Teaching Coloured Petri Nets — a Gentle Introduction to Formal Methods in a Distributed Systems Course

Søren Christensen and Kjeld H. Mortensen

University of Aarhus, Computer Science Department,
Ny Munkegade, Bldg. 540, DK-8000 Aarhus C, Denmark
{schristensen,khm}@daimi.aau.dk

Abstract. This paper is about the two compulsory project assignments set to the students in an undergraduate course on distributed systems. In the first assignment the students design and validate a non-trivial layered protocol by means of Coloured Petri Nets, and in the second they implement the designed protocol in an object-oriented language. From the two assignments the students experience that Coloured Petri Nets, as a formal method, are useful for designing and analysing distributed systems. In the course students are introduced to basic concepts and techniques for distributed systems, and it is explained that such systems are often too complex to manage without using formal methods. In this paper we also report on our experience with teaching the course and describe the didactic methods applied. Based on the obtained experience we conclude that the combination of distributed systems and Coloured Petri Nets is fruitful — the two areas complement each other. Although our experiences origin in Coloured Petri Nets, we believe that many of our observations hold for other formal methods as well.

Topics. Educational issues related to nets; Coloured Petri Nets; distributed systems; experience with using nets, case studies; applications of nets to protocols.

1 Introduction

At the Computer Science Department (DAIMI), University of Aarhus, we have provided a one-semester course on distributed systems for the past four years. As part of the course the students are required to undertake two comprehensive project assignments. In the first assignment the students design and validate a layered communication protocol in a distributed system by means of Coloured Petri Nets (henceforth abbreviated as CP-nets or CPN) [9], and in the second they implement their design in an object-oriented language. In the course the roles of the two authors were to be a lecturer and a teaching assistant, respectively. There authors were also responsible for the planning and execution of the two project assignments.

In this paper we use the following concepts in relation with the course: A *teaching assistant* is a person who assists the lecturer in carrying out the course, e.g., by carrying out a project assignment. A *tutorial* is a class for students in groups of size typically up to 20 people where they work with the material and small exercises based on the recent lectures. The tutorials are supervised by *tutors* who often are PhD students. Teaching assistants also act as tutors.

The course is partly based on the second half of a textbook on operating systems by Tanenbaum [17]. His definition of a distributed system is as follows:

> "A distributed system is one that runs on a collection of machines that do not have shared memory, yet looks to its users like a single computer."

A collection of machines implies a concurrent system, the lack of shared memory suggests communication, and communication implies protocols. CP-nets is a well-suited technique for designing, validating, and verifying concurrent systems and communication protocols. About five years ago the curriculum for undergraduate studies at DAIMI did not have a special course to cover the topic of distributed systems. The department has a long tradition in research on high-level nets (Coloured Petri Nets) in conjunction with the development of a supporting tool (Design/CPN [11]). Therefore it was natural to start offering a computer science course with the combination of distributed systems and CP-nets.

The primary goal of the course is to introduce basic concepts and techniques for distributed systems. The concepts and techniques are illustrated by the distributed systems that the students use every day such as, e.g., file systems, printers, and electronic mail — concepts they already know from using the workstations at DAIMI. In the course the students are exposed to formal methods as a practically applicable tool for designing and analysing distributed systems. This is in contrast to the students' background in formal methods which is mostly on a theoretical level. CP-nets are used to illustrate and introduce concepts encountered in Tanenbaum's book in a gradual fashion.

This paper reports on experiences in teaching the course and the lessons we learned concerning the use of the formalism (CP-nets) and the tools — a digest of the past four years of experience.

The synopsis of the paper is as follows: In Sect. 2 we outline the course and describe a selection of activities in the course, and in Sect. 3 we describe the two compulsory project assignments. Sections 4 and 5 contain thoughts on future improvements and concluding remarks, respectively.

2 Course Description

The course on distributed systems is offered one semester each year, where there are 60–80 students signing up. It is mandatory for students taking computer science as their major subject. Students are typically on the level of the 6th semester. In order to take the course the students are required to have fundamental computer science skills. These skills cover algorithms and data structures, computer architecture, fundamental theoretical models, and basic concepts of computer programming languages and language design.

During the course 25 lab computers are available to the students. They are Sun Sparc workstations each equipped with 64Mb RAM and connected in a network.

2.1 Contents of the Course

In the course fundamental topics on distributed systems are treated: system architecture (e.g., the client/server model), networks and protocols (e.g., LAN and remote procedure calls), management of shared resources (e.g., by means of semaphores and monitors), and design and implementation principles for distributed operating systems (exemplified with Amoeba and Mach). All these topics are covered by a book on modern operating systems by Tanenbaum [17] (Chapts. 9–15).

Tanenbaum's book is a sufficient introduction to distributed systems for undergraduate students as it treats most of the central topics. Unfortunately, the level of the exercises in the book does not meet the requirements of the course. Most of the exercises do not sufficiently make demands on the creativity of the students. Therefore Tanenbaum's book must be supplemented with more in-depth exercises in order to make the tutorials more interesting for the students. In general, this is a problem with most of the other textbooks we have evaluated for the course.

Another significant part of the course is to illustrate that reasoning about distributed systems is far more complex than reasoning about traditional sequential systems. This is again illustrated by situations that the students already know from the systems they work with such as unexpected live-locks due to shared resources and apparently unnecessary waiting due to the crash of some remote computer. Hierarchical CP-nets are introduced as a modelling and validation technique suitable for distributed systems. The book used here is the introduction to CP-nets by Jensen [9], though without detailed studying of the most difficult parts of the formalism.

