Science of Computer Programming 94 (2014) 144–163

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Refinement of decomposed models by interface instantiation

Stefan Hallerstede [a,*], Thai Son Hoang [b]

[a] Aarhus University, Denmark
[b] ETH Zurich, Switzerland

HIGHLIGHTS

- A limited form of refinement called interface instantiation is proposed for the Event-B formalism.
- Interface instantiation equips Event-B with a technique for shared-variable refinement.
- Interface instantiation fits seamlessly with Event-B decomposition and refinement.
- A worked out example is used to demonstrate this.
- A correctness proof is provided.

ARTICLE INFO

Article history:
Received 15 February 2013
Received in revised form 30 April 2014
Accepted 5 May 2014
Available online 20 May 2014

Keywords:
Event-B
Decomposition
Refinement
External variables
Interface instantiation

ABSTRACT

Decomposition is a technique to separate the design of a complex system into smaller sub-models, which improves scalability and team development. In the shared-variable decomposition approach for Event-B, sub-models share external variables and communicate through external events which cannot be easily refined.

Our first contribution hence is a proposal for a new construct called interface that encapsulates the external variables, along with a mechanism for interface instantiation. Using the new construct and mechanism, external variables can be refined consistently. Our second contribution is an approach for verifying the correctness of Event-B extensions using the supporting Rodin tool. We illustrate our approach by proving the correctness of interface instantiation.

1. Introduction

When decomposing a model into sub-models we intend to continue refining the sub-models independently of each other while preserving the properties of the full model. A suitable decomposition method for Event-B has been proposed by Abrial [1]. It partitions events of a model between its sub-models. Variables of the model are split correspondingly into external variables shared by the sub-models and internal variables private to each sub-model. For all external variables of a sub-model, external events that mimic the effect of corresponding (internal) events of other sub-models have to be added. If we want to refine external variables, we have to provide a gluing invariant that is functional, say, $v = h(w)$ where v are the abstract variables and w the concrete variables. Abrial [1] also proposes to rewrite the external events with $v := h(w)$ so that concrete and abstract events are equivalent. Internal variables and internal events are refined as usual in Event-B [2].

We propose a practical method for external event refinement that aids in structuring and understanding complex models. This requires a trade-off between generality and practicality. We believe that it would be difficult to generalise the method

* Corresponding author.
E-mail addresses: stefan.hallerstede@wanadoo.fr (S. Hallerstede), htson@inf.ethz.ch (T.S. Hoang).

http://dx.doi.org/10.1016/j.scico.2014.05.005
0167-6423/

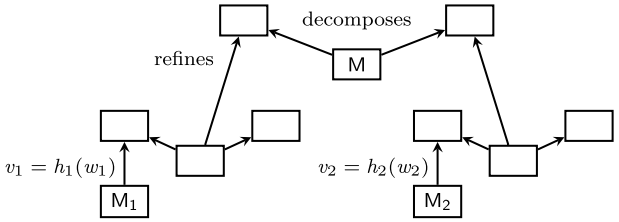

Fig. 1. Maintaining the external invariants of several sub-models "manually".

that we propose without sacrificing its practicality. The theory of the method is not difficult. Our aim is to make using a difficult technique, the refinement of external variables and external events in Event-B, as easy as possible.

We call a collection of external variables with the external invariants an *interface*. Modelling interfaces "manually" by marking the corresponding variables as being external and refining them by specifying functional invariants makes it difficult to decompose and refine a model repeatedly. Fig. 1 illustrates the problem where a model M is decomposed three times and the resulting sub-models are refined. We are interested in the two sub-models M_1 and M_2 at the bottom. How do we find the shared external invariants?

The lists of variables w_1, w_2, v_1 and v_2 are not necessarily disjoint. Let w be the list of variables occurring in w_1 or w_2 and v be the list of variables occurring in v_1 or v_2. We need to find one suitable external invariant $v = h(w)$ to be used in the sub-models M_1 and M_2. What is the shape of h? Furthermore, when refining M_2 we have to think about the necessary changes to M_1. As a consequence of the current situation, interfaces are refined to implementation level before decomposition. This complicates the use of decomposition on higher levels of abstraction. We would prefer a method where the necessary reasoning can be restricted to one place. The functional invariant h should be evident and easily maintainable also in the face of potential changes to the sub-models and the interfaces.

Using our approach of interface instantiation this can be done. Because we are treating instantiation as a special form of refinement, we can combine interface instantiation steps with refinement steps. This gives us some liberty in arranging complex refinements. We also encourage a decomposition style where a separate theory of interface instantiation is maintained. We think that this contributes substantially to obtain models that are easier to understand and to modify. Interface instantiation supports a more incremental approach to decomposition because modifications that concern several components can be confined to only one place: the interface.

We call the very specific form of interface refinement that we use *interface instantiation*. To be useful, it should

(i) be more liberal than [1] while not increasing the proof effort,
(ii) help to structure complex mixtures of decomposition and refinement,
(iii) work seamlessly with Event-B as it is. (It should not depend on translations.)

We argue by means of a case study that we have achieved this. The case study addresses a difficulty of relating Event-B refinement to Problem Frames elaboration [12] discussed in [7]. It has been composed from [7] and [15]. We have down-sized it in order to focus on the problem of the refinement of external variables, that is, the interfaces. We have a tool for decomposition [18] but we do not have implemented a software tool for interface instantiation. Instantiation of carrier sets has been implemented similarly internally in the ProB tool [14], in order to achieve better performance when model checking and constraint checking [7]. The case study as presented in [15] uses Problem Frames to achieve traceability of requirements. We have not used Problem Frames in this article because they are not required to explain interface instantiation. This also permits us to cast the problem entirely in Event-B terminology. However, the proposed method of instantiation could be used with Problem Frames as employed in [7,15]. This work extends prior work presented in [9] by allowing instantiations in a lattice of interfaces.

In the modularisation approach for Event-B presented in [11], the notion of interface has been used to capture software specifications using some interface variables and operations acting on these variables. The intention behind the use of interfaces is to separate specifications from their implementations. Our notion of interface is intended to provide efficient support for refining external variables following Abrial's decomposition method for system models. There was an earlier attempt at external variable refinement that is hinted at in the specification of the proof obligation generator for the Rodin tool [8]. This was considered too complicated and not feasible for large systems that are decomposed and refined repeatedly. We think, that our approach solves the problem. Poppleton [16] discusses external refinement based on Abrial's approach but also does not provide a practicable technique for doing so. The approach of modelling extensible records [6] also permits a form interface instantiation. A difficulty with using this approach is caused by the explicit mathematical model used for record representations and the need to specify always values for all fields of a record. However, extensible records could be used with our approach where it would appear useful. Behavioural interface refinement such as discussed in [17] addresses changing traces sub-models can exhibit, usually adding new events. It does not consider refinement of shared variables.