Distributed Execution of Ptolemy Models

Yang Zhao and Thomas H. Feng

EECS 290N Report
November 18, 2004

University of California at Berkeley
Berkeley, CA, 94720, USA
{ellen zh, tfeng} @ eecs.berkeley.edu

Abstract

In this paper, we offer an overview of distributed model execution with Ptolemy. Our discussion focuses on the Process Network domain, while the platform that we propose is domain-independent. Components such as DNS and PTA-gents are introduced as separate processes in addition to Ptolemy with models running in it. They provide services important to the execution of the system, such as dynamic name lookup, data relay, and life-time measurement.

1 Introduction

Distributed modeling and simulation are an active research area. It has received increasing concern, particularly because high-speed networks and cluster computer systems become widely available. There are many good reasons for using such distributed systems in model execution, some of which are:

- to make full use of the combined power of the computing resources available in a network by involving them in a simulation, which is too slow or too large to fit into a single affordable machine;
- to simulate a real application featuring the ability of high-speed distributed computation;
- to build distributed collaborative shared data spaces; and
- to model the behavior of a P2P (peer-to-peer) or CS (client-server) network, to analyze its performance with a simulation identical to its real behavior, and finally to optimize it and re-engineer it.

Ptolemy II is a modeling, design, and simulation framework suitable for concurrent systems [11]. It supports a component-based construction of systems and it uses well-defined models of computation to govern the interaction between components. To leverage this framework to provide a formal specification and run-time environment for distributed systems, and to incorporate the ability of high-speed distributed computation into Ptolemy II, we develop a platform for handling distributed deployment, remote interaction, and life-cycle management. We also explore how to integrate it with Ptolemy II cleanly (without substantial modification to the current framework) to execute a distributed system obeying some model of computation. We begin by defining the problem and requirements in section 2. We then discuss the platform design in section 3. Section 4 addresses some implementation issues. Section 5 concludes the paper.

2 Problem Statement

The goal of this project is to extend the Ptolemy framework to support distributed computation without requiring the user to know much techniques of distributed computation or to build up the low level communication mechanism from scratch. Assume a user has designed a system as a Ptolemy model and wants to distributedly execute it on multiple machines. The problem we address here is how to distribute the model transparently without requiring the user to modify the design, how to automatically deploy the system according to specific configurations, how to set up the remote communication mechanism, how to manage the execution life-cycle, and how to provide monitoring support for the user to monitor and debug the system.

We assume there are a set of computers in a network.
that are available for executing Ptolemy models. This set of computers may change over time due to computers connecting and disconnecting. In each of these computers, a entity called PtAgent is running, which is used for model distribution and remote communication. There is a global domain name service (DNS), and each PtAgent has the knowledge of locating it in order to find other PtAgents. Both the DNS and the PtAgents are implemented as separate processes interacting with each other with a network protocol, such as Java RMI, CORBA and HTTP.

A user can design a system as a Ptolemy model on any of these computers. A configuration specifies where to execute which actor, and whether an actor should be replicated to multiple computers. The configuration can either be manually fed in by the user or be determined by an algorithm based on some criteria, e.g. load balancing. The user is then able to press some button in Ptolemy and request to distribute the model and execute it automatically. We call the model for the whole system the master model. During the execution, the user may monitor the progress from the master model.

In order to support automatic deployment and execution, a set of tasks need to be addressed:

- **Discovery Mechanism** for each component to discover available resources.
- **Parser** to parse the master model and generate the sub-models that will be deployed on the hosts according to the given configuration. This can be done by applying an XSLT transformer to the MoML file of the master model.
- **Launch Service** to launch a model on request and manage its execution.
- **Remote Communication** for handling distributed interaction.
- **Monitor Service** for the user to monitor the execution.
- **Fault Tolerance** to handle host failures (future work).

### 3 The Design

In this section, we will discuss two approaches we have tried. Approach I experiments with Java RMI and extends the current Ptolemy infrastructure to mainly explore how to realize communication between distributed Ptolemy models. Approach II develops the discovery mechanism, launch service, monitor service and refines the remote communication mechanism.
D iscovery nicely and also facilities the discovery service.