Note that we use CP-nets as the starting point of teaching Petri Nets. Although we mention other Petri Net formalisms such as PT-nets, we deliberately spend the time chiefly on CP-nets. Our experience is that it is possible to teach CP-nets right from the beginning in a course. We feel this is important when stressing the pragmatics of a formal method such as CP-nets.

Compared with other graphical-oriented formal methods, such as SDL and Harel's Statecharts, CP-nets have a number of didactic advantages. CP-nets, and Petri Nets in general, are simple languages with only a few elementary constructs, yet expressive. CP-nets, and high-level nets in general, offer, in contrast to many other formal methods, a simple pragmatic framework for describing, validating, and verifying complex concurrent systems. Although the students of our course are not exposed in detail to other formal methods, we wish to teach the students a number of essential and general concepts, such that the students can think about distributed systems by means of the techniques of formal methods in general.

The books of Tanenbaum and Jensen, the material on which the course is based, are sufficiently comprehensive to support a one-semester course. One weakness of the syllabus, however, is that the two books do not share related topics. Tanenbaum does not write about CP-nets and Jensen does not treat distributed systems. Thus there is a risk that the course may become two independent parts. In the course we ensure that the two parts are strongly interconnected by means of the compulsory project assignments, where the students must apply concepts and techniques from both parts. Furthermore, during the lectures concepts from distributed systems are explained by means of CP-nets and vice versa.

2.2 Format of the Course

The syllabus is taught by means of three 45-minutes lectures each week, and tutorials of three hours per week. The course is assumed to occupy one third of the students' time for 14 weeks.

At the beginning of the semester, there are tool demonstrations and hands-on lab sessions. Demonstrations during lectures deal with the techniques for constructing CPN models and an overview of tool features. A very simple example protocol is used as a running story. Usually, lectures provide only a quick overview so, in addition, there is a need for more intensive tool experience. This is accomplished during one of the tutorials where the students undertake some of their weekly exercises at the lab computer, and the tutor is available for immediate assistance. Such a hands-on session is a quick means to get the students started with the tools used in the course. They get immediate feedback and do not waste time on trivial technical problems.

In general, the lectures act mainly as one-way communication where the goal is to provide an overview of the syllabus, while the tutorials are a forum for intensive treatment of the weekly exercises and questions on the course material. Both lectures and tutorials are available as an opportunity for the students to work with the course material on multiple levels of detail. Experience indicates that the tutorials are very important for learning the course material. At home the students study the exercises independently. At the tutorials the students often work in small groups in order to solve the exercises, and then several solution proposals are presented and discussed. This demands responsibility and encourages involvement from the students. The tutor acts as a supervisor and a catalyst for the discussions and is therefore also the person who has a general overview of how well the students grasp the syllabus. There is a weekly meeting between the tutors and the lecturer with the purpose of continuously coordinating and evaluating the course.

Course information is made accessible electronically. A local news group created for the course is used by lecturers, tutors, and students for broadcasting announcements and discussing any issue that does not demand immediate attention, although cannot wait until next tutorial. The World Wide Web (WWW) is used to structure a large bulk of information about the tutorials, tutors, lectures, syllabus, assignments, tips and tricks, manuals, and links to related topics which can be studied further, see [2]. Although the WWW is excellent for structuring information and cheap to maintain, it is not a panacea. It is difficult for the reader to identify changes, and some important information must in addition be distributed on paper such as the compulsory project assignments and examination requirements.

The students are assessed by an oral examination based on about 12 questions covering the syllabus. These questions are published in advance. Each student starts the exam by drawing a random examination question and then gets half an hour to prepare for the oral examination. In the examination the student presents and discusses the topic with the lecturer, witnessed by an external examiner. The lecturer and the external examiner negotiate a mark — usually approximately 80% of the students pass.

2.3 On the Teaching with CP-nets and Design/CPN

CP-nets are simple to teach and easy to learn, because of the few elementary concepts involved. After a few hours of lectures the students are able to solve simple problems and understand more sophisticated, though still, toy examples. The most serious obstacle is how to use CP-nets as a technique for designing and analysing systems. As with other expressive computer languages it takes time to combine and apply the elements of CP-nets to structure complicated systems. It takes several weeks to master the CP-net technique on a level where the students are able to apply CP-nets to problems of the size of the project assignments. Practice and constructive feedback from tutors are the means to overcome this problem. Additionally, knowledge about type systems is practically a precondition when learning CP-nets. Although students normally know about types from previous courses, it may in some forums be an advantage to teach PT-nets instead.

A problem of technical character is to understand and to use features of the CPN tool, Design/CPN. In this case the students can take advantage of their skills acquired from previous courses, e.g., they already have a background in functional programming which can help them get acquainted with the inscription language of the tool — the functional language Standard ML [16]. However, our general experience is that students in the beginning have difficulties using Design/CPN. In spite of the students' background we feel this is more than expected. The many features seem to be overwhelming. Moreover, many also confuse the theory of CP-nets with the tool.

In the following section, we elaborate on the compulsory assignments, which binds together the two topics of distributed systems and CP-nets in the course.

3 Compulsory Project Assignments

During the semester the students are given two compulsory project assignments to solve. Each of the assignments nominally takes one third of the students' available time for three weeks. The assignments are solved in groups of three persons, which adds up to 100–120 person-hours for each assignment. Below we first describe the preparations of the assignments. In this section, we suggest a set of criteria for judging the quality of a project assignment idea. Then we describe the realisation and experience with our most developed and recent project work idea; viz. the design, validation, and implementation of a layered communication protocol to be applied as a component in a shared drawing system.

