Reading Report: Modeling and Simulation of Heterogeneous System in Ptolemy

Yanwar Asrigo

McGill University, School of Computer Science
yanwar.asrigo@mail.mcgill.ca

Abstract. Many modern systems are heterogeneous systems as they are composed of different subsystems with very different characteristics. Consequently, the modeling environment used to design such system has to support this heterogeneity. This report describe how this problem can be solved in Ptolemy. Ptolemy's *hierarchical heterogeneity* approach in modeling heterogeneous systems will be described. Finally, a high level explanation of how this approach can be realized is provided.

1 Introduction

Building a complex system straight from functional specification to implementation is a daunting, if not impossible, task. One solution to bridge this gap is to create a model of the system to be developed. The benefit of having a model of a system is two folds. First, it helps the development cycle and system maintenance or modification. Second, model allows system designer to check the correctness of the system even before it is built. An important factor in modeling a system is choosing the right level of abstraction and the right formalism. This decision is determined by the system designer's intention. For example, finite state machines (FSM) have long been used to describe and analyze intricate control sequences.

There are also systems that are heterogeneous. Such systems are composed by subsystems that have different characteristics and communicate and interact in a variety of ways. In this case, different formalisms have to be used for different subsystems to naturally model them. For instance, a power window system that has a motor which is controlled by a micro-controller. Since there are only disparate models of the subsystems, they can only be evaluated independently of the others. Holistic evaluation of the system would require the models of the subsystems to be composed together. This *heterogeneous composition* should faithfully represent the interaction and communication between models while maintaining the properties of each individual model.

The most common approach in realizing the composition is brute-force composition. The designer of the system would have to manually define the relationships of the models and the interaction between them. This approach may result in surprising interactions, violating properties, and ambiguities that are unforeseen by the designers which render the whole system analysis unreliable.

2 Heterogeneous Composition in Ptolemy

Ptolemy[1] is designed to address the challenges of heterogeneous composition by introducing a more structured approach to compose different models, i.e. *hierarchically heterogeneous*. In Ptolemy, complex model are hierarchically decomposed where each level of decomposition contains a network of interacting components. Each level serves as a refinement for a component from the layer above. Each local network is governed by a single interaction mechanism; However, different interaction mechanisms can be specified at different levels in the hierarchy. Figure 1 shows an example of hierarchically heterogeneous model.

