Towards Component-based Reuse for Event-B

Andrew Edmunds[1], Marina Walden[1], and Colin Snook[2]

[1] Åbo Akademi University, Turku, Finland
aedmunds@abo.fi, mwalden@abo.fi
[2] University of Southampton, UK cfs@ecs.soton.ac.uk

Abstract: An efficient re-use mechanism is a primary goal of many software development strategies; and is also important in the safety-critical domain, where formal development is required. Event-B can be used to develop safety-critical systems, but could be improved by development of a component-based re-use strategy. In this paper we outline a methodology, and the tool support required, for facilitating re-use of Event-B machines. As part of the ADVICeS project [10] we are seeking to improve re-use of Event-B artefacts. The creation of a library of components, and a way to assemble them, would facilitate this. We propose to extend iUML-B class diagrams [9], and extend the composition techniques introduced in [7], to allow specification of Event-B components, interfaces, and composite components. Initial investigation has been undertaken as part of the project ADVICeS, funded by Academy of Finland, grant No. 266373. The approach also addresses the need, in Event-B, for bottom-up scalability. We describe the process of creating library components, their composition, and specification of new properties (of the composed elements). We introduce the notion of Event-B components, component interfaces, and composite components. We describe the additional annotations, and discuss composition invariants.

1) Preliminaries

Event-B is a language and methodology [1, 2], with tool support provided by Rodin [3]. The system and its properties are specified using set-theory and predicate logic. It uses refinement [6] to show that the properties hold as the development proceeds. Refinement is used to add detail to the development. Event-B tools are designed to reduce the amount of interactive proof required during specification, and refinement steps [4]. Proof obligations (P.O.s) in the form of sequents, are automatically generated by the Rodin tool. The automatic prover can discharge many of the P.O.s. The remainder can be tackled with the interactive prover. Complex systems can be simplified using decomposition techniques [8].

The basic Event-B elements are *contexts*, *machines* and, *composed-machines*. Contexts define the static parts of the system using sets, constants and axioms. Machines describe the dynamic parts of a system using variables and events, and use invariant predicates to describe properties that should hold. We specify an event in the following way,

$$e \triangleq \textbf{ ANY } p \textbf{ WHERE } G \textbf{ THEN } A \textbf{ END}$$

where e has parameters p; a guarding predicate G; and actions A. For the state updates (described in the action) to take place, the guard must be true.

The Composition of Decomposed Machines: Previous work [7] describes the composition that arises from the decomposition of a single machine. Multiple, decomposed sub-units, and the composed-machine construct, form a refinement of the abstract machine. We use the shared-event approach for decomposition, where the combined-events clause, of the composed-machine, refines an abstract event e. We write $e_a \parallel e_b$ to combine events e_a and e_b, where subscripts a and b also identify the sub-units (machines). These events are said to *synchronize* (i.e., the events are enabled) when the conjunction of the guards are true. The combined actions are composed in parallel.

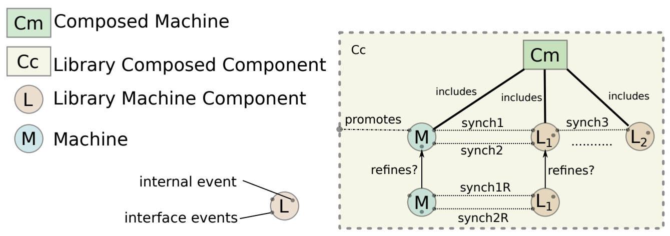
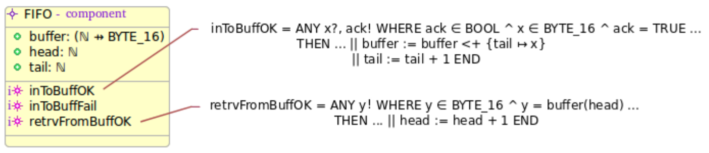


Figure 1: Machines "Included" in a Composition



```
inToBuffOK = ANY x?, ack! WHERE ack ∈ BOOL ^ x ∈ BYTE_16 ^ ack = TRUE ...
              THEN ... || buffer := buffer <+ {tail ↦ x}
                       || tail := tail + 1 END

retrvFromBuffOK = ANY y! WHERE y ∈ BYTE_16 ^ y = buffer(head) ...
                  THEN ... || head := head + 1 END
```