3.1 Criteria for Selecting a Project Assignment

Initially we defined a set of criteria to help us select among many ideas for project assignments — a simple framework with the purpose of helping us to design a project of sufficient quality in order to make the students motivated and interested in generating creative solutions and engender discussions. In the following we list the set of criteria that we used:

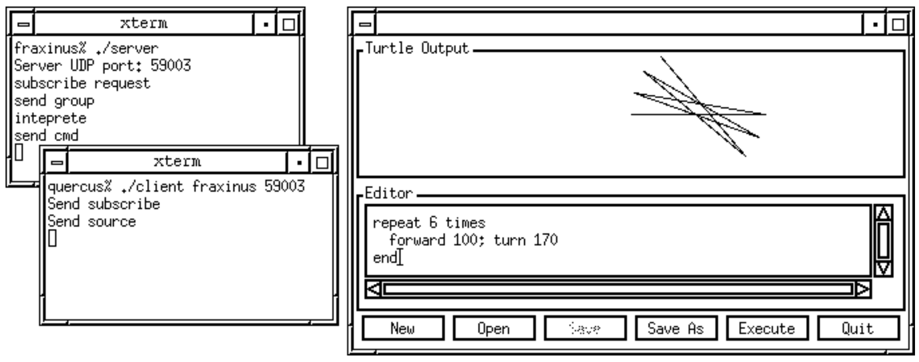
– *The project assignment should exercise the concepts from the course textbooks.* The students must prove their ability to apply and combine concepts to form a solution. From CP-nets they should apply, e.g., hierarchical organisation and data types in order to structure the solution. From distributed systems they should apply, e.g., concepts like group communication, threads, and synchronisation.
– *The project assignment should have a meaningful context.* We would like to avoid projects which are difficult to motivate due to contents without connotation that only suit the purpose of exercising textbook theory. We are looking for non-trivial problems which promote creativity and independent problem-solving, and which at the same time encourage mutual inspiration among the groups.
– *The project assignment should not alienate.* We want to choose a subject which is interesting for everyone. As an example, a distributed version of some arbitrary violent arcade action game is not necessarily of interest because many students would be likely to dissociate themselves from it.
– *The project assignment should have industrial characteristics.* The students should learn to make an approximate solution to an unsolvable problem. Some possibilities are to make the formulation of the assignment intentionally incomplete and ambiguous, to allow a wide range of very different solutions to be acceptable, to implant unsolvable problems, and to imitate well-known real-world distributed systems.
– *The project assignment should be reusable.* It demands a lot of work of the lecturers and teaching assistants to create an assignment from scratch. We can save a significant amount of time if variations of the assignment can be used through several semesters. A problem which can be designed in a modular fashion is a good candidate because that makes it easy to make local variations without the need to modify the whole assignment. Finally, it is difficult to make a perfect project assignment the first time. Experience from previous semesters can be used to refine the assignment over time.

– *The project assignment should be solvable within the given time frame, but should at the same time be open to further work for the ambitious and curious students.* The intention is, on one hand, to force the students to make a decision on when a solution is acceptable, and on the other to suggest digressions and encourage pursuit of their own ideas.

Over the years, we have carried out different projects: a security system, a multi-user chat system, and a shared drawing system. In the following, we describe the shared drawing system which is the most refined and recently used idea.

3.2 Overview of the Shared Drawing System

The project work is divided into two assignments, which are set separately during the semester. The two parts are interconnected as they are based on the same problem, namely to study communication in a layered protocol to be used in a distributed system. The shared turtle drawing system works in the following way (see also Fig. 1): A number of clients each have a user interface with a display



```
fraxinus% ./server
Server UDP port: 59003
subscribe request
send group
inteprete
send cmd
```

```
quercus% ./client fraxinus 59003
Send subscribe
Send source
```

```
repeat 6 times
  forward 100; turn 170
end
```

Fig. 1. The shared drawing system user interface with one client (rightmost window) connected to a server.

area (“Turtle Output”), and a text input area for turtle instructions (“Editor”).[1] When a client is started it subscribes to the current group via a server. All clients subscribed share the drawing area such that if one client instructs the turtle to move forward then the turtles on all other clients do the same. Clients cannot translate turtle instructions into drawing codes. Instead they must first be sent to the server for translation. Once translated, the code is sent back to the client which subsequently broadcasts the code to all subscribed clients such that all displays can be updated.

[1] The turtle language is inspired by LOGO [4].

The clients run concurrently and can therefore make broadcasts simultaneously. Such simultaneous broadcasts (of drawing code) must be consistently ordered, otherwise the drawings on the displays of the clients may turn out differently. We could have chosen to let the server handle all broadcasts. But if all broadcasts are handled by the server then it would be easy to serialise everything, and thus the students would not be exposed to the interesting issue on consistent ordering of simultaneous broadcasts.

As an example of a shared drawing system configuration consider the case with a server and three clients (see Fig. 2). We assume that clients 1 and 2 have

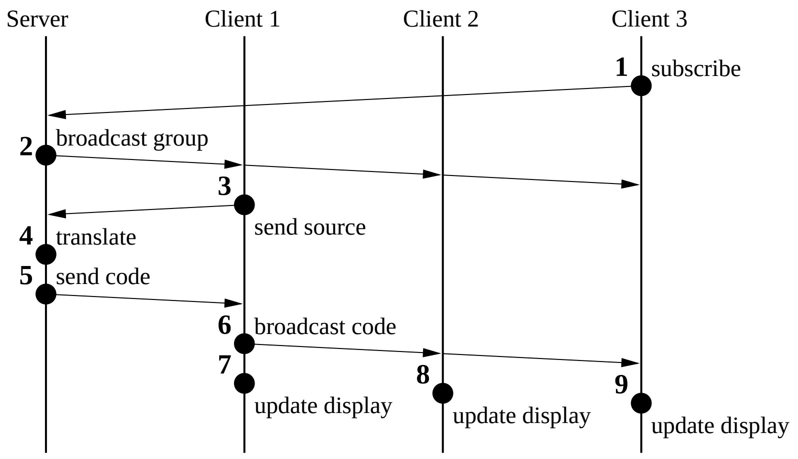


Fig. 2. Message sequence chart representation of a typical course of events in the shared drawing system.

