



**PEPITO**  
**IST-2001-33234**  
**PEer-to-Peer Implementation and TheOry**

**Deliverable no: D1.8**

## **Second Progress Report on Formal Models**

REPORT VERSION: first

REPORT PREPARATION DATE: 2003.12.31

CLASSIFICATION: Public

DELIVERABLE NO: D1.8      DUE DATE: Month 24      DELIVERY DATE: Month 24

PROJECT START DATE: 2002.01.01      PROJECT DURATION: 36 months

RESPONSIBLE PARTNER: UCAM

PARTICIPATING PARTNERS: EPFL, INRIA, KTH, UCAM, UCL

PROJECT COORDINATOR: Swedish Institute of Computer Science AB

PROJECT PARTNERS: EPFL Lausanne, INRIA Paris, KTH Stockholm, UCL Louvain, University of Cambridge UK



**Project funded by the European Community under the 'Information Society Technologies' Programme (1998–2002)**

Project Number: IST-2001-33234  
Project Acronym: PEPITO  
Title: PEer-to-Peer Implementation and TheOry  
Deliverable No: D1.8  
Second Progress Report on Formal Models  
Due date: project month 24  
Delivery date: 2003.12.31

Responsible Partner: UCAM  
Participating Partners: EPFL, INRIA, KTH, UCAM, UCL

25th January 2004

Editor: Peter Sewell

Authors: Johannes Borgström, Vincent Cremet, Uwe Nestmann, James Leifer, Luc Onana Alima, Andrei Serjantov, Peter Sewell, Peter Van Roy, and Keith Wansbrough.

## Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Changes to the Technical Annex . . . . .	3
1.2	Summary and Relations to Other Workpackages . . . . .	3
1.3	Plans for 2004 . . . . .	4
<b>2</b>	<b>Tasks</b>	<b>4</b>
2.1	Transactions and the Semantics of Failure . . . . .	5
2.1.1	Low-level Interaction and Failure Semantics: Sockets, UDP, and TCP . . . . .	5
2.1.2	Failure Detectors . . . . .	6
2.1.3	Transaction Axiomatization . . . . .	7
2.1.4	GlobalStore . . . . .	7
2.2	Models of Peer-to-Peer Group Collaboration . . . . .	8
2.2.1	Implementation of Chord in Acute, above the TCP Semantics . . . . .	8
2.2.2	DKS Formalization . . . . .	8
2.2.3	Models for Anonymity Analysis . . . . .	8
2.2.4	Confinement control . . . . .	9
<b>3</b>	<b>Attached papers</b>	<b>10</b>

# 1 Introduction

## 1.1 Changes to the Technical Annex

The Technical Annex contains two tasks in this workpackage:

- ▷ Transactions and the Semantics of Failure [EPFL:10, UCAM:22, UCL:12]
- ▷ Models of Peer-to-Peer Group Collaboration [EPFL:6, INRIA:6, UCAM:6, UCL:6]

The original technical annex contained a third task in WP1,

- ▷ Versioning and Modularity [UCAM:24]

but in the revised Technical Annex (of 9 July, 2003) this has been merged into WP3, forming a combined task *Resource Control, Versioning, and Modularity*; we therefore report on it in detail in the WP3 section of the PPR, but also summarise it below to make this document self-contained.

## 1.2 Summary and Relations to Other Workpackages

### 1. Resource Control, Versioning, and Modularity (now in WP3; this is a partial summary).

- (a) We have developed a programming language, *Acute*, with support for type- and abstraction-safe marshalling and rebinding to local resources. The language has a full formal semantics and a prototype implementation, and a TCP sockets library matching that of our low-level failure semantics. We can thereby have completely semantic models of networks of executable programs, with distributed algorithms (eg Chord or DKS) expressed in *Acute*, and their semantics given by the combination of the *Acute* and low-level network semantics. This renders them amenable in principle to complete formal proof, and the semantics may also be used to support informal reasoning.

### 2. Transactions and the Semantics of Failure

- (a) Much progress has been made on low-level interaction and failure semantics, with a draft specification of TCP/UDP sockets completed and released internally to the project partners.
- (b) An experiment in fully-formal use of this specification has begun, using the Isabelle theorem prover.
- (c) A new failure detector model has been proposed, that we consider easier to understand, easier to work with, and more natural than existing models (ASIAN 2003 publication).
- (d) Work on transaction axiomatization has been completed (FSTTCS03 publication), and work on formalization of our GlobalStore abstraction is underway.

