Modularisation Plugin

A. Iliasov

August 6, 2010

The Modularisation plugin provides facilities for structuring Event B developments into logical units of modelling, called modules. The module concept is close to the notion Event B development (a refinement tree of Event B machines). However, unlike a conventional development, a module comes with an interface. An interface defines the conditions on how a module may be incorporated into another development (that is, another module). The plugin follows an approach where an interface is characterised by a list of operations specifying the services provided by the module. An integration of a module into a main development is accomplished by referring operations from Event B machine actions using the procedure call metaphor [1].

While a specification realised as a single machine may describe fairly large systems such an approach presents a number of limitations. The two structuring mechanisms of a machine are the notions of variable, structuring an overall system state, and the notion of event, structuring the behavioral part of a specification. A sufficiently advanced model would normally correspond to a machine with a substantial number of events and variables. This leads to the scalability problems as the number of events and actions contained in them is linearly proportional to the number of proof obligations.

Another point worth considering is the requirement to a modeller to keep the track of all the details contained in a large model, that is, the readability problem. A large specification lacking any form of structuring is hard to comprehend and thus is also hard to develop. These problems are addressed by structuring a specification into a set *modules*.

In our approach a system is made of a number of modules weaved together so that they work on the same global problem. Being a self-contained piece of specification, a module is reusable across a range of developments. A hypothetical library of modules could facilitate formal developments through the reuse of ready third-party designs. To couple two modules one has to provide the means for a module to benefit from the functionality of the another module. A module interface describes a collection of externally accessible operations; interface variables permit the observation of a module state by other modules.

More information on using the modularisation principles and the plugin functionality can be found in [2, 3].

The talk will present the recent progress in the Modularisation methodological and tool support and report on the application of the Modularisation mechanism in the development of the SSF AOCS case study [4, 5] - one of the case studies offered by the Deploy project industrial partners. The talk will conclude with a short tutorial session discussing some of the more challenging issues such as writing import invariants and using the type instantiation feature.

References

[1] A. Iliasov, E. Troubitsyna, L. Laibinis, A. Romanovsky, K. Varpaaniemi, D. Ilic, and T. Latvala, "Supporting reuse in event b development: Modularisation approach," in *ASM*, ser. Lecture Notes in Computer Science, M. Frappier, U. Glässer, S. Khurshid, R. Laleau, and S. Reeves, Eds., vol. 5977. Springer, 2010, pp. 174–188.

[2] "Modularisation plugin wiki," http://wiki.event-b.org/index.php/Modularisation_Plug-in.

[3] "Modularisation plugin tutorial," http://deploy-eprints.ecs.soton.ac.uk/227/.

[4] A. Iliasov, E. Troubitsyna, L. Laibinis, A. Romanovsky, K. Varpaaniemi, D. Ilic, and T. Latvala, "Developing mode-rich satellite software by refinement in event b," in *FMICS'10*, 2010.

[5] A. Iliasov, A. Romanovsky, E. Troubitsyna, L. Laibinis, K. Varpaaniemi, P. Vaisanen, D. Ilic, and T. Latvala, "Verifying mode consistency for on-board satellite software," in *Safecomp'10*, 2010.