subscribed (connected) to the server. Then client 3 subscribes (label 1) and as a result the server updates its group database and subsequently broadcasts the new database (label 2). At some point the user at client 1 wants to execute some turtle instructions which results in a message with the source code sent to the server (label 3). The server translates the source into primitive drawing code (label 4) and sends the code back to the client (label 5). Client 1 then broadcasts the code to all subscribed clients (label 6) and subsequently each involved client can execute the primitive drawing code resulting in updating of the displays (labels 7–9).

This project assignment exposes the students to central topics from Tanenbaum's book such as layered protocols, group communication, synchronisation, and threads. Furthermore, the students exercise systems engineering activities such as making a design based on a requirements specification and validating and implementing the design. Indeed such a combination of topics and principles creates a scenario which has industrial characteristics.

3.3 Preparation of the Project Assignments

As a proof of concept we solved each assignment completely before they were set to the students. During our own problem solving phase we gained detailed knowledge of both practical and conceptual problems which the students would be likely to experience. Finding a solution ourselves proved that the assignments were feasible and provided hints on the workload. The insight gained from problem solving is also useful in relation to supervision during the tutorials, and to prepare the final formulation of the assignments.

Each assignment set to the students was based on the full solution: We took our own solution and cut away the contents of a number of protocol layers, and left only a minimal functioning system. The task of the students was then to complete the missing parts, and use the minimal functioning system as a guideline for problem solving.

As an alternative we considered letting each group work on only one layer. When finished the groups should then combine their solutions to form a complete system. Although it is a realistic scenario, such an approach would be disastrous if a group fails in making a functioning layer. Only one missing layer would affect the other groups, and the students would be likely to get a negative experience.

In the following, we describe each of the two assignments in detail.

3.4 Project Assignment Part One — Design and Validation

The tool which is used in the first assignment is Design/CPN [11]. It supports hierarchical CP-nets and integrates an editor, a simulator, and a state space analysis tool. Design/CPN has been used in many industrial projects (see, e.g., [8,3]) and has matured significantly over the years.

The wording of the assignment provides the students with an overview of the protocol, which is depicted here in Fig. 3. The layered protocol consists of 6 layers, each modelled by a CPN module. We use the hierarchy facility of CP-nets to structure the relationship between the layers. Without the hierarchy it would be hard to manage a model of this size. In the assignment the students' task is to complete the design of layers 3–5, where the only constraints and guidelines are a fixed interface specification, and an incomplete and ambiguous specification of the expected functionality. The layers 3–5 are provided as a scaffolding with minimal functionality – a supporting framework. Without such a scaffolding, the students would unnecessarily spend a lot of time on diverging decisions. A scaffolding provides a common starting point and ensures more uniform solutions. Although it is important to get experience with building larger models from scratch, the given time-frame of the course does not make it practically possible for including this aspect.

The other layers are given to the students as complete functioning modules. The CPN model works under the assumptions that there is only one server and client, and that communication is reliable. Therefore the students can initially simulate the model to get an impression of behaviour and the flow of messages. The top-level module of the hierarchical CPN model is depicted in Fig. 4. The

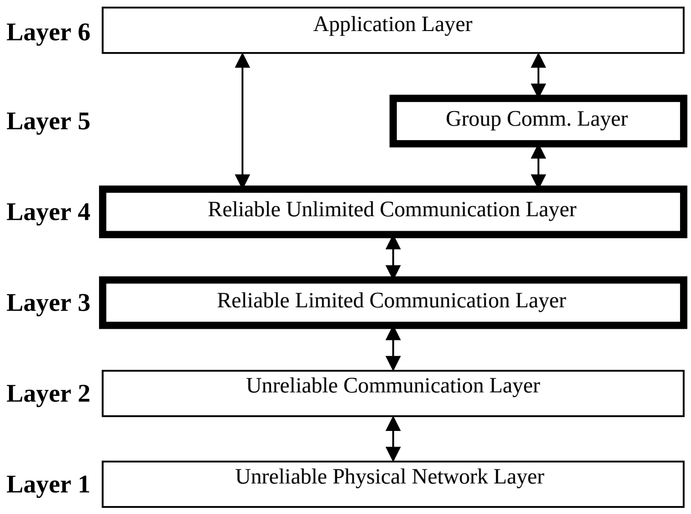

Layer 6
Application Layer
Layer 5
Group Comm. Layer
Layer 4
Reliable Unlimited Communication Layer
Layer 3
Reliable Limited Communication Layer
Layer 2
Unreliable Communication Layer
Layer 1
Unreliable Physical Network Layer

Fig. 3. Top-level overview of the layered protocol.

students immediately recognise the structure from the overview of the specification of the layered protocol in Fig. 3. In the following we describe each layer, and our idea of what the students should experience. The interface of each layer services a send and a receive primitive.

Layer 1 is a representation of the physical network. The network is unreliable such that messages can be lost, duplicated, and overtake each other. Nothing is said about message corruption in the wording of the assignment, which is then left as a decision to the students whether this complication should be taken into account or be considered a limitation. It is perfectly acceptable to disregard corruption of messages. Message corruption is an example of a hidden choice in the assignment. The CPN model of this layer has parameters which determine how reliable the network should be. The students can take advantage of adjusting these parameters while validating the protocol.