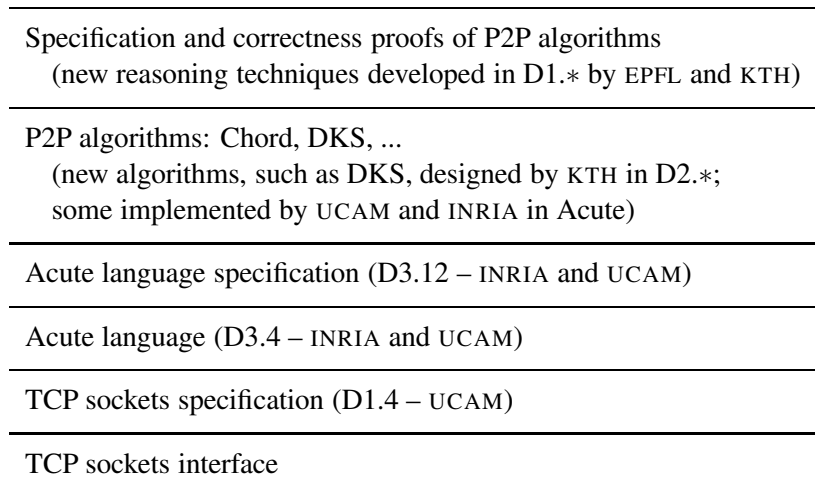
### 3. Models of Peer-to-Peer Group Collaboration

- (a) Work is in progress on an implementation of the Chord P2P algorithm in *Acute*, drawing together the WP1 low-level TCP semantics, the WP3 work on language design, and an existing P2P algorithm.
- (b) Work is in progress on a formalization of the DKS P2P algorithm of WP2. This will need failure detector models, and we also intend to make an executable model (with a tight relationship to the verification) above the UDP/TCP/Socket semantics, building on our work on Chord.

- (c) We have developed special-purpose models for analysis of the anonymity provided by low-latency connection-based anonymity systems, among other points discussing quantitatively the extent to which they should be fully P2P (ESORICS2003 publication).
- (d) Initial work on models for confinement control has begun.

Some of the key relationships between these tasks and the other PEPITO workpackages are summarised in the “layer cake” diagram below, which is presented with more detail of our vision in the PPR2.

*A “layer cake” interleaving specification and implementation*



The diagram shows the interleaving layers of implementation and specification work — rising from the lowest level, namely Unix socket-based TCP communication, to the highest, namely the behavior of P2P algorithms.

### 1.3 Plans for 2004

No changes are required or envisaged at this point. We expect to continue work on low-level failure semantics, completing the TCP/UDP specification; on model implementations of P2P algorithms in *Acute* and on formalization and reasoning about P2P algorithms, completing both of these; and on models for confinement control.

By the end of the project, drawing together work from WP1, WP2, and WP3, we expect to have exemplar *fully specified systems*, with rigorous semantic foundations for the underlying TCP/UDP/Sockets networking, for the *Acute* programming language with marshalling support, and for the P2P algorithms written therein.

## 2 Tasks

We now discuss each WP1 task in turn, concluding by listing the associated papers published during this year.

## 2.1 Transactions and the Semantics of Failure

### 2.1.1 Low-level Interaction and Failure Semantics: Sockets, UDP, and TCP

Here we aim to develop rigorous semantic models of the low-level interactions and failures that may occur in a peer-to-peer system, not in the rather abstract styles of traditional distributed algorithm theory or process calculi, but for the actual networking APIs used.

This entails development of a formal model of the behaviour of UDP, TCP, and the Sockets interface, covering the behaviour of real implementations, and validated against them (in particular, against BSD, Linux, and WXP behaviour) to ensure the specification includes all the real-world behaviour. This is a very substantial task; the results should be of wide interest for those building Global Computing systems, enabling them to see clearly the failure semantics and differences between implementations, and hence to build more robust systems.

Work has continued on several fronts: the specification, the test harness, the tests themselves, the symbolic evaluator, the validation harness, publication, and, finally, initial usage of the specifications. The current draft specification [4] is being distributed internally within the project, and also made available to reviewers. The progress since the state reported in Deliverable 1.7 (First Progress Report on Formal Models) is as follows.