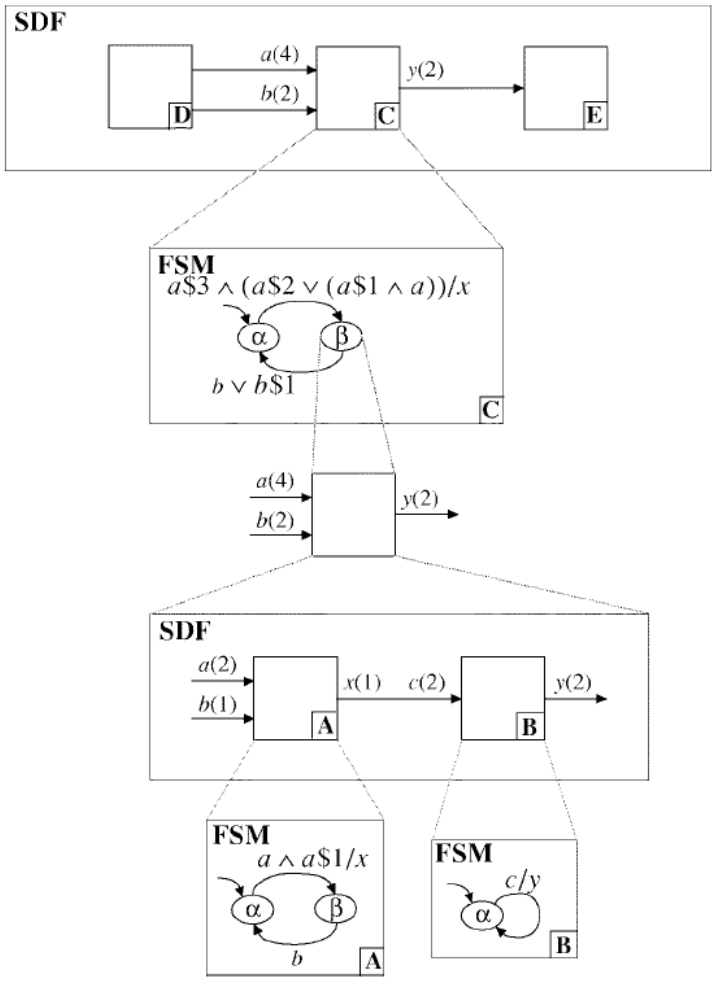

Fig. 1. Hierarchical heterogeneous model with two FSMs embedded in a Synchronous Data Flow (SDF) system. Taken from [2].

A fundamental concept of hierarchical heterogeneity is the clear separation of flow of data and the flow of control in defining components interaction in a local network. Such a framework is called a *model of computation* (MoC) as it attaches a semantic to a network structure by defining how computation takes places among computational components in the structure. On the other hand, MoC must be compositional as well to support hierarchical composition. MoC must be able to encapsulate the network of components it described into a component that may then be composed again with other components, possibly under a different MoC.

Ptolemy adopts an *actor-oriented* view of a system, where actor forms the basic building block of a system. *Actors* are concurrent components that communicate through interfaces called ports by passing messages. Interconnection between these ports forms the communication structure between actors. Contrast with object orientation whereby control flows through an object in a sequential manner through method calls, actor orientation emphasizes on the flow of stream of data into and out of every actor. Every actor encapsulates an abstract functionality. Actor orientation enables the decoupling of the transmission of data from the transfer of control in Prolemy.

To support the hierarchical composition, an actor can be *atomic*, which is a component at the bottom of the hierarchy, or *composite*, which aggregates other actors. In the hierarchical model illustrated in Fig. 2, the top level composite actor contains actors A1 and A2. Actor A2 is also a composite actor which contains actrs A3 and A4. Actors A1, A3, and A4 are atomic actors. In Ptolemy, actors atomic executions (called *iterations*) consist of 3 phases: *prefire*, *fire*, and *postfire*. This is to provide the flexibility to MoC of a composite actor in managing the relation of actors iteration in the same composite. The prefire phase checks the preconditions for the actor to execute. The fire phase is when an actor typically performs its computation. However, the persistent state of the actor is updated only in the *postfire* phase.

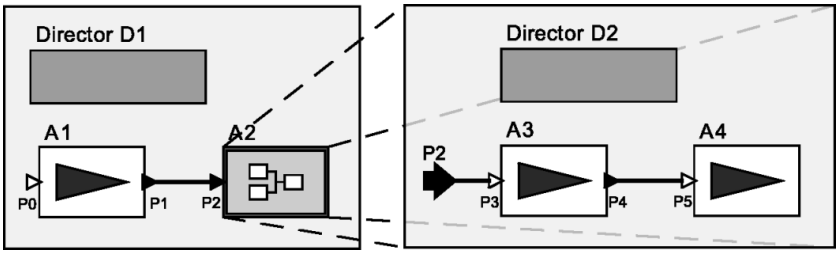

Fig. 2. An example of Hierarchical Model in Ptolemy. Taken from [1].