Layer 2 is a simple interface to the physical network, and supports unreliable communication of fixed length messages. In the layers 2–5 it is specified that the send primitive is called as an ordinary blocking procedure, while the receive primitive acts as a call-back. The students know the concept of blocking from Tanenbaum: “The instruction following the call to *send* is not executed until the message has been completely sent...” With such a vague definition it is up to the students to decide exactly when the sender can be unblocked; from when the message is on the network to when the message has been completely stored in the receiver. Although the concept is discussed in Tanenbaum’s book the students still must make choices. By *call-back* we understand a mechanism where the call-chain is directed upwards in the layers instead of downwards which is the case for the send procedure. The reason we use call-backs is to make the

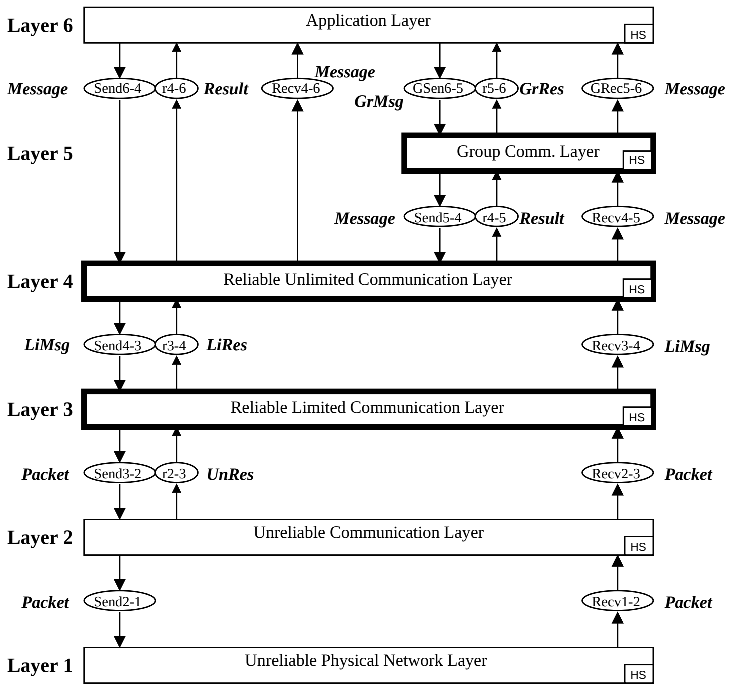


Fig. 4. Top-level module in the hierarchical CP-net model of the layered protocol. (The HS-tags denote substitution transitions.)

receive-call passive in some sense in order to avoid active polling or busy-waiting for incoming messages. With call-backs the receive-call is only active when a message has arrived. Although the call-back mechanism is an advanced topic it turned out to be a useful abstraction for the students. There is no restriction on the number of simultaneous sender and receiver threads in the layers.

In **layer 3** the students should accomplish reliable communication of fixed length messages. Since the underlying layer is unreliable the students need to investigate the degree of reliability. One problem to consider is what should happen if a receiver does not respond. The sender does not know if the receiver is slow or has crashed. If we do not want infinite waiting then design of time-outs is inevitable. Although time seems to be explicit the students should not use timed CP-nets. In this course, we are only interested in the logical behaviour of systems and not in performance measures.

Another problem is global knowledge (also called the consensus problem). Suppose that a sender sends a message to a receiver. Given unreliable communication, can it be ensured that both parties agree that the message has been received? Researchers of distributed systems have already proved that attempting to ensure global knowledge is futile — it is unsolvable in general for asynchronously distributed systems (see the consensus problem in Lynch's book [13]).

The students do not know that they have an unsolvable problem. First they must identify that they have a problem with global knowledge, and then they must design an approximate solution which handles the problem acceptably. In the documentation of the assignments the students need to explain which assumptions and decisions they make.

In **layer 4** the students should accomplish reliable communication of messages with arbitrary length. The layer below is already reliable, therefore the problem is reduced to splitting messages that are longer than the fixed length restricted by the lower layer. The sender should split messages and the receiver should assemble messages in the right order. The standard solution is to label each sub-message with two serial numbers: one for identifying the whole message such that two concurrent messages are not mixed, and one for identifying the sequence of sub-messages. This layer is similar to the transport layer in the OSI protocol which is also described in Tanenbaum's book.

In **layer 5** the students should accomplish reliable group communication of arbitrary length messages. The layer must guarantee consistent time ordering of messages.[2] The students must use the ABCAST algorithm to ensure consistent time ordering. ABCAST is only described briefly in Tanenbaum's book, therefore the students get a paper on the algorithm [12] which provides a complete but abstract description. The challenge is to understand the paper and make an interpretation in terms of a CP-net description.

Layer 6 is a simplified representation of the application using the layered protocol as a library. It is not a complete CPN model of the shared drawing system but a minimal environment sufficient for validation. A complete model of the shared drawing system is not suitable because we are only interested in validating the layered protocol.

It is our intention that layers 3–5 can be completed independently of each other and in any order, although working from 3 to 5 is the most natural order. As the typical group size is three, there is one layer for each student. The students are already familiar with the strength of splitting work to manage high complexity, and this is also what most groups do in this case.

Once the members of a group work independently they are faced with many choices for each layer as indicated above. The group must meet frequently to synchronise their individual decisions and to discuss alternatives. Choices have to be made about degree of concurrency, critical regions, blocking, and data structures — just to mention a few. The tutors are always available for advice, and the weekly tutorials are still used for general discussions.