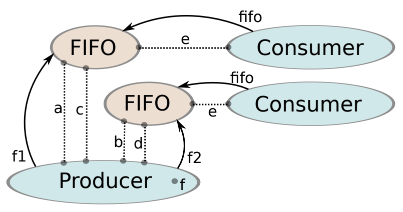
Figure 2: The FIFO Buffer Component

2) Composition with Components
By extending existing techniques, we aim to facilitate creation/use of a library of machines. Figure 1 shows a composed-machine Cm that includes library machine components L_1 and L_2, and machines under construction M. Combined-events are shown using a dashed line between the machines. An interface reveals a set of events that may synchronize with some other machines, annotated with i against the event. See Fig. 2, an extended version of an existing iUML-B class diagram [9]. For parameter passing, the names and types of the *communicating* parameters are revealed, using ? and ! for input and output resp. A composed-machine may also be treated as a library component.
Using Component Instances: The component defined in Fig. 2 may be used to buffer data for producer and consumer models, see Fig. 3, a new diagrammatic representation in Event-B. Here, arrows represent associations, and dashed lines represent combined events.
The Composition Invariant: Each individual machine has its own set of invariants, and the composed-machine has *composition invariants* which specify properties about the composition. These properties cannot be specified in the individual machines. The composed machine needs visibility of *all* of the variables contained within the composition, and their included sets and constants. To ensure the composition invariant is satisfiable, we should add a guard $\boldsymbol{G_{CI}}$ to the composed event, but currently there is no feature in the tool to do this. The guards that preserve the new composition invariant can be added to the composed-machine's combined event clause, subject to a tool enhancement. The guard will be added as follows,

$$e_a \parallel e_b \triangleq \textbf{ ANY } p_a, p_b \textbf{ WHERE } \boldsymbol{G_{CI}}(v) \wedge\ G_a\ \wedge\ G_b \textbf{ THEN } A_a\ \parallel A_b \textbf{ END}$$

In the example we may want the fifo buffer $f1$ to hold odd numbers, and $f2$ to hold even numbers. This is a property of the composition, and should be specified in the composition invariant clause. To specify this, we add an invariant, stating that all of the values in the producer's $f1$ buffers must have *mod* 2 of 1, and the values of the $f2$ buffers must be *mod* 2 of 0; as in the following,

$$\forall p \cdot p \in dom(f1) \implies (\forall v \cdot v \in ran(buffer(f1(p))) \implies v\ mod\ 2 = 1)$$



Combined Events:
a) Producer.inToBuffOK1 || FIFO.inToBuffOK
b) Producer.inToBuffOK2 || FIFO.inToBuffOK
c) Producer.inToBuffFail1 || FIFO.inToBuffFail
d) Producer.inToBuffFail2 || FIFO.inToBuffFail
e) Consumer.retrvFromBuffOK || FIFO.retrvFromBuffOK
f) Producer.generate

Figure 3: Component Instance Diagram

Proof Obligations We propose to take a *Design-By-Contract* (DBC) [5] view for input and output parameters. In our work the input and output parameters, and their type and direction information, form part of the interface specification. Using this, we can ensure that matching parameter's output values fall within the range of the allowable inputs, by generating proof obligations.

3) Conclusions
In order to make the Even-B approach more flexible, we propose an extension to the existing composition approach, to introduce Event-B components. We add input, and output specifiers, "?" and "!" to event parameters, and extend iUML-B to describe components, and their interfaces. We add annotations, to identify externally visible events, while the remainder are hidden. We ensure communication is feasible, by generating additional precondition-style proof obligations; and provide a mechanism to add additional guards, to discharge the *composition invariant* proof obligations. We plan to investigate the use of components w.r.t. team-working. The parallel development of components, and artefacts within components, is key to making Event-B more agile.

References

[1] The Rodin User's Handbook. Available at http:// handbook.event-b.org/.

[2] J.R. Abrial. *Modeling in Event-B: System and Software Engineering.* Cambridge University Press, 2010.

[3] J.R. Abrial, M. Butler, S. Hallerstede, T.S. Hoang, F. Mehta, and L. Voisin. Rodin: An Open Toolset for Modelling and Reasoning in Event-B. *Software Tools for Technology Transfer*, 12(6):447–466, November 2010.

[4] S. Hallerstede. Justifications for the Event-B Modelling Notation. In J. Julliand and O. Kouchnarenko, editors, *B*, volume 4355 of *Lecture Notes in Computer Science*, pages 49–63. Springer, 2007.

[5] B. Meyer. Design by Contract: The Eiffel Method. In *TOOLS (26)*, page 446. IEEE Computer Society, 1998.

[6] J. Wright R. Back. *Refinement Calculus: a systematic introduction.* Springer Science & Business Media, 2012.

[7] R. Silva. *Supporting Development of Event-B Models.* PhD thesis, University of Southampton, May 2012.

[8] R. Silva and M. Butler. Shared Event Composition/Decomposition in Event-B. In *FMCO Formal Methods for Components and Objects*, November 2010. Event Dates: 29 November - 1 December 2010.

[9] C. Snook and M. Butler. UML-B and Event-B: An Integration of Languages and Tools. In *The IASTED International Conference on Software Engineering - SE2008*, February 2008.

[10] The ADVICeS Team. The ADVICeS Project. available at https://research.it.abo.fi/ADVICeS/.