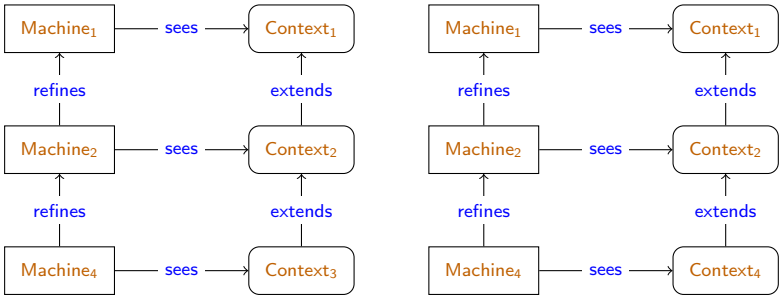
Crossed-Project Reference for Managing Model Variations

Hironobu Kuruma[1] and Thai Son Hoang[2]

[1] Research and Development Group, Hitachi Ltd., Japan
[2] ECS, University of Southamtpon, U.K.

Background. A model in Event-B [1] (typically located within a project) is composed of components combined using refines, extends or sees relations. Fig. 1 shows an example of model variations containing some common machines and contexts, i.e., $\mathsf{Machine}_1$, $\mathsf{Machine}_2$, $\mathsf{Context}_1$ and $\mathsf{Context}_2$. $\mathsf{Machine}_3$ and $\mathsf{Context}_3$ (resp. $\mathsf{Machine}_4$ and $\mathsf{Context}_4$) are additional component specific to project **A** (resp. **B**).



(a) Project **A** (b) Project **B**

Fig. 1: A family of models

Motivation. To maintain these models, it is desirable to separate the common components and to share them between the projects. In Fig. 2, project **C** is for the shared components, where the original projects **A** and **B** contain only the additional components. This structure is beneficial for composing a family of models that has common properties by sharing components.

Concept. We experimentally introduced a crossed-project reference mechanism into the RODIN platform [2] to manage collections of components and enable reference to components in different projects. The crossed-project reference mechanism uses a manifest to identify components to be imported from other projects (Fig. 2). The names of imported components are prefixed with their project name. The components in importing project refer the imported components by their names declared in the manifest. The imports relation between projects must be acyclic.

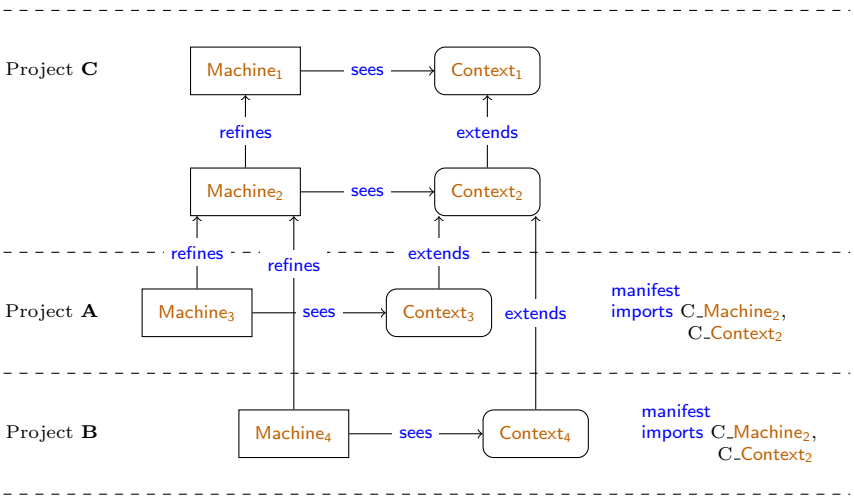


Fig. 2: Crossed-project Reference

Implementation. We implemented the cross project reference by *renaming and copying the statically checked files* of imported components into the importing project. In our example, two files in **C**, i.e., the statically checked files of $\mathsf{Machine}_2$ and $\mathsf{Context}_2$, are renamed to by prefexing, and copied into **A** and **B**. Since the imported components are expected to be verified in their source project, verification is required only for the additional components.

Conclusion and Limitation. The crossed-project reference mechanism reduces the risk of unintended modification of components since it separates the components into hierarchical collections. This property is useful for managing model variations. However, our implementation is experimental and insufficient for practical usage. Most RODIN plug-ins refer the unchecked file of components and are incompatible with our implementation. For example, the components that refer imported components cannot be easily edited by ordinary editors because they do not recognize the imported components. Although we expect the generic instantiation [3] is an effective way to compose variations, the generic instantiation plug-in also uses the unchecked files and does not work with the crossed-project reference. Furthermore, since our implementation does not propagate the change of original components, the imported file must be updated manually in the importing project. The propagation of the change of components and the collaboration with the generic instantiation are our future work.

References

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

2. 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.
3. Andreas Fürst, Thai Son Hoang, David Basin, Naoto Sato, and Kunihiko Miyazaki. Formal system modelling using abstract data types in Event-B. In Yamine Aït Ameur and Klaus-Dieter Schewe, editors, *ABZ*, volume 8477 of *Lecture Notes in Computer Science*, pages 222–237, Toulouse, France, June 2014. Springer-Verlag. http://dx.doi.org/10.1007/978-3-662-43652-3_20.