Chapter 9 from: C. Brooks, E. A. Lee, X. Liu, S. Neuendorffer, Y. Zhao, H. Zheng "Heterogeneous Concurrent Modeling and Design in Java (Volume 3: Ptolemy II Domains)," Technical Memorandum UCB/ERL M04/17, University of California, Berkeley, CA USA 94720, June 24, 2004.

9

PSDF Domain

Author: *Steve Neuendorffer*
Contributor: *Shuvra Bhattacharyya*

9.1 Purpose of the Domain

The parameterized synchronous dataflow (PSDF) domain is useful for modeling dataflow systems with reconfiguration. Reconfiguration allows more interesting systems to be modeled without resort to more generalized dataflow models. As such, the PSDF domain represents a design point between complete static scheduling in the synchronous dataflow (SDF) domain, and completely dynamic execution under the process networks (PN) domain. Under the PSDF domain, symbolic analysis of the model is used to generate a quasi-static schedule that statically determines an execution order of actors, but dynamically determines the number of times each actor fires. The quasi-static schedule guarantees bounded memory execution as long as the parameters that the schedule depends on are bounded. Additionally, the model is guaranteed to be deadlock free, regardless of the parameter values. The primary disadvantage of the PSDF domain is that current scheduling techniques are only valid for acyclic graphs.

9.2 Using PSDF

There are several issues that must be addressed when using the PSDF domain, in addition to the normal issues of synchronous dataflow models:

- Restricted reconfiguration
- Symbolic scheduling limitations

This section will present a short description of these issues. For a more complete description, see section 9.3.

9.2.1 Restricted Reconfiguration

Consider the SDF model shown in figure 9.1. This model displays arrays of various sizes. Every time the SequenceToArray actor fires, it consumes an integer indicating the size of the next array from the Ramp actor and then consumes that many tokens from its top input port and wraps them into an array. The SequenceToArray actor in this case is obviously not valid in SDF, since its rates change. This particular model is also not a valid PSDF model, although with some modifications it can be made valid.

The difficulty with the below model arises because the schedule for the model depends on the run-time value consumed from the Ramp actor. However, this value is not consumed (or even produced) until after the schedule has begun executing. In the terminology of parameterized synchronous data-flow, the model is not *locally synchronous*[15]. One solution is to build the model hierarchically, as shown in figure 9.2. In this model, the Ramp actor executes first, and reconfigures the PSDF model. The reconfiguration happens before the PSDF schedule begins executing, and the quasi-static schedule is instantiated with the correct token rates. Another possible solution would be to make the model a process network instead, but this would preclude static scheduling and efficient code generation.

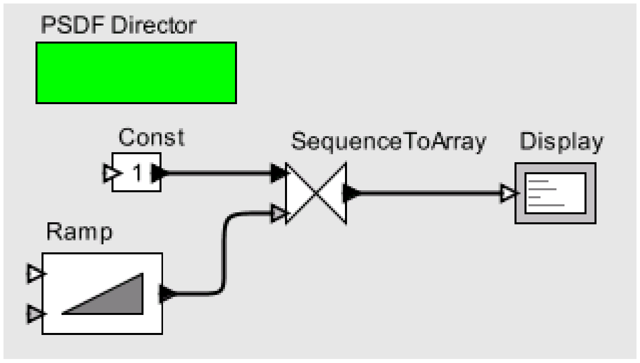

FIGURE 9.1. A PSDF model that is not locally synchronous.

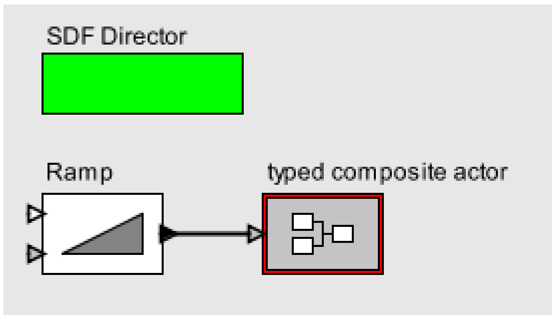

FIGURE 9.2. The model of figure 9.1 corrected using hierarchy.