The syntactic structure of actors does not have any semantic until it is associated with a MoC. In Ptolemy, an implementation of a MoC associated with a composite actor is called a *domain*. A domain consists of two classes: a *director*

and a *receiver* class. The receivers implement the communication mechanisms used in the domain. Note that receivers are contained in input ports and there is only one receiver assigned for each communication channel. When an actor is assigned to a domain, it obtains the domain specific receivers at its input ports. A director is responsible for managing the execution order of the actors contained in a composite. In the Fig. 2 example, it is a hierarchical model using two different domains. The top-level composite actor is controlled by director D1, and A2 is a composite actor with director D2. Director D1 controls the execution order of actors A1 and A2, and director D2 controls the excution of A3 and A4 when A2 is executed. This separation of computation and communication allows actors to be reused in different domains (called *domain-polymorphisms*). Many well known domains have been implemented in Ptolemy, such as Communicating Sequential Processes (CSP), Continuous Time (CT), Discrete Event (DE), Synchronous Dataflow (SDF), and Finite State Machine (FSM).

Domain helps actors to interface with their environment as all actors rely on a common notion of execution (atomic firing) and a common notion of communication (receivers). Thus, actors and models can ideally be embedded into any other model. Bevertheless, this is only true to a certain extent because the realization of the domains may not be compatible between one and another. For example, an actor in SDF domain will only be fired when there are tokens in its receivers, but this is not guaranteed in DE domain. Hence, using an SDF actor in DE domain may cause an exception while executing fire(). Ptolemy overcomes this by performing static compatibility test of an actor with the domain it is embedded in. In this test, an interface automaton is used to model the combined role of receiver and director of each domain. This automaton is called *domain automaton*. Figure 3 shows the SDF domain automaton. The behavior of the actor is also described using an automaton (*actor automaton*). Figure 4 shows the automaton of SDF actor. The static compatibility test is performed by composing the domain automata with the actor automata. The actor is compatible with the domain if the resulting composition is not empty. The result of composing SDF domain and SDF actor automata can be seen in Figure 5.

3 Conclusion

Ptolemy proposes a new approach in modeling and simulating heterogeneous system using a structured approach, i.e. an actor-oriented, hierarchically heterogeneous approach. Central to this approach is the notion of a domain, which implements a particular MoC. By using concepts of actor orientation and domain, Ptolemy is able to separate the flow of control and data between components from the actual functionality of individual. This decoupling facilitates hierarchical modeling in composing different models. Hierarchy enables localization of analyzability. Particularly, each local submodal can be understood and analyzed solely in terms of the assigned MoC. At the next higher level of hierarchy, this sub-model is in turn considered atomic for the purpose of understanding or analyzing its containing model. This localization of analyzability allows individ-

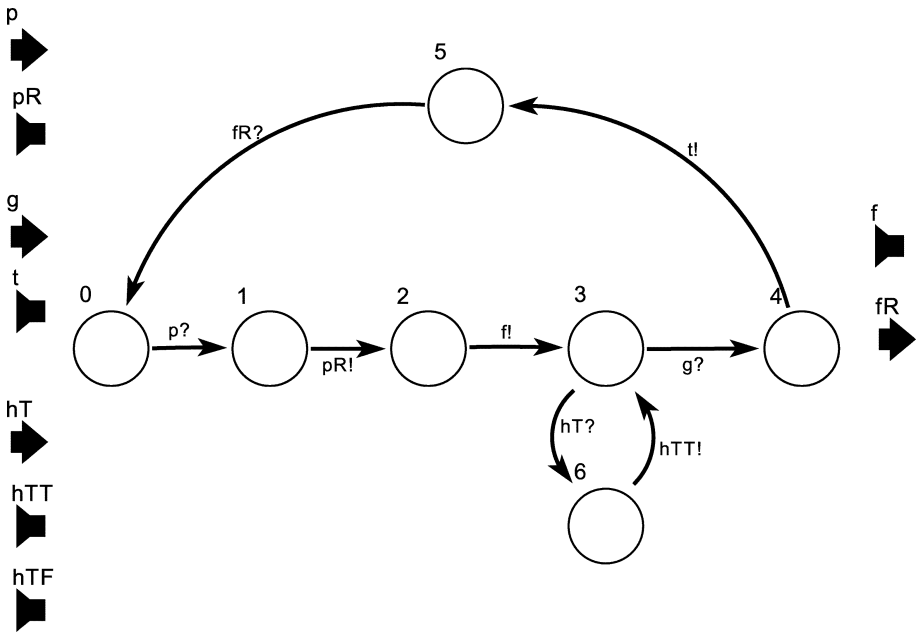

Fig. 3. SDF domain automaton. Taken from [1].

