ViSiDiA: a Java Framework for Designing, Simulating and Visualizing Distributed Algorithms

Wahabou Abdou, Nesrine Ouled Abdallah, Mohamed Mosbah
Univ. Bordeaux, LaBRI, UMR 5800, 351 Cours de la Libération, 33 400 Talence, France
{wahabou.abdou, nesrine.ouled-abdallah, mohamed.mosbah}@labri.fr

Abstract—Simulation tools are useful for detailed analysis of distributed systems that are increasingly present in our daily lives. ViSiDiA (Visualization and Simulation of Distributed Algorithms) is a platform that aims both to facilitate teaching distributed algorithms and to contribute to the research activities. This paper presents design features and examples of how to implement new ViSiDiA's algorithms. These implementations can be done using the Java language or by using the ViSiDiA's GUI (just drawing the relabelling rules that correspond to considered algorithm). We also present a more complex problem for which ViSiDiA served as a simulation and analysis tool.

Index Terms—Simulation; Visualization; Distributed system

I. INTRODUCTION

The omnipresence of distributed systems (embedded systems, sensor networks, distributed databases, etc.) increases the interest of distributed algorithms in the design of applications. The specific features of distributed systems such as the lack of global information or the need of cooperation between entities in order to perform a global task make difficult the development of distributed applications. Studying the behaviour of these algorithms is difficult because it requires to continuously observe the state of each processor and the state of the communication links between processors. However, simulation and animation tools could facilitate this task.

ViSiDiA (Visualization and Simulation of Distributed Algorithms) is a framework that allows to implement, simulate and visualize distributed algorithms. It has a dual purpose: educational and research. ViSiDiA implements several types of distributed systems: synchronous or asynchronous ones, mobile agents and sensor networks. In addition, this framework allows users to implement their own algorithms using Java programming language. It is also possible to define a distributed algorithm without writing any source code through the graphical user interface (GUI) that enables to draw the rewriting rules that correspond to the algorithm.

The remaining of this paper is organized as follows: Section II presents the state of the art on frameworks that are used to simulate and/or visualize distributed algorithms. ViSiDiA's functionalities and the framework's architecture are introduced in Section III. Section IV presents an example of how to implement new algorithms. Finally, the Section V recalls one relevant result obtained thanks to ViSiDiA.

II. RELATED WORK

Some frameworks for simulating and/or visualizing distributed algorithms in the literature focus on aspects of high level abstraction including transition systems, while others study in more details the lower layers of the networks (communication aspects and modelling the communication channels). ViSiDiA belongs to the first category.

Distal [1] is a Scala framework for implementing fault-tolerant distributed algorithms. It proposes a simple language for writing algorithms. Indeed, Distal's architecture involves two parts: a domain specific language (DSL) and a messaging layer. The defined DSL aims to make easier the translation of pseudo-codes into executable codes. This DSL is based on the concept of state machines. The messaging layer deals with issues related to communications (particularly the connection management, the serialization of messages). These two parts are implemented in such a way that the messaging layer can be replaced with another communications system.

Like ViSiDiA, LYDIAN (Library of Distributed Algorithms and Animations) [2] focuses on educational aspects. LYDIAN has been designed in order to support students in understanding, developing and analysing distributed algorithms. It provides a TCL/TK graphical user interface endowed with an animator module to visualize the behaviour of the algorithms. The users can either create their own protocol to simulate (and eventually to visualize) or import an animation trace file for visualization. In case of simulation, a trace file is built prior to the animation step. LYDIAN allows to visualize some advanced informations such as causal relations and message complexity. FADA (Framework Animations of Distributed Algorithms) [3] is another framework strongly oriented towards educational goals. It uses a message passing model.

VADE (Visualization of Algorithms in Distributed Environments) [4] allows to simulate and visualize asynchronous distributed algorithms. It is based on an architecture in which the algorithms run on a server, while animations can be viewed on remote machines via a web page.

DAJ (Distributed Algorithms in Java) [5] allows students to interact with the states of processes, leading to better understanding of the state changes of the algorithms. Another toolkit of the same name [6] exists (the authors stopped referring to this name to avoid confusion). One of the main differences between these two platforms is that the second simulates distributed system by a multi-threaded application.
PeerSim [7] is widely used for simulating peer to peer (P2P) systems. It is an open source Java simulator which can be used for extremely large (million of nodes) and dynamic networks. PeerSim supports cycle-based (simplest) and event-based (more realistic) models. Other simulation tools such as NS2, NS3, OMNet++ can simulate distributed systems, for example ad hoc networks. But they particularly focus on lower layers of the OSI (Open Systems Interconnection) model. Since these simulators do not work at the same level of abstraction than ViSiDiA, they are not detailed in this paper.

Compared to the aforementioned frameworks, ViSiDiA allows to implement various standard distributed models, such as message passing, mobile agents, sensors and graph relabelling rules, using both synchronous and asynchronous systems. This ability relies on the theoretical studies based on the local computations model done in the ViSiDiA project (Bauderon and Mosbah introduced this work in [8]). In addition, ViSiDiA simulates fully distributed systems and enables to better investigate the interactions between processors. This investigation can be done in details using the debugging functionalities (due to the lack of space, the distributed debugging technique is not presented in this paper; please refer to [9]). Finally, ViSiDiA can be run either as a Java application or as a Java applet.

III. ViSiDiA: AN OVERVIEW OF THE FRAMEWORK

A. Main Functionalities

The ViSiDiA framework allows to implement, simulate and visualize distributed algorithms. It is based on the modelling of a network as a graph wherein each node represents a processor and the edges represent communication channels. Figure 1 presents ViSiDiA’s functionalities from users perspective. A user can implement a new algorithm (see Section IV), run an experiment or replay a simulation. To run an experiment, the user should first define a network topology either by creating it in ViSiDiA or by importing a GML (Graph Modelling Language) file. Thereafter he/she can select an algorithm among those provided in ViSiDiA’s database. Once an algorithm is selected, the simulation can be carried out. The user can visualize the progress of the simulation (through animations), but this is not mandatory. Instead of running a new experiment, a user can also replay a previously recorded simulation.

B. Architecture

ViSiDiA is implemented in Java and has been designed using a multi-layer architecture. This design choice allows to easily extend the functionalities of each layer independently of the other layers. For instance, this architecture facilitates the implementation of new algorithms. The flowchart of this multi-layer architecture is illustrated by Figure 2 (where plain arrows represent direct association, while the dashed arrow is an indirect association). This architecture relies on a Model-View-Controller (MVC) pattern where the model is the graph representing a distributed system, the view is the GUI, and the controller is the simulator module.

Fig. 2. ViSiDiA’s architecture

The GUI is the first layer. It enables to build the graph which represents the network topology, to visualize the progress of the simulation and to potentially influence this progress (speed up, slow down or pause the simulation).

The second layer includes the graph description tool and the distributed algorithms simulator (the main module of ViSiDiA). It is worth noting that this module is independent from the GUI. Indeed, a simulation can run without any GUI. ViSiDiA uses an event-driven simulator. Any action leads to the generation of an event. These events are interpreted by the simulator which (eventually) modifies the graph. For example, the process of communication can lead to several events: \((e_1)\) the source node issues a message, \((e_2)\) the GUI (if any) displays the message going from the source node to the destination(s), \((e_3)\) once the message reaches the destination, it is put into the receiver’s queue, \((e_4)\) the receiver performs some actions according to the content of the message. These events are transparently managed by the simulation module and algorithm developers do not have to deal with them.

The last layer contains an Application Programming Interface (API) that allows users to program their own distributed algorithms. This layer will be detailed in Section IV.

IV. DEVELOPING NEW ViSiDiA ALGORITHMS

A. Application Programming Interface

ViSiDiA offers simulation and visualization tools to run and to analyse some well-known algorithms such as leader election, spanning tree computation, ring colouring, graph enumeration, etc. The API allows users to implement their own algorithm. In order to write a new algorithm, one must choose the distributed model to use: message passing, graph relabelling rules, mobile agent or sensors. Figure 3 shows the diagram class of the algorithm layer.

The basic class that defines ViSiDiA’s algorithm is the Algorithm class. A copy of the algorithm is loaded on each
node by the simulator console before the simulation starts. Therefore, all the nodes have the same sequence of instructions to run. The changes of each node’s state depend on its own state and the states of its neighbours.

Algorithms should include the following two methods:

- clone(): which creates a new instance of the class;
- init(): which contains the sequence of instructions of the algorithm (the core of the algorithm).

Besides, the developers can add any auxiliary methods. Some useful methods are described in section IV-B.

The Algorithm class provides some methods to get or to modify the graph properties, particularly:

- for node properties:
  - getId(): returns the node’s Id.
  - getArity(): returns the degree of the node.
  - getProperty(key): gets the node’s property associated with the key (for example the label).
  - setProperty(key, value, [visibility]): sets a node’s property that may be displayed or not (according to the desired visibility option).

- for edge properties:
  - getEdgeProperty(door, key): returns the property value associated with the provided key of the edge identified by the door number (on the current vertex) or null if the key does not exist.
  - setEdgeProperty(door, key, value, [visibility])

For communication purposes, ViSiDiA provides simple functions for sending and receiving messages:

- sendTo(door, message): to send a message to a neighbour via a given door.
- sendAll(message): to send a message to all neighbours.
- receiveFrom(int door): gets the first message arriving on target door.
- receive(Door door): gets the first message arriving on the node (all doors are considered).

**B. Implementing a New Algorithm**

As an example of the implementation of algorithms, we focus the implementation of a broadcast algorithm, using a message passing model. We suppose that a given node (say v), labelled “A” has an information to broadcast. v sends the message to all its neighbours. Upon receiving the message, each node changes its own label to “A”, then forwards this message to its respective neighbours (except the one which previously sent this message). The implementation can rely on both synchronous and asynchronous systems. In this paper we only present the asynchronous version of this broadcast algorithm. This means there is no global clock and messages can arrive at arbitrary times. The asynchronous Broadcast algorithm extends ViSiDiA’s Algorithm class (see Listing 1).

```
public class Broadcast extends Algorithm {
    @Override
    public void init() {
        ...  
        @Override
        public void init() {
            return 
        }
    }
}
```

The getDescription() method is not mandatory. It is used to add a description of the algorithm that will be displayed in the GUI when the user will browse the list of the algorithm in order to select one of them.

The getMessageTypeList() is used by ViSiDiA to get the list of messages that will be displayed on the GUI (the user can choose to display them or not). In this example, there is only one message type, called wave.

```
Listing 1. Asynchronous broadcast algorithm

Listing 2. Asynchronous broadcast algorithm: the "init" method

```
The core of the broadcast algorithm is described in the init method (Listing 2). The instruction in line 3 allows to get the label of the node. If the node’s label equals to “A” (line 5), this means the node is the initiator of the broadcast (decision taken by the user), it sends to all its neighbours a message that contains the text “Hello” (line 6). If the label of the node is not “A”, then it waits for the first message coming on one of its doors (lines 8-9). Then the node changes its label (line 10) before forwarding the received message (lines 12-15).

C. Drawing Relabelling Rules

To make easier the design of distributed algorithms, ViSiDiA allows to implement algorithms without writing any Java code. These algorithms use the local computations model [8]. The implementation of such algorithms involves the use of a simple GUI. Figure 4 shows the spanning tree algorithm for any graph that contains exactly one initiator node labelled “A” (the other nodes are labelled “N”). If an “A” node is synchronized with a “N” node, the latter becomes “A” and the edge which connects these two nodes is marked (that means this edge is part of the spanning tree).

In the left part of Figure 4, the transition system that allows to relabel the graph is described. In the right part, some forbidden contexts can be added.

![Fig. 4. Relabelling Rule for Spanning Tree Computation](image)

V. AN EXAMPLE OF A DISTRIBUTED PROBLEM ADDRESSED BY VI SIDI A: SIMULATING POPULATION PROTOCOLS WITH LOCAL COMPUTATIONS

A Population Protocol (PP) [10] is a pairwise computation model for anonymous asynchronous passively mobile networks. The interacting neighbour nodes are chosen by a scheduler. In a communicating pair, one node represents the initiator and one the receiver. The two neighbour nodes exchange their states, do some computations according to the protocol transition rules, and then update them. ViSiDiA is an interesting environment to implement, simulate and visualize some stably computable PPs. The LC0 (Local Computations of class 0) synchronization algorithm [8] can play the role of the scheduler as it is able to choose only some links among all those of the communication graph. These chosen pairs will be synchronized to do the required computations according to the transition rules of the PP. Also, these transition rules can be translated to LC0 relabelling rules where the center node will represent the initiator and the second node will be the receiver (provided the symmetry is broken in the LC0 synchronization procedure). In [11], to broadcast an information in a mobile anonymous asynchronous network with no collision or duplication problems, the authors proposed a broadcast algorithm with PP. They proved that their algorithm was equivalent to the “OR” predicate which is stably computable by PP. This protocol was implemented in ViSiDiA using the LC0 model. Simulations with ViSiDiA using different types of communication graphs and different behaviors of the scheduler were made to validate the theoretical results concerning the time complexity average of the presented broadcast PP.

VI. CONCLUSION

In this paper, we introduced ViSiDiA, a visualization and simulation framework dedicated to distributed algorithms. ViSiDiA considers both synchronous and asynchronous systems and implements them using various standard distributed models. We detailed the implementation procedure of a broadcast algorithm in order to illustrate the ease with which new algorithms can be added to ViSiDiA. Moreover algorithms can be described without writing any source code. Finally this paper presented one more difficult problem that have been addressed using ViSiDiA. As future work, we plan to extend ViSiDiA’s models to dynamic graphs. One possible approach would be to build a model based on evolving graphs.

ACKNOWLEDGMENT

This study has been carried out with financial support from the French State, managed by the French National Research Agency (ANR) in the frame of the “Investments for the future” Programme IdEx Bordeaux - CPU (ANR-10-IDEX-03-02)

REFERENCES