Individual Systems Practical Proposal

Hans-Dieter A. Hiep

November 3, 2016

Abstract

A project proposal for the Individual Systems Practical (ISP) course to design and develop a simulation and visualization system for distributed algorithms.

1 Introduction

Distributed Algorithms is a master course taught by Wan Fokkink at the Vrije Universiteit. From the study guide: the objective of that course is to provide students with a frame of mind for solving fundamental problems in distributed computing. Students are offered a bird's-eye view on a wide range of algorithms. The course has an intuitive approach and shows during lectures many examples of distributed algorithms.

The author of this proposal found that a supplement to the course, being a systematic way of visualizing the examples by means of a computer program, might help students to study algorithms. This proposal, thus, amounts to a project to design and develop a system that correctly simulates a broad range of taught algorithms and visualizes them in a straight-forward and interactive manner, to promote experimentation.

A project summary and planning is given in section 2. A description of the project deliverables is given in section 3. More details concerning the system and the preliminary design choices is given in section 4. Finally, related work is given in section 5.

2 Overview

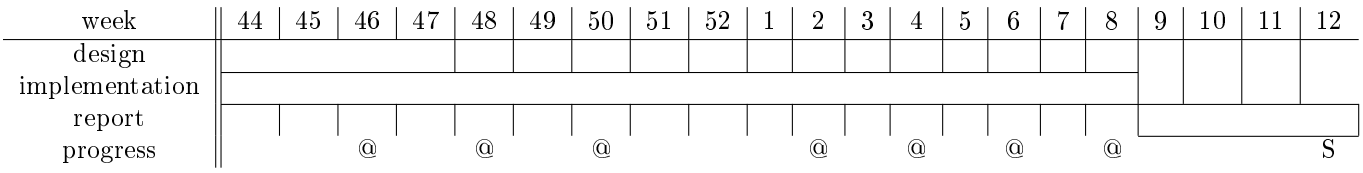
Supervisor	Wan Fokkink
Department	Computer Science
Section	Theoretical Computer Science
Student	Hans-Dieter Hiep
Student no.	#2526195
ECTS	6
Hours	$6 \times 28 = 168$

The project consists of the design and development of a system for simulating and visualizing distributed algorithms. The goal of the project is to let students explore computations of distributed algorithms by means of an interactive desktop applications. A precise and concise description of the system follows.

Students can load a network description and choose from a set of predetermined suitable algorithms that will execute on the simulated network. Networks are visually represented by an interactive graph, in which nodes (processes) are connected by edges (channels). A timeline shows an interactive view of a particular execution of a selected algorithm, and students can adapt the execution by reordering events. The change is reflected in the timeline by recomputing the consequences of such a reordering on the execution of the algorithm. Furthermore, students can step through the timeline and view the changes reflected in the network, similar in nature to a debugger for programming languages. At each point in time, the network visualization updates to reflect the current state of the network. Students can select individual processes to view their internal state, and channels to view messages in transit. For more advanced concepts, for example, processes can be marked crashed by the student to explore the consequences on the execution of the algorithm, and, for probabilistic algorithms, students can influence the source of randomness.

In summary, the project aims to supplement the intuitive approach set out by the course, by letting students interactively explore distributed algorithms that allows for experimentation and in-depth study of the algorithms.

The project plan is as follows: over the course of 21 weeks, at 8 hours per week, the proposer will work on the project. During the design and implementation phase, a biweekly report of the progress of the project is sent to the supervisor by e-mail (@). At the end of the project, the resulting software deliverable is demonstrated at a seminar (S) where (at least) the supervisor is present.



3 Deliverables

The most important deliverable of the project is a software deliverable. Secondary is a technical report that serves as end-user and system maintainer documentation, describing in detail the functionality of each aspect of the system and ways of extending the system in the future.

The software deliverable comprises a full source code repository, including the scripts for building and packaging, and a deployable software package. Additional files required for the operation of the system are included as well, including example network descriptions.

The technical report comprises not more than ten pages of documentation, describing in detail the functionality and architecture of the delivered system, in a similar format as this proposal. The contents of this report will be discussed during the demonstration at the end of the project.

The proposal files, all source code files, required files and documentation files are authored by the proposer only. However, design decisions may be made in collaboration with the supervisor or with other students.

4 System Design Parameters

The system consists of two pillars: a simulation framework and a visualization framework. The simulation framework is embedded into the visualization framework to produce the final software deliverable, and the frameworks are developed in lockstep. The overarching technology is the Java platform, that enables wide deployment of the software on different platforms that student end-users might use.

The simulation framework is to be implemented using the Haskell programming language. For the simulation framework, this project will make use of the Frege project, see the http://github.com/frege website: "Frege is a Haskell for the JVM. It brings purely functional programing to the Java platform." All algorithm simulation implementations are written in Haskell accordingly. Due to slight incompatibilities between Frege and, for instance, GHC, this implementation is not immediately usable in other Haskell implementations, but can be adapted with only minor tweaks.

For the visualization framework, this project will make use of the standard Java implementation for graphical user interfaces: Swing. Since the Swing toolkit is usable among a wide range of versions of the Java platform, and since the toolkit is highly extensible, the visualization framework will consist of custom components for visualizing networks (processes, channels and annotations), process state & messages, and timeline events. The framework will reuse other common components of Swing. Futhermore, the framework will be developed according to the standard Model-View-Controller (MVC) principles of Swing, and, furthermore, according to the principles of Pluggable Look-and-Feel (PLAF). Potentially, (parts of) this framework can be reused or adapted for other visualization projects.

The glue between the visualization framework and the simulation framework will consist of routines for launching the desktop application, interpreting Frege structures as Java models for use in the visualization framework, and interfacing between user interface command structures and Frege (re)evaluations. This glue code is highly project specific and will be kept minimal in size and complexity.

5 Related Work

Related work is the visualization software provided with the book "An Introduction to Formal Languages and Automata", 5th Edition by Peter Linz. It includes a CD-ROM that contains software to experiment with formal languages, called JFLAP. From the www.jflap.org website: "JFLAP is software for experimenting with formal languages topics including non-deterministic finite automata, non-deterministic push-down automata, multi-tape Turing machines, several types of grammars, parsing, and L-systems. In addition to constructing and testing examples for these, JFLAP allows one to experiment with construction proofs from one form to another, such as converting an NFA to a DFA to a minimal state DFA to a regular expression or regular grammar." The author of this proposal found that software immensely useful for the bachelor course Automata & Complexity. This work is similar in its aim to help students study the subject and promote experimentation. Additionally, the work is similar in deployment and also features graph visualizations and interactivity.

Other related work is PseuCo, a web application for interactively exploring transition systems to teach concurrent programming. From the www.pseuco.com website: "This application has initially been developed for the Concurrent Programming lecture, but is now targeted towards everyone teaching or learning concurrent programming. It can parse pseuCo and CCS, includes a compiler from pseuCo to CCS and CCS-to-LTS semantics, and can visualize and minimize the Labeled Transitions System, featuring automatic graph layout." This work is similar in its aim to help students study the subject, and also features graph visualizations. Additionally, this work is similar in its design by using a formal model, Calculus of Communicating Systems (CCS). It goes further by letting students program systems, which is more involved than what this project aims for.