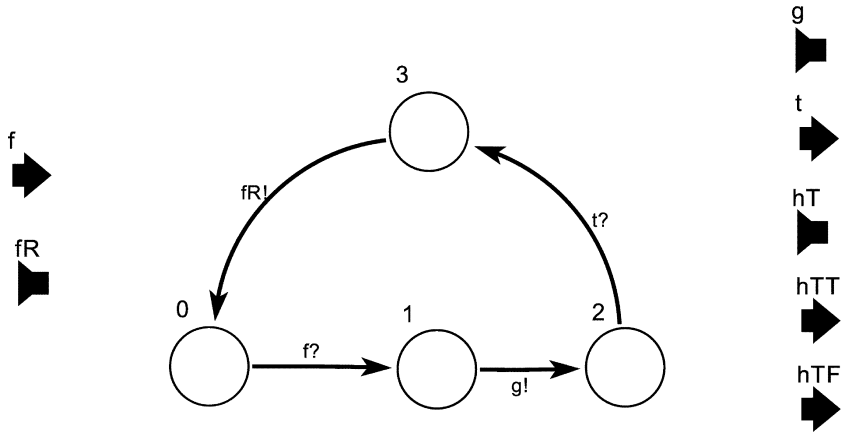

Fig. 4. Interface automaton model for an SDF actor. Taken from [1].

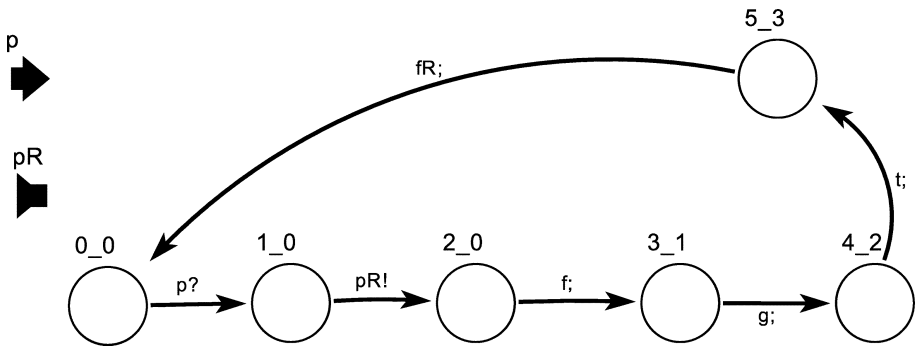

Fig. 5. Composition of SDF domain and SDF actor automata. Taken from [1].

ual components to be refined into more detailed models without affecting the properties of the rest of the system.

Another benefit of factoring the flow of control and data from the individual functionality of the component is the increase of reusability of components and models because they can assume different semantics depending on the MoC assigned to the network. This concept is known as domain polymorphisms. However, this also introduces compatibility issues of actors with domains, and the compositionality of domains. These issues are caused by the variation in the dynamic interaction between actors and domains. These interactions can actually be captured using interface automata. The compositionality of these automata precisely defines the compatibility of actors with domains.

References

1. Eker, J., Janneck, J., Lee, E., Liu, J., Liu, X., Ludvig, J., Neuendorffer, S., Sachs, S., Xiong, Y.: Taming heterogeneity - the ptolemy approach. Proceedings of the IEEE **91**(1) (Jan 2003) 127–144
2. Girault, A., Lee, B., Lee, E.: Hierarchical finite state machines with multiple concurrency models. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on **18**(6) (Jun 1999) 742–760