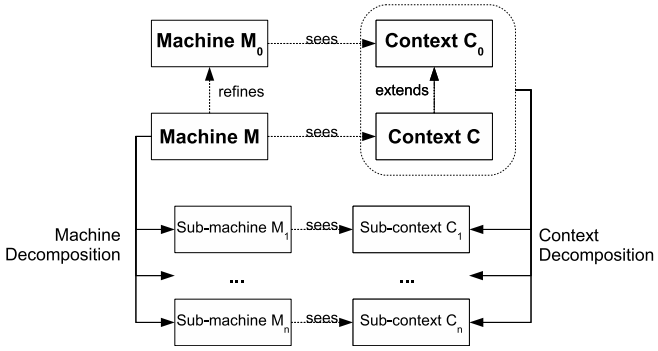

Fig. 1. Event model decomposition

The entry point for the decomposition of a model is a machine M and its whole hierarchy of seen contexts as illustrated in Fig. 1. The resulting sub-models are independent of the non-decomposed model and the partition of the models includes the splitting of variables, invariants, events and even context elements like sets, constants and axioms. The main constraint is that if these refined models are recomposed into a model MR, it is guaranteed that MR refines MC as depicted in Fig. 2 (monotonicity). Decomposition does not generate new proof obligations (POs).

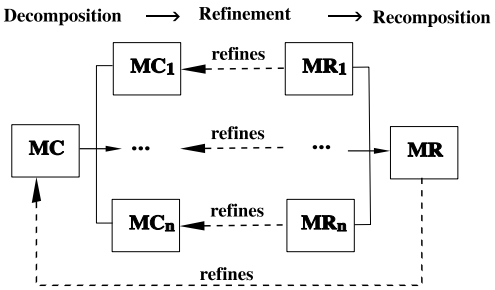

Fig. 2. Model decomposition, refinement, and re-composition

2 Decomposition Styles

This section concretely explains how to decompose a given machine M in sub-machines $M_1, ..., M_n$, depending on the chosen decomposition style: *shared variable* or *shared event*. The shared event approach seems particularly suitable for message-passing distributed programs, while the shared variable approach seems more suitable when designing concurrent programs [3].

2.1 Shared Variable (A-style) Decomposition

Figure 3 illustrates the shared variable decomposition proposed by Abrial [4], Metayer et. al [5] and Abrial and Hallerstede [6]. Machine M has events *e1* to *e4* and variables *v1* to *v3*. The solid lines connect variables used by the events. Events of the non-decomposed component (machine M) are partition among the sub-components (Machine *M1*:*e1* and *e2*; Machine *M2*:*e3* and *e4*). After the event partition it is necessary to split the variables. The sub-components can be refined independently respecting some restrictions and the re-composition of the sub-components should always be a refinement of the non-decomposed component.

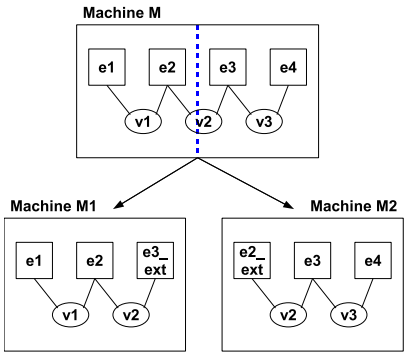

Fig. 3. Shared Variable Decomposition of machine M into machines *M1* and *M2* with shared variable *v2*

The following sequence must be observed to decompose M [7]:

1. The events of M are first partitioned over $M_1, ..., M_n$.
2. The variables of M are distributed according to the event partition. Variables accessed (i.e. read or written) by events of distinct sub-machines, are called *shared variables* like *v2* (in opposition to *private variables* like *v1* and *v3*).
3. The invariants of M are distributed according to the variable distribution. An invariant based on a predicate $P(v_1, ..., v_m)$ is copied to M_i if and only if M_i contains all the $v_1, ..., v_m$ variables.
4. *External* events are built in $M_1, ..., M_n$. If $e2$ is an event of M_i that modifies a shared variable $v2$, then an external event is built from $e2$ in each sub-machine M_j where $v2$ is accessed. An external event of M_i must always be present and be strictly identical in any refinement of M_i.

2.2 Shared Event (B-style) Decomposition

The B-style decomposition is achieved with the partition of synchronized events among the sub-models. Figure 4 depicts the shared event decomposition proposed by Butler [8]. The state variables are refined so they may be partitioned

amongst the subsystems, introducing internal events representing interaction between subsystems. More details about this approach can be found in [9, 10].

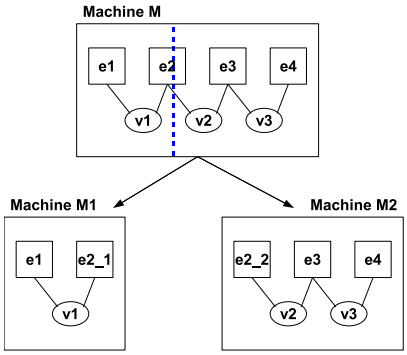

Fig. 4. Shared Event Decomposition of Machine M in Machines *M1* and *M2* with shared event *e2*

The several steps for the shared event decomposition are given below:

1. The variables of M are first partitioned among $M_1, ..., M_n$.
2. The events of M are distributed among $M_1, ..., M_n$, according to the variable partition. There is no notion of shared variables. 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
3. The invariants of M are distributed similarly to A-style.