We assume the DummyActor’s receiver works as a producer and the DistributedReceiver on the remote process works as a consumer. When ever there is token received on the DummyActor’s receiver, the token is forwarded to the DistributedReceiver via a remote method call. The DummyActor has a special receiver, called DelegatingReceiver for handling the remote method call. To enforce the PN semantics on this model, we implement blocking read and blocking write between participants of message send and message receive over the network. The blocking on the receiver is realistically reflected on the sender side, as if they were executed on the same machine.

One difficulty we have with this kind of point-to-point distributed communication is to handle the starting sequencing. Note that Host1 has some DelegatingReceiver needed to bind to some DistributedReceiver on Host2 and vice versa. To find a distributed object, it needs to be created and registered with the naming service first. So we need take care of when to register a DistributedReceiver and when to bind to a DistributedReceiver. We first assume the DistributedReceiver is created during the pre-initialization phase and delay the binding until a DelegatingReceiver receives the first token (the earliest time could happen is in initialize phase). This also requires synchronizing the demons to start the execution after all of them have done the pre-initialization phase.

These experiments help us understand the problems and requirements for a distributed execution environment. However, it has some flaws. The strict start sequencing requirement reduces the flexibility and it could contribute to a big part of distributed exceptions. In the next approach, we will discuss a more flexible architecture that breaks down the dependency nicely and also facilities the discovery service and monitoring service.

### 3.2 Approach II

Figure 2 illustrates the main components in the PtAgent, and how they interact with Ptolemy sub-models. We assume each machine will run exactly one PtAgent process. There are four main module in the PtAgent:

- **Launcher** is responsible for distributing, loading, starting, and stopping a sub-model.

- **Discovery** provides the functions for Ptolemy to discover available computers or resources.

- **Relay** provides service for Ptolemy to transfer data to the sub-models located on other machines.

#### Logging

- Logging records the models’ execution information and can be queried for debugging and monitoring purposes.

Each PtAgent may register itself to the DNS so that other PtAgents can find it. When a user wants to distribute a model (the master model), the DistributeHelper class will parse the model to sub-models and contact its local PtAgent (the only PtAgent running on the same machine) to distribute the model, which will in turn contact the related PtAgents to load and start the sub-models.

After a sub-model is started, it may need to transfer data to a remote sub-model. As shown in Figure 2, the DelegatingReceiver will contact an entity in Ptolemy called RelayProxy to send the data, which asks the PtAgent to relay the data to the corresponding remote PtAgent. Then the data is sent to the right port via the RelayProxy in the remote sub-model. This looks a bit complex, but there are several reasons to have it work this way. First, using PtAgents to relay data rather than sending the data directly to the remote receiver helps to break down the dependency loop as pointed in approach I. The idea is quite similar to CORBA using Event Channels to decouple the communication between a Producer and Consumer, except that here we distribute the Event Channels also, according to the configuration. Second, the RelayProxy is used to decouple the design of PtAgent and Ptolemy. Third, the RelayProxy can be domain specific to guard the execution semantics. For example, although it is possible to depend on some distributed middleware’s blocking mechanism to throttle a “fast” process in PN, it is quite expensive. The RelayProxy can then be used to block the process locally. Fourth, the agent may record the data sending via it for debugging and monitoring.
for its response, the PtAgent timely detects the malfunction of Ptolemy processes. Another example of such functions is authentication checking every time when a sub-model is being connected to.

5 Conclusion

We have discussed the high-level design of a distributed platform for Ptolemy II. This platform enables distributed model execution without requiring much experience of the user. The use of a global name system and a PtAgent for each machine greatly improves the flexibility of a dynamically deployed system.

We limit our current implementation to the PN domain, but we are also careful enough to make our platform easily extensible to other domains supported by Ptolemy. For distributed execution of timed domains, fault tolerance and backtracking techniques are required. These issues will be studied extensively in our future work.

References