9.2.2 Symbolic scheduling limitations

Parameterized synchronous dataflow scheduling results in a quasi-static schedule. In contrast to synchronous dataflow schedules, which are a simple sequence of actor firings, a quasi-static schedule is more of a program that generates a sequence at runtime. Essentially, the goal of the PSDF scheduler is to find a fixed program that will generate the correct schedules for any parameter value. PSDF scheduling for models without feedback is reasonably understood. Additionally, in models where feedback does not restrict scheduling, the feedback can be eliminated for parameterized scheduling. However, for models with tight feedback loops scheduling is not quite as simple and cannot be eliminated. The current scheduler implementation provides none of these improvements and only schedules models without feedback.

Another limitation of symbolic scheduling is that it must rely on a general symbolic math package to guarantee a valid schedule. One solution is to provide side information to the model, to assist with symbolic simplification. The current scheduler implementation provides no mechanism for this.

9.3 Properties of the PSDF domain

PSDF is an untimed model of computation, similar to the SDF domain (see Chapter 3). The key improvement is that the PSDF domain generates a schedule by considering the token consumption and production rates symbolically. Data rates on each port are allowed to change through reconfiguration, as long as reconfiguration does not occur during execution of the schedule, implying that the model is locally synchronous. Reconfiguration analysis is performed to statically determine when rate parameters change to ensure a valid schedule.

9.3.1 Scheduling

The first step in constructing synchronous dataflow schedules is to solve the *balance equations* [84]. These equations determine the number of times each actor will fire during an iteration. Ordinarily, these equations are solved for a numeric solution, and an equivalent schedule is generated. In the PSDF domain, the balance equations are solved for a symbolic solution in the other variables. The symbolic solutions are converted into a quasi-static schedule where the number of firings of each actor is computed at run-time. For example, consider the model in figure 9.3. This model implies the following system of equations, where *ProductionRate* and *ConsumptionRate* are declared properties of each port, and *Firings* is a property of each actor that will be solved for:

$$\mathit{Firings}(\mathrm{A}) \times \mathit{ProductionRate}(\mathrm{A1}) = \mathit{Firings}(\mathrm{B}) \times \mathit{ConsumptionRate}(\mathrm{B1})$$

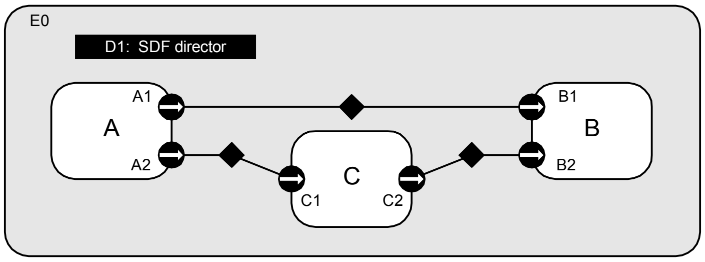

FIGURE 9.3. An example SDF model.

$$\mathit{Firings}(\mathrm{A}) \times \mathit{ProductionRate}(\mathrm{A2}) = \mathit{Firings}(\mathrm{C}) \times \mathit{ConsumptionRate}(\mathrm{C1})$$

$$\mathit{Firings}(\mathrm{C}) \times \mathit{ProductionRate}(\mathrm{C2}) = \mathit{Firings}(\mathrm{B}) \times \mathit{ConsumptionRate}(\mathrm{B2})$$

These equations express constraints that the number of tokens created on a relation during an iteration is equal to the number of tokens consumed. A symbolic solution to these equations asserts, for instance, that:

$$\mathit{Firings}(\mathrm{A}) = \mathit{ConsumptionRate}(\mathrm{B1}) / (\mathrm{gcd}(\mathit{ConsumptionRate}(\mathrm{C1}) * \mathit{ProductionRate}(\mathrm{A1}) / \mathrm{gcd}(\mathit{ProductionRate}(\mathrm{A2}), \mathit{ConsumptionRate}(\mathrm{C1})), \mathit{ConsumptionRate}(\mathrm{B1}))) * \mathit{ConsumptionRate}(\mathrm{C1}) / \mathrm{gcd}(\mathit{ProductionRate}(\mathrm{A2}), \mathit{ConsumptionRate}(\mathrm{C1}))$$

where gcd(a,b) is the greatest common divisor of a and b. As with SDF models, it is possible for a model to be inconsistent, where no symbolic solution to the balance equations exists.