**Specification.** We continue to: fill in gaps in the specification, clarify what is there, refine nondeterministic behaviours into specific details where appropriate, correct errors discovered by code reading and by experiment, and simplify and clarify the structure of the specification for the benefit both of readers and of the symbolic evaluation engine. We are also integrating our earlier UDP semantics with the current TCP and socket semantics in HOL.

**Test harness.** While the bulk of this was completed in the previous reporting period (D1.7), once we began to use it ‘in anger’ the need for various improvements and extensions became clear. Furthermore, in order to support semi-automatic generation of tests, a high-level layer supporting construction of sockets-based programs had to be built over the existing low-level system-call and injection API. One significant component of this was the library that synthesises appropriate (and, for certain tests, inappropriate!) response datagrams to those generated by the host under test – essentially a cut-down TCP stack. A final, unexciting but necessary, component was the tracking of various syntactic changes in the specification as that work progressed. The test harness work was almost entirely undertaken by our pre-doctoral Research Assistant, Steve Bishop.

**Tests.** The purpose of the test harness is to support the generation of large numbers of tests – traces of the behaviour of actual TCP/IP stacks in various situations. The aim here is *coverage* – all rules in the specification must be covered for each architecture, and as many different conditions as possible must be established and tested. This is something of an art, since total coverage strictly interpreted is impossible (the model has an infinite number of states), and even loosely interpreted is practically impossible (consider: the TCP control block contained in each socket has 44 fields; even were each only a single bit there would be  $2^{44} \simeq 10^{13}$  states, and the host state contains multiple sockets and other information besides). Test generation is therefore performed white-box, with some knowledge of TCP itself, the way in which it is implemented on various architectures, and the structure of the specification. This is not ideal, but it is forced. However, the inclusion of both implementation and specification knowledge in the test design process means that the *explicit* assumptions of each, at least, are certainly tested. Test design in practice means writing an OCaml program above the high-level library already mentioned, which will drive the test harness to perform the desired sequences operations and generate traces. Usually such a program contains a number of nested loops iterating over different states of the test parameters; this permits exponential multiplication of effort. We currently generate 3007 traces, from 238 test scripts covering around 24 sorts of rule. This work was also undertaken by Steve Bishop, and has been

a significant effort this year.

**Symbolic evaluator.** The work begun last year on adding symbolic evaluation to HOL, in order to permit validating the model against the traces, has continued. Our specification is the largest attempted in HOL, and so various elements of the HOL tools have needed to be extended – notably the datatype handling, which had quadratic behaviour in the number of record fields (unacceptable for the TCP control block with 44 fields!). The symbolic evaluator has been updated to accommodate the idioms used in the specification, and in particular to control when evaluation occurs (blind evaluation leads to much wasted work, for example on the not-taken branch of an **if** expression). This work benefits the rest of the HOL community, as well as ourselves. It is being performed part-time by Michael Norrish, in Canberra, with only a small use of Pepito resources (machine-time).

**Validation harness.** Operating on the scale of 3007 traces, applying the symbolic evaluator by interactive use of the HOL tool is not a practical means of validation. Instead, a combination of HOL scripting (in HOL's native Moscow ML), shell, and Perl scripting performs automated checking of each trace, along with progress and result reporting. Both summaries and detailed results are reported, and some HTML/CSS/ECMAScript coding enables easy navigation of the often-large success/failure reports (median 5000 lines). Further, at this scale, and with a single trace taking several minutes to check, parallelisation becomes necessary, and so we have developed a system that distributes the work across multiple machines. In this way we are able to check all traces in a three- to five-day window, quite acceptable given that both the trace set and the evolving specification change only slowly. (We could easily improve this time merely by adding more machines, and are in negotiation with our site technical staff to leverage unused time on the laboratory's workstations; using other Pepito sites' machine time would also be an option. The loosely-coupled problem is extremely amenable to distribution, although there are significant memory requirements which limit the use of workstations in active use at the time.) We have found the reporting component of the validation harness to be indispensable for tracking progress and identifying and prioritising the work required to be done next, both on the specification and the symbolic evaluator.

**Publication.** We intend to publish our TCP specification as a book or monograph. In order to achieve this, some small improvements to the typesetting system will be required, but primarily it is a case of writing the explanatory text that goes with the formal document. In addition, some significant restructuring of the specification is still necessary, along with an ongoing process of 'cleaning up', removing overspecificities and implementation details due to its BSD and C heritage, and isolating and clarifying the essential structures and algorithms of TCP and the sockets layer. This is a fair amount of work.

**Using the Spec** As an experiment in fully-formal use of the specification, PhD student Michael Compton has taken the UDP fragment, ported it to the Isabelle theorem prover, written an Isabelle semantics for a fragment of OCaml extended with the UDP socket calls, and proved within Isabelle some sanity properties and properties of a simple example program using the socket layer.

In Section 2.2.1 below we describe ongoing work on an implementation of the Chord P2P algorithm above the TCP semantics.

## 2.1.2 Failure Detectors

The concept of *unreliable failure detectors for reliable distributed systems* was introduced by Chandra and Toueg as a fine-grained means to add weak forms of synchrony into asynchronous distributed systems. Various kinds of such failure detectors have been identified as each being the weakest to solve some specific distributed programming problem. In [2], we provide a fresh look at failure detectors from the point of view of program-

ming languages, more precisely using the formal tool of operational semantics. Inspired by this, we propose a new failure detector model that we consider easier to understand, easier to work with and more natural. Using operational semantics, we prove formally that representations of failure detectors in the new model are equivalent to their original representations within the model used by Chandra and Toueg. In summary, we provide an original presentation, using operational semantics, of existing work by Chandra and Toueg, which is targeted at an audience in process calculi and programming language semantics. As main contribution, the new model to represent failure detectors eliminates a number of drawbacks of the original model used by Chandra and Toueg. Many other failure detectors have been studied in the literature; up to now, we restricted our attention to the ones introduced by Chandra and Toueg and by Chandra, Hadzilacos and Toueg.

We expect to be able to profit from the experience with the new failure detector description model once we approach more dynamic versions of DKS in which failure detection is required (see Section 2.2.2 below).

### 2.1.3 Transaction Axiomatization

This year we completed our work on the axiomatization of transactions. The deliverable D1.3, available since June 2003, is our final report for the project. This work also led to a publication in the proceedings of a recognized conference [1].

The workpackage including this research aims at defining formal models for aspects of distributed computation which are central for peer-to-peer systems. Transactions fall into this category and can help to design robust and fault-tolerant distributed programs.

Transactions are commonly described as being ACID: All-or-nothing, Consistent, Isolated and Durable. However, although these words convey a powerful intuition, the ACID properties have never been given a precise semantics in a way that disentangles each property from the others. Among the benefits of such a semantics would be the ability to trade-off the value of a property against the cost of its implementation.

Our contribution is to give a sound equational semantics for the transaction properties. We define three categories of actions, A-actions, I-actions and D-actions, while we view Consistency as an induction rule that enables us to derive system-wide consistency from local consistency. The three kinds of action can be nested, leading to different forms of transactions, each with a well-defined semantics. Conventional transactions are then simply obtained as ADI-actions.

From the equational semantics we develop a formal proof principle for transactional programs, from which we derive the induction rule for Consistency.

We hope that this theoretical work on the decomposition of transactions can be applied to the design and implementation of simple and well-understood language constructs for transactional computation in a peer-to-peer context. Contrary to the domain of databases where a transaction is always supposed to satisfy the four ACID properties, using transactions at the level of applications can lead to choose only a subset of these properties to match specific requirements, for instance in terms of efficiency.

### 2.1.4 GlobalStore

A UCL student has formalized the working of our GlobalStore abstraction (a transactional store with active fault tolerance). Further work is required on scalability to make this directly usable in peer-to-peer computation.

## 2.2 Models of Peer-to-Peer Group Collaboration

### 2.2.1 Implementation of Chord in *Acute*, above the TCP Semantics

As we describe in the progress report for WP3, in task *Resource Control, Modularity, and Versioning* we have developed a programming language, *Acute*, with support for type- and abstraction-safe marshalling and rebinding to local resources. The language has a full formal semantics and a prototype implementation. It also has various libraries, including a TCP sockets library matching that of our low-level failure semantics. The behaviour of networks of interacting *Acute* programs is therefore completely semantically specified, rendering them amenable in principle to complete formal proof. The problem of scale may rule out proofs about large examples – it remains to be seen – but nonetheless this is a major achievement, and should contribute to robust software development by providing precise documentation for the entire infrastructure (language and network).

For an example, work is in progress on an implementation of the Chord P2P algorithm in *Acute*, drawing together the low-level TCP semantics, the work on language design, and a P2P algorithm (albeit an existing algorithm, rather than one developed in PEPITO).

### 2.2.2 DKS Formalization

As a first step in the formal verification of peer-to-peer protocols, we studied the DKS lookup algorithm in a static setting, i.e., where no processes may join or leave the system. We expressed the specification of lookup as well as appropriate portions of the DKS lookup algorithm in a value-passing process calculus, and related the specification and the implementation by weak bisimulation. Interestingly, even though weak bisimilarity is a priori not sensitive to the number of internal messages (or even divergent behavior), the performance-driven design of the algorithm required us to prove performance properties (maximum number of messages per lookup) in order to obtain the bisimulation.

We now plan to proceed towards a more dynamic setting, where nodes can concurrently join and leave the system, even in concurrency with lookups. Compared to the static case, we will have to treat the local state of the nodes (routing tables, membership status) in a more refined way. Another important issue, common to formal methods for distributed systems, is how to model timeout-based failure detection. Moreover, in some dynamic systems, including DKS, a node that leaves the system according to protocol may still have dangling pointers associated to it, something that is also detected using timeouts. For this reason we will have to treat failure detection even in a model without host failures.

We hope also to make an executable model of the algorithm (in particular, of the variant of the algorithm of the formal model) in a fully semantically-specified setting, ie in *Acute* above the UDP/TCP/Socket semantics, building on our work on Chord described above.

### 2.2.3 Models for Anonymity Analysis

In this work [3] we developed special-purpose models for analysis of the anonymity provided by low-latency connection-based anonymity systems, which can be used for applications like web browsing or SSH. Although several such systems have been designed and built, their anonymity has so far not been adequately evaluated.

We analyse the anonymity of connection-based systems against passive adversaries. We give a precise description of two attacks, evaluate their effectiveness, and calculate the amount of traffic necessary to provide a

minimum degree of protection against them.

Both fully P2P and hybrid (partially centralized) system architectures can be proposed: we demonstrate quantitatively that for this application the hybrid architectures are in fact preferable – intuitively, good anonymity requires sufficient traffic flowing between nodes, and a fully P2P design spreads the traffic too thinly; one must identify a subset of nodes as forwarders.

#### **2.2.4 Confinement control**

Preliminary work has started at UCL on formal models for confinement control, inspired by Schmitt and Stefani's work on the M calculus and Kell calculus.

### 3 Attached papers

- [1] A. P. Black, V. Cremet, R. Guerraoui, and M. Odersky. An equational theory for transactions. In *Proceedings of FSTTCS03: 23rd Conference on Foundations of Software Technology and Theoretical Computer Science, Mumbai (Bombay), India*, December 2003. Full version available as EPFL technical report IC/2003/26, 2003. <http://lamp.epfl.ch/~cremet/publications/fsttcs03.ps>.
- [2] U. Nestmann and R. Fuzzati. Unreliable failure detectors via operational semantics. In V. A. Saraswat, editor, *Proceedings of ASIAN 2003, LNCS 2896*, pages 54–71, Dec. 2003. <http://lampwww.epfl.ch/~uwe/doc/nestmann.fuzzati-asian03.pdf>.
- [3] A. Serjantov and P. Sewell. Passive attack analysis for connection-based anonymity systems. In *Proc. ESORICS 2003, LNCS 2808*, Oct. 2003. [http://www.cl.cam.ac.uk/users/aas23/papers\\_aas/conn\\_sys.ps](http://www.cl.cam.ac.uk/users/aas23/papers_aas/conn_sys.ps).
- [4] K. Wansbrough, P. Sewell, M. Norrish, S. Bishop, and M. Fairbairn. An experimentally-validated semantics for TCP, UDP, and the Sockets API. Working draft, distributed within PEPITO, 2003. The semantics comprises the following documents:
  - spec-intro.ps
  - TCP1\_LIBinterface-doc.ps
  - TCP1\_auxFns-doc.ps
  - TCP1\_baseTypes-doc.ps
  - ||||| del-1.8-bib.bib TCP1\_errors-doc.ps
  - ===== TCP1\_errors-doc.ps
  - |||||| 1.2 TCP1\_host0-doc.ps
  - TCP1\_hostLTS-doc.ps
  - TCP1\_hostTypes-doc.ps
  - TCP1\_netTypes-doc.ps
  - TCP1\_timers-doc.ps
  - TCP1\_utils-doc.ps