Breakout Session 2 on Bidirectional Programming in Self-Adaptive SystemsRecorded by Chair Lionel Montrieux, NII, Japan

We discussed the use of bidirectional programs, and bidirectional transformations, in the context of self-adaptive systems. We identified 5 areas of interest, connected to the participants' research.

** Model abstractions

Bidirectional programs can be used to transform a concrete, platform-specific model of the system into an abstract, platform-independent model of the system. Adaptation can then happen on the abstract model, and changes will be propagated to the concrete model. This allows developers to support heterogeneous environments, and to migrate from one implementation to another, or from one version to another, without having to update their self-adaptation architecture.

** Extraction of sub-models for efficient analysis

A large model (abstract or concrete) can be expensive to analyse. Using bidirectional programs, we can extract a portion of the model for a particular analysis. This can be done multiple times, and each of these views can be used by a separate MAPE loop. Synchronisation between the views is relatively simple: every 'put' to the large model can trigger a new 'get' to each view that could be affected by the changes made.

We could go further. If the amount of data needed by a particular MAPE loop can vary, it should be possible to adapt the transformation at runtime to narrow the view, giving the mape loop the smallest view possible, all from a single bidirectional program. If the program describes a transformation over the largest view that the MAPE loop could need, it is trivial to automatically generate a transformation that produces a subset of the largest possible view.

One of the participants likened the extraction of small views to the concept of crosscutting concerns in aspect-oriented programming. This is an interesting point of view to explore, and it may lead to more interesting uses of bidirectional programs.

** Beyond self-configuration: current state of the system vs. desired state of the system

Modifying system models in the context of self-configuration is relatively easy: if the part of the model that represents the system can be entirely translated into configuration files, then effecting the changes is as simple as updating the configuration files, and possibly reloading the system to take the new configuration into account.

However, in general, changes to the model may not always be effected by changes to configuration files. For example, changes may have to be

done through an API. Those changes may or may not succeed, and hence failures must be taken into account. In such a solution, the model may represent the /current/ state of the system (if the model is extracted from the system and the environment), but if modified, it then represents the /desired/ state of the system, until (and if...) the modifications are successfully reflected in the system.

Bidirectional programs can help deal with this. A program can be written to isolate the changes to be made, and to keep track of their results. Two very similar programs can then produce the /current/ model of the system and the /desired/ model of the system.

** Bidirectional programs and context-oriented programming/self-adaptation

We discussed the 'traditional' self-adaptation model, where gauges and probes capture the state of the system and its environment, and effectors act on the system or its environment to enact adaptation. In general, the effectors can be completely different from the gauges and probes. However, if we were able to describe relationships between these, we may be able to configure and deploy compatible pairs of gauges/probes and effectors to achieve adaptation.

** Bidirectional programs for partial model

Partial models can represent alternatives to choose from. It should be possible to write bidirectional programs to synchronise each partial model with the overall model, keep then synchronised, and handle conflicts and merging.