Towards Modular Development in Event-B

Thai Son Hoang\textsuperscript{1}, Hironobu Kuruma\textsuperscript{2}, and Michael Butler\textsuperscript{1}

\textsuperscript{1} ECS, University of Southampton, U.K.
\textsuperscript{2} Research and Development Group, Hitachi Ltd., Japan

\textbf{Background.} Event-B \cite{2} developments are mostly structured around \textit{refinement} and \textit{decomposition} relationships \cite{3}. This top-down development architecture enables system details to be gradually introduced into the formal model. More often, this result in large model with monolithic structures.

\textbf{Motivation.} Various composition approaches have been proposed \cite{6,4,7,5}. This proposal is inspired by these approaches and development methods such as classical-B \cite{1}, working towards modular development in Event-B.

\textbf{Machine inclusion.} Our first concept is \textit{machine inclusions}. Machine A that includes machine B inherits B’s variables and invariants. Variables of B cannot be modified directly, but only through event’s synchronisation \cite{7}. Multiple instance of the included machine can be achieved via prefixing, similar to \cite{5}. The syntactical “flatten” of A can be seen in Fig. 1.

\begin{verbatim}
machine B
variables y
invariants J(y)

events
event f
  any u where
  G_B(y, u)
then
  BAP_B(y, u, y')
end

machine A
includes p_B
variables x
invariants I(x, p_y)

events
event e
  synchronises p_f
  any t where
  G_A(x, t)
  H_A(x, p_2y, t, p_u)
then
  BAP_A(x, t, x')
end

machine (flatten_A)
variables x, p_y
invariants I(x, p_y)
J(y)

events
event e
  any t, p_u where
  G_A(x, t)
  H_A(x, p_2y, t, p_u)
  G_B(y, u)
then
  BAP_A(x, t, x')
  BAP_B(p_2y, p_2u, p_y')
end

Fig. 1: Machine inclusion
\end{verbatim}

\textbf{Refinement-chain inclusion.} While machine-inclusion mechanism gives us a direct reuse of machine, it is often the case that we want to reuse a refinement-chain. Approaches such as \cite{4,8} allow to incorporate refinement chains in to
the development. However, these approaches involve generating of model which is often cumbersome to accommodate changes. We propose to include the refinement-chain inside the machine itself. Fig. 2 illustrates a situation where A includes a refinement chain from \( B_1 \) to \( B_m \). Semantically, A has double “in-

\[
\text{machine } A \\
\text{includes } B_1 \longrightarrow B_m \\
\ldots \\
\text{event } e \\
\text{synchronises } f \\
\ldots
\]

Fig. 2: Refinement-chain inclusion

terfaces”. To its abstract machine, it acts as a machine with an inclusion of \( B_1 \). To its concrete machine, it acts as a machine with an inclusion of \( B_m \). Since the refinement of \( B_1 \) by \( B_m \) has been proved separately, the refinement of \( (A + B_1) \) by \( (A + B_m) \) is almost correct-by-construction. The extra manual work required is the refinement of the extra guard, e.g., \( H_{AB} \) in Fig. 1.

Some evaluation. We have applied the idea (manually) to several examples. The result is quite encouraging with reduction of the modelling and proving efforts. In particular, this approach can be used several times, e.g., to have a hierarchy of inclusions. The resulting models also easier to comprehend and in particular, it should incorporate with changes to the imported model without any additional effort.

Conclusion. The concept here is not new and incorporate many existing work. We consider this as an effective way to have modular development in Event-B which will reduce the modelling and proving efforts. We are investigating how to extend Rodin to support the approach in the most efficient way.

References