The students must also validate their designs, i.e., convince the tutor (and themselves) that their solution works according to the requirements of the specification. Simulation is used throughout the entire design process by the students,

[2] In group communication with simultaneous broadcast *consistent time ordering* is ensured by requiring an agreement of exactly one common receiving order. It is a more realistic and pragmatic method than global time ordering (which relies on having a global clock).

therefore they have detailed knowledge about the behaviour of the system from early on in the design process. It is required that the students use annotated simulation runs as documentation for the validity of the system. The students realise that the analysis does not formally prove that the design meets the specification. In return they learn the lesson that modelling and validating is a means to get detailed knowledge of behaviour and to identify errors. As a result their confidence in the system is increased.

Finally, an important part of the assignment is to document their work. It is required that the students provide a description of their solutions including arguments for choices and limitations made during the design process. The students are already familiar with documenting sequential textual languages, but here they are faced with a new kind of challenge: to explain a graphical CPN model exhibiting concurrency.

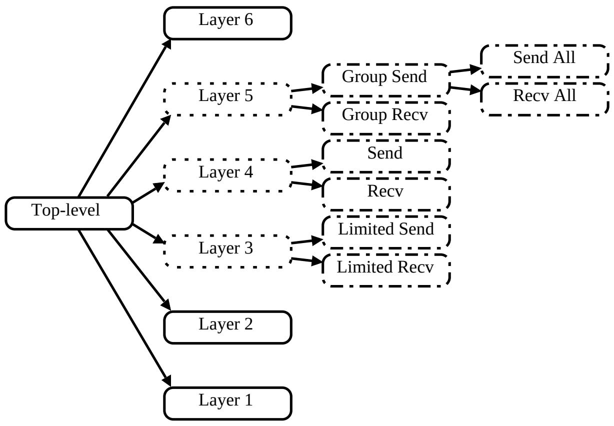
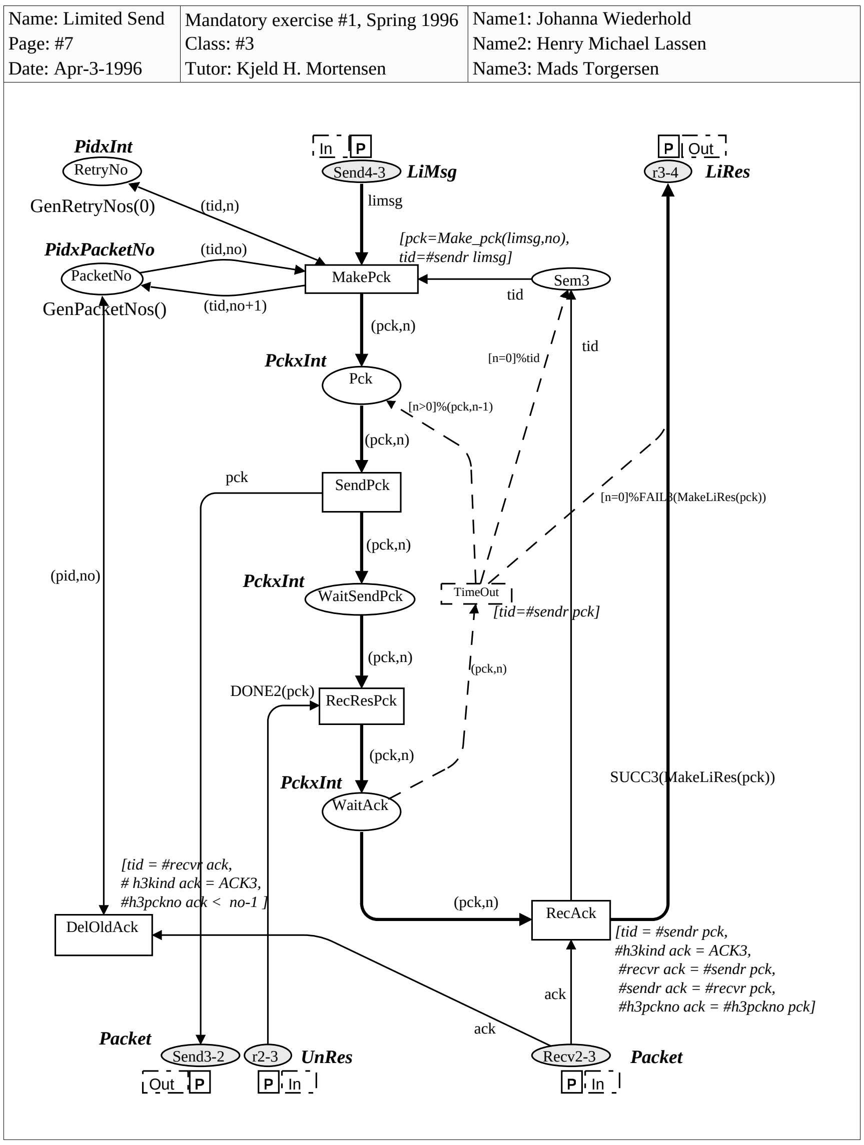


Fig. 5. Hierarchical structure of modules in a typical solution. The module *Top-level* corresponds to Fig. 4.