3 Tool Specification

Both decomposition styles are implemented in the same Rodin platform plug-in. The decomposition input is a machine of a given Rodin project selected by the end-user. The next step of the decomposition (i.e. the event partition for A-style or the variable partition for B-style) requires the intervention of the end-user. Afterwards the sub-components resulting from the decomposition are defined. All other steps are automatically performed by the plug-in. Summarising these are the steps to be followed in order to decompose (we decompose M in Fig. 5(a)):

1. End-user selects a machine M to decompose.
2. End-user defines sub-components to be generated: M_1, M_2, ... M_n.
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).

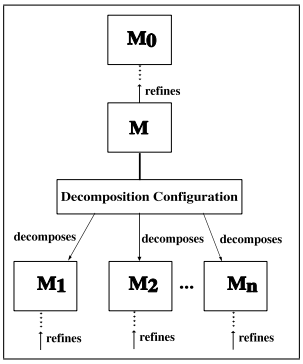

Fig. 5. Decomposition tool diagram for machine M

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. The contexts seen by $M_1, ..., M_n$, are built from the hierarchy of contexts associated to M and are built following the sequence:
   (a) A constant in this hierarchy is copied to a sub-context C_i associated to M_i if and only if it appears in a predicate (invariant or guard) or in an assignment (action) of M_i.
   (b) A carrier set in this hierarchy is copied to C_i if and only if it appears in a predicate or in an assignment of M_i, or if it types a constant previously copied to C_i.
   (c) An axiom is copied to M_i if and only if M_i contains the referenced constants and sets.
5. Sub-components are fulfilled according to the decomposition configuration.
6. The decomposition configuration is stored.
7. Sub-components M_1, M_2, ... M_n can be further refined.

The generated sub-components can be created in the same project as the non-decomposed model or created as new projects, according to the user's decision. Moreover, the following requirements have been identified for the decomposition plug-in:

1. The configuration (i.e. input machine, decomposition style selection, identification of sub-components to be generated and respective partition) shall be performed through the Graphical User Interface (GUI) of the Rodin platform. It is indeed more suitable for the end-user to visualise the configuration. In the future the option of using GMF (Graphical Modelling Framework) for the decomposition visualization will be explored.
2. The decomposition configuration shall be stored persistently. Since there is no direct relation between the non-decomposed model and the decomposed

sub-models, the possibility of saving, editing and replaying a model decomposition seems suitable.
3. A `Decompose` action shall be added. It shall be available from the toolbar and from the popup menu of the Event-B explorer (right-clicking on a machine).
4. The created projects and components (machines and contexts) shall be tagged as "automatically generated".

4 Conclusion

This paper presents a preliminary study about the decomposition of Event-B models. The Event-B model decomposition can advantageously be used if the motivation is to decrease the complexity and increase the modularity of large systems, especially after several layers of refinements. The main benefits are the distribution of POs into several sub-models, which is expected to be easier to discharge, and the further refinement of independent sub-models in parallel. Moreover the possibility of team development seems a very attractive option for the industry. This paper propose the reuse of sub-components in Event-B through the shared variable (A-style) or shared event (B-style) decompositon. The shared event approach seems particularly suitable for message-passing distributed programs, while the shared variable approach seems more suitable when designing concurrent programs. However the end-user will choose a decomposition style depending on the specific system and on their modelling preferences.

References

1. Rodin: RODIN project Homepage. `http://rodin.cs.ncl.ac.uk` (September 2008)
2. Abrial, J.R.: Mathematical models for refinement and decomposition. `http://www.event-b.org/abook.html` (2009, To be published)
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. Abrial, J.R.: Event model decomposition. Technical report, ETH Zurich (2009, Private communication)
5. Métayer, C., Abrial, J.R., Voisin, L.: Event-B Language. Technical report, Deliverable 3.2, EU Project IST-511599 - RODIN (May 2005)
6. Abrial, J.R., Hallerstede, S.: Refinement, Decomposition, and Instantiation of Discrete Models: Application to Event-B. Fundam. Inf. **77**(1-2) (2007) 1–28
7. Pascal, C.: Event Model Decomposition. `http://wiki.event-b.org/index.php/Event_Model_Decomposition` (2009)
8. Butler, M.: Synchronisation-based Decomposition for Event-B. In: RODIN Deliverable D19 Intermediate report on methodology. (2006)
9. Butler, M.: Incremental Design of Distributed Systems with Event-B. Marktoberdorf Summer School 2008 Lecture Notes (November 2008)
10. Butler, M.: Decomposition Structures for Event-B. Integrated Formal Methods iFM2009 (February 2009)