The second step in constructing a PSDF schedule is dataflow analysis. Dataflow analysis in an ordinary synchronous dataflow model orders the firing of actors, based on the relations between them. In SDF scheduling, dataflow analysis involves simulation of the schedule execution until each actor has been fired the required number of times. In PSDF scheduling, the situation is more complex, since the number of times an actor will fire is not known beforehand. To simplify the problem, we assume that each actor appears exactly once in the schedule, and executes the number of times according to the symbolic balance equations. This simplification cannot be done for all models with feedback, but can always be done for models without feedback, hence the restriction mentioned in the previous section.

9.3.2 Local Synchrony and Reconfiguration Analysis

Although rate parameters in PSDF models are allowed to change between schedule iterations, they must not change during the execution of a schedule. This property is known as local synchrony. In order to ensure local synchrony of PSDF models, reconfiguration analysis is performed during scheduling. This reconfiguration analysis combines information on parameter dependency (which parameters depend on other parameters) and reconfiguration (which models modify parameter values). The result of reconfiguration analysis is the least change context of a parameter. The least change context of a parameter is a composite actor that contains the parameter, reconfigures the parameter, and is contained by all other composite actors that reconfigure the parameter. Local synchrony requires that rate parameters used for scheduling are either not reconfigured, or have a least change context that contains the PSDF model or is the PSDF model.

Reconfiguration analysis considers the following possible types of reconfiguration:

Reconfiguration ports: PortParameters are reconfigured when tokens are received from a ParameterPort.

Modal Models: Parameters may be reconfigured in finite-state machine transitions.

Reconfiguration analysis is performed in a similar matter to data type analysis. A set of inequalities are extracted from a model where each inequality represents a constraint on the least change context of a parameter. One class of constraint requires that the least change context of a parameter P cannot be any higher in the hierarchy than the least change context of a parameter that P depends on. The second constraint requires that if a parameter is reconfigured by an actor, then the actor must contain the least change context of the parameter. These constraints are solved to determine the least change context of every parameter. The PSDF scheduler uses this information to check for local synchrony of the model and to reduce symbolic computations where possible if some rate parameters are not reconfigured. The algorithm is described more completely in [99].

9.4 Software Architecture

The PSDF kernel package implements the PSDF model of computation. The structure of this package is similar to the structure of the SDF kernel package. The PSDF scheduler relies on a scheduling library built by students at the University of Maryland. This library implements a parameterized version of the APGAN algorithm and makes relatively few optimizations to the symbolic schedule. Unlike the SDF domain, the PSDF domain computes external port rates which may be symbolic. The SDF domain also leverages reconfiguration analysis to detect such cases and report a modeling error.

Reconfiguration analysis is implemented by the ptolemy.actor.util.ConstVariableModelAnalysis class. This analysis operates globally on a model and is intended to be used during the preinitialization phase of execution by schedulers. The ConstVariableModelAnalysis class relies on declarations by objects in a model which directly invoke the setToken() method on other parameters during execution of a model. The ptolemy.domains.fsm.FSMActor class and the ptolemy.actor.parameters.ParameterPort class implement the ptolemy.actor.util.ExplicitChangeContext interface to declare that they modify parameters directly. On some occasions, an object might detect a changed parameter, such as through the attributeChanged() method and modify another parameter. Such objects are expected to use the ptolemy.actor.util.DependencyDeclaration attribute to declare this parameter dependence, which would otherwise not be visible to the reconfiguration analysis. Dependencies which arise through parameter expressions are directly understood by the reconfiguration analysis and do not need to be explicitly declared.

9.5 Actors

Most domain-polymorphic actors can be used under the PSDF domain, with similar restrictions as with the SDF domain. The SampleDelay actor can be used in PSDF models, although models with feedback are not.