3.5 A Sample Design Solution

The solutions handed in are very different. In general, most students solve the assignment satisfactory. In this project assignment, the most conspicuous difference was the degree of concurrency realised. Many design proposals realised concurrency at a rather coarse level where at most one thread was allowed to exist in one or several layers. Some proposals suggested many simultaneous threads in each layer, and encapsulated shared data structures with critical regions. In Fig. 5 below there is an example of the hierarchical structure of modules in a typical solution. This group of students have added 8 CPN modules (dot-dash nodes). The dotted nodes are the 3 scaffolding modules which all groups built on as a starting point, and the solid border nodes represent the supporting modules which were given. Figure 6 depicts one of the modules from layer 3 (*Limited Send* in Fig. 5), and is similar in complexity to the other modified or added modules. Note that the group has used various graphical attributes to enhance readability



Name: Limited Send	Mandatory exercise #1, Spring 1996	Name1: Johanna Wiederhold
Page: #7	Class: #3	Name2: Henry Michael Lassen
Date: Apr-3-1996	Tutor: Kjeld H. Mortensen	Name3: Mads Torgersen

Fig. 6. A CP-net module of a typical solution (see Fig. 5).

in order to guide the reader; thick arcs are used to emphasise the main flow of messages and dashed arcs to identify time-out constructions.

The module in Fig. 6 works as follows: The purpose of the CPN model is to describe the flow of messages. Threads are implicit in the way that a message or packet contains information about its associated thread by means of a *tid* (thread identity). The entry point to the send primitive of this layer is modelled by the top-most input place *Send4-3*. When layer 4 calls send in layer 3 a message token is put on *Send4-3*. Nothing will happen if this layer is already processing a message, i.e., the semaphore modelled by the place *Sem3* is empty and thus the transition *MakePck* disabled. Otherwise, the fixed length message *limsg* (limited message) is first converted by the transition *MakePck* to a packet format appropriate for layer 2 by means of the function *Make_pck*. This function also associates to the packet a unique serial number, *no*, which is generated from the place *PacketNo*. Additionally, a number, *n*, specified by the place *RetryNo*, determines the maximal number of packet retransmissions. The result *(pck,n)* is put on the place called *Pck*, and the semaphore on the place *Sem3* is cleared such that the critical region is opened for waiting calls. Now this layer calls the send primitive in layer 2 which is modelled by placing the packet *pck* on the output place *Send3-2* via transition *SendPck*. The "thread" *(pck,n)* is now blocked modelled by waiting on the place *WaitSendPck*. When the send call in layer 2 has completed a result token is put on the place *r2-3* by layer 2, and the thread can be unblocked by the occurrence of transition *RecResPck*. The thread in question is now located on the place *WaitAck*, and two possibilities exist. Either an acknowledgement is received via the place *Recv2-3* or a time-out can occur. Time-out is modelled by the occurrence of the dashed transition *TimeOut*.[3] If the re-transmittal count, *n*, has not reached zero then the packet is attempted sent again by putting it back on the place *Pck* with *n* counted down by one. Otherwise, a failure result is returned back to layer 4 via the place *r3-4*, and also remember to set the semaphore *Sem3* such that possibly waiting send calls can proceed in this layer. On the other hand, an acknowledgement may be received while waiting on the place *WaitAck*. It is known that it is the right acknowledgement for the waiting thread because the *tid* associated with the packet is also sent back with the acknowledgement. The guard sub-expression *(#h3pckno ack = #h3pckno pck)*[4] of transition *RecAck* checks that it is the right acknowledgement. If this is the case a success result is returned to layer 4 via the place *r3-4* and again remember to set the semaphore *Sem3* to allow entrance to the critical region of waiting send calls. Outdated acknowledgements are discarded by the transition *DelOldAck*.

A different group experimented with design ideas involving more advanced multi-phase commit of acknowledgements in this protocol layer, and yet another

[3] An expression of the form *[b]%exp* evaluates to *exp* if *b* is true, otherwise evaluates to the empty multi-set.

[4] The # notation denotes access to a record field. Consider, e.g., the expression *#h3pckno ack* which means that the *h3pckno* field of the record *ack* is extracted.

insisted on making the layer as concurrent as possible. The students clearly exhibited creative problem solving.

3.6 Project Assignment Part Two — Implementation

The second part of the compulsory project assignment is to implement the layered protocol in the object-oriented language BETA. The programming environment used in this assignment is the Mjølner BETA system [14]. BETA is an object-oriented language which is usable for industrial size applications (see, e.g., [5]). Historically, the language has followed the Simula tradition. BETA is mainly developed by the computer language group at DAIMI.

It is advantageous to choose a language like BETA for several reasons. Firstly, the computer science curriculum at DAIMI already contains a course on computer programming, where the students are exposed to BETA. This means that all have general knowledge about the language, and therefore can solve the assignment based on the same premises and qualifications. In fact, we have experience with using C as the implementation language in this course from an earlier semester. Students who were unfamiliar with C were overly inhibited by technical problems, thus having a negative experience. Secondly, DAIMI has a language design research group on BETA, and also many tutors are experts on the language. Therefore, we have a significant resource for BETA language support when designing the development environment for the students. Finally, a language like BETA is more general purpose than, e.g., C++ or Java. It is difficult to program threads and call-backs in C++, while both have elegant support in BETA. BETA supports concurrent programming, has support for programming in-the-large, and a comprehensive class framework for interface building. Therefore, for our purposes, BETA is a natural choice for the implementation language for the second project assignment.

As in the previous assignment we provided the scaffolding; a minimal functioning system as a starting point. One part was to provide the graphical user interface (see Fig. 1), and another to provide the implementation of the layers not considered by the students (layers 1, 2, and 6). In this assignment the application layer was fully implemented. Apart from making the implementation interesting, the purpose of the shared drawing system is to provide an environment for testing of consistent time ordering. If the students make a flaw in the implementation of consistent time ordering, it will be immediately evident from the graphical output: the graphics will be different in one or more of the windows of the clients. The turtle has its own state which is a position, a direction, and whether the pen is up or down. Assuming that the pen is down, consider the two commands “move forward 100 units” and “take pen up” issued concurrently from two clients, respectively. The order of the two commands results in different drawings. For testing purposes the students could again set parameters via shell environment variables to determine how reliable layer 1, the physical network, should be. The local area network at DAIMI is too reliable to be used for stress testing. The students realise that it is sometimes necessary to imitate worst case scenarios in order to test extreme situations which would otherwise be unlikely to occur in real life.

