Modular Development of Control and Computational Modules using Reactive Objects

Frédéric Boulanger[1] and Guy Vidal-Naquet[1,2]

[1] Supélec–Service Informatique, Plateau de Moulon, F-91192 Gif-sur-Yvette cedex,
{Frederic.Boulanger | Guy.Vidal-Naquet}@supelec.fr,
http://www.supelec.fr

[2] Laboratoire de Recherche en Informatique, CNRS – Université de Paris-Sud,
F-91405 Orsay cedex

Abstract. In this paper, we show how a clear separation between control and data processing in reactive applications increases the range of application of the proof tools of the synchronous reactive model and of the reuse mechanism of the object-oriented approach. We present an approach and some tools that allow this separation in the Ptolemy framework.

1 Introduction

Aside a few particular cases, an application is composed of computations (transformations of data) and control (choice and parameterization of computations). These two aspects are often intertwined. Consider, for example, the following simple instruction: `if (s > threshold)`. Here, `if` is control since it enables to choose between two computations, on the other hand `s > threshold` is a computation whose result is used by the control.

If we want to prove that when the speed `s` is greater than `threshold`, a given signal is emitted, the computation induced by this test must be done outside the control module, so that only a boolean occurs as an argument of the `if`. Furthermore, if the speed must be in a certain range, instead of being less than `threshold`, the change will not concern the control module — which might have been validated — but only the modules that evaluates the condition. Moreover, if the evaluation of the control criterion is complex, one will be able to use standard libraries, and will not have to take into account what is provided by the language used for the control module when writing the code associated with the computation.

This example illustrates two major inconveniences that arise when control and computations are intertwined: the scope of the validation tools is limited by the occurrence of data in the control, and the reuse of computations modules is limited by the need to change the way they are controlled.

We propose an approach and tools for the modular development of control and computation modules. The main difficulty is to make explicit the retro-action loops that are often hidden inside the computations.

One advantage of this approach is that it enables one to choose the best paradigm for each component. We will use the synchronous reactive approach [2] for control, and a synchronous (in its signal processing meaning[1]) data flow approach for data processing.

Such a modular development brings the need of a communication interface between modules, and an execution machine that enables them to run together [1]. We have based the associated tools on the PTOLEMY [3] platform[2], developed at the University of California at Berkeley. This platform uses objects to enable the integration of several computation paradigms called "domains". A major benefit coming from the use of PTOLEMY is that it is possible to use several domains within the same application, and that a domain used for simulation, for instance "Synchronous Data Flow", may have a corresponding dual domain used for code generation, for instance "Code Generation C"). It is therefore possible to simulate a system, then to generate the corresponding code just by choosing the dual domain.

2 Control and Computation

The distinction between control and computation is not as clear as it is shown in the above example, and it cannot be made by considering only syntactical aspects. During the conception of the application, decisions must be taken on what will be considered as control and what will be considered as computations. When verification tools are used on a module, it means that this module has already been identified as control, and it is then easy to take away all computations.

The main difficulty is to identify control that is embedded into computations, for instance, adaptive filters that change their coefficients according to their input.

A possible criterion for the identification of control is that it changes the way data is handled, so some programming language control statements may not be considered as control in some contexts—think of a `for` loop used to compute the scalar product of two vectors.

The approach we advocate is therefore to introduce an explicit step where control modules are identified in the development of an application. All computations are then removed from these modules and put into specific computation modules or implemented using standard modules from libraries. Thus, control modules will only receive and produce boolean values, and it will be possible to use formal verification tools on them. This would be very difficult or even impossible to do if the module had to deal with data types even as simple as integers.

---

[1] data production and consumption rates are in fixed ratio.

[2] http://ptolemy.eecs.berkeley.edu

3 Example: A cruise controller

Although this example has been completely treated, we will only discuss its results for the sake of brevity.

We consider a cruise control system for a car, and whatever the sophistication of the system, it is mandatory to prove that an action on the brakes disables the cruise control.

A sensor gives the position of the brake pedal as, for instance, an integer between 0 and 255. If the control module receives this raw information, it must compare the position of the pedal to its default position. The number of possible states for the control module is therefore multiplied by the number of values an integer can take (or at least by 256 if the verification tool is clever). With several valued inputs, the number of possible states can become so large that no tool could explore them in realistic time and memory. Moreover, to handle such values, the verification tool must have a formal specification of their data type—it must know what `pedal > 0` means.

By processing the comparison in a computation module and feeding the control module with the boolean result of the comparison, we could formally check that the dangerous state—active control *and* brakes—could not be reached for any combination and history of the inputs. We are therefore sure that the design of this control module is correct with respect to this point.

4 Integration of Control and Computation modules

Separate development of control and computations leads to another difficulty: they must finally be integrated to build the application. In this example, we use a simple method, where the control modules are written in ESTEREL or LUSTRE and translated into SDF or CGC. We have developed a translator OCPL[3] from the OC state machine produced by the ESTEREL or LUSTRE compiler to the PL language that describes stars (basic entities of PTOLEMY).

The semantics of the communications between control and data processing is therefore built into the translator. Moreover, with this approach, all data processing modules run continuously, and the control modules select the right outputs according to the current mode. This is not very efficient because some time is wasted to compute outputs that are not used.

Another approach is to use the "Synchronous Reactive" domain to specify the control, and the SDF domain to specify the computations.

SR [4] allows to build synchronous reactive systems by assembling components. At the start of each instant, all signals are unknown excepted the inputs of the system. The components are activated according to a statically determined schedule, and each time they are activated, they produce as much output as they can from their known inputs. The final values of the signals are the least fixed point of the system.

[3] available by anonymous ftp on `ftp://ftp.supelec.fr/pub/cs/distrib/`.

SR is a good candidate for writing control modules in PTOLEMY, and we have developed the dual SRCGC domain (for C code generation) in collaboration with Thomson-CSF Optronique. This way, a complete application can be built, with control developed in SR or SRCGC, and data processing in SDF or CGC.

Unfortunately, we can only generate "strict" SR stars from OC (strict stars can react only when *all* their inputs are known), so we have developed a new SSCPL code generation tool based on the SSC format of ESTEREL. This format contains information on the dependencies between signals, and it is therefore possible to compute some outputs without knowing all inputs.

With this approach, the semantics of the communications between control and data processing is implemented by special interfaces called "Worm holes" in PTOLEMY. This allows the control to change the schedule of computations, and therefore to trigger only the necessary computations. However, a few technical issues prevent this scheme from working with the code generation domains, but they should be solved soon.

5 Conclusion

The object oriented approach used in PTOLEMY allowed us to integrate control modules written in ESTEREL or LUSTRE with data processing modules written in SDF, both for simulation (use of the SDF domain) and code generation (use of the CGC domain). Work remains to be done for the full exploitation of this approach, both at the methodology level and at the tool level. The immediate benefits are more structured code and the ability to check properties on the control. In the long term, we should benefit from well designed libraries of data processing and control components.

References

1. C. André, F. Boulanger, M.A. Péraldi, J.P. Rigault, G. Vidal-Naquet
   Objects and Synchronous Programming
   European Journal of Automation, Vol. 31, # 3/1997, p.418-432
2. A. Benveniste, G. Berry
   The Synchronous Approach to Reactive and Real-Time Systems.
   Proceedings of the IEEE, vol 4, April 1994
3. J.T. Buck, S. Ha, E.A. Lee, D.G. Messerschmitt
   PTOLEMY, a Framework for Simulating and
   Prototyping Heterogeneous Systems
   International Journal of Computer Simulation, 19(2):87-152, November 1992.
4. S.A. Edwards
   The Specification and Execution of
   Heterogeneous Synchronous Reactive Systems
   Ph.D. thesis, University of California, Berkeley, May 1997