The students commented that implementation work was surprisingly simple, which they credited to the design assignment. They had become familiar with the behaviour of the system and could use the design as a recipe. The only real challenge was how to interpret the CPN model as an object-oriented system — the usual obstacle caused by the gap between paradigms used in the design and implementation phases. Again there were a range of suggestions; from interpreting the elements of the CPN model literally, i.e., considering places and transitions as BETA objects, to interpretations more detached from the design.

Another requirement of this assignment was to consider behavioural relations between design and implementation. The task was to identify and justify differences in behaviour rather than to prove that the behaviour of the implementation reflected the design.

3.7 Assessment, Correction, and Evaluation of the Students

The students are exposed to a lot of material and exercises during the course. We observe the students in a variety of problem solving scenarios in order to evaluate their performance. On an informal level the tutorials are useful to get an impression on how students are able to solve problems, on the semi-formal level there is the compulsory project assignment, and on the formal level there is the final exam.

A recent survey shows that a majority of the graduates at DAIMI get jobs as software developers. The project assignments induce a scenario where the students get familiar with future work situations. Working in groups naturally requires the essential ability to cooperate. The cause of poor performance is typically failure to cooperate, since the assignment is too large to be solved individually within the time frame given. Another performance factor is related to the ability to think abstractly. In the design part of the assignments, there is a tendency that students who get too involved with low-level technical details run out of time compared with students who abstract away technicalities. Then there is the ability to decide when a design or implementation is deliverable. When is, e.g., a CPN model sufficiently detailed? Are the students able to make limitations on an appropriate level?

As part of the project assignments the students are evaluated resulting in corrections and constructive critique. Each group hands in one report to be assessed by the tutor. In order to prevent a sleeping partner in a group the students must present their work at two tutorials. In the first tutorial each group must make a live demonstration of their implementation in a lab session (which often is very entertaining — especially when groups exchange their implementations for mutual testing). The lab session exercises software demonstration. In the second tutorial each group must present and defend their work to the other students in order to exercise oral presentations. As an experiment for the presentation session we let two groups present the work of each other — a critical mutual review. The tutor advised them on how to approach such a review. This turned out to be rewarding, because the focus of presentation was not simply biased toward the strong points in the solutions. Weak points were identified and critically analysed.

The two evaluation tutorials described above are supervised and guided by the tutor who has read, in advance, all the reports handed in; possessing the role of being both a catalyst and moderator at the same time. An assessment is made based on the students' solutions which results in either a pass or fail mark — a fail mark happens seldom.

4 Ideas for Future Improvements

In general, the question is how much can be put into the syllabus of a course. Some topics, such as simulation of CP-nets, depend crucially on tools and the power of computers. As tools mature, the more advanced topics can and should be considered for inclusion. Also the faster the computers are and the more memory they have has a strong influence on what can be considered feasible topics. As an example, during the period we have taught the course, the memory has doubled and the speed has quadrupled of our lab computers. As a result, the project assignments have grown in size and complexity — without compromising feasibility. For the future we consider including into the syllabus aspects of verification methods such as the state space method. It can be used in an easy fashion without requiring the user to conduct hard and error-prone manual proofs, and is therefore a potential candidate for an easy accessible introduction to verification. Other verification techniques with CP-nets are introduced in a subsequent advanced course based on material in [10].

As part of the requirements of the project assignments the students must document their work. We are considering to use a hyper-media tool developed by the hyper-media group at DAIMI [5]. Design/CPN already supports communication with the hyper-media tool which means that it is possible to create a hyper-text where, e.g., there are links between CPN elements (such as places and transitions), documentation (text), pictures, etc. In this way the students can be exposed to state-of-the-art documentation techniques.

We have implemented a CPN library supporting message sequence charts (see Fig. 2), which is a standardised notation that is prevalent in the telecommunications industry [6]. It is our expectations that message sequence charts can easily be used by the students in the first project assignment. In this way they also become familiar with another notation used in industry.

5 Conclusion

The core of the course is the compulsory project assignments. The students get practical experience with design, validation, implementation, and documentation. In the beginning the students tend to be reluctant to using formal methods and developing detailed designs, but during the implementation phase they realise that what we lose on the roundabouts we gain on the swings. Although some amount of time is spent on designing and validating a CPN model the students experience that implementation suddenly is the easy task. It is an advantage to make a (CPN) model of the system, which is a prototyping activity, and then use the resulting design as a recipe for implementation. As an additional profit we get a design, implementation, and documentation which is cheaper in maintenance.

Using a graphical language such as CP-nets often proves to be a helpful didactic method for introducing new concepts. We agree with Oberquelle [15] and Jantzen [7] that Place Transition nets (PT-nets) are a useful language for explaining concepts on both the informal and rigorous level. However, the students are quickly faced with the limitations of PT-nets because, even for toy examples, such as the Dining Philosophers, they lack encoding facilities for data. CP-nets, and high-level nets in general, are more applicable than PT-nets for visualising large systems due to the abstraction facilities such as hierarchies and data types. Our experience is that it is possible to use CP-nets as the starting point in teaching the pragmatics of nets, thus avoiding simpler and less succinct net-formalisms such as PT-nets. The main principles can be taught in a matter of a few hours, and CP-nets can thus be used very early in an intuitive fashion to illustrate both static and dynamic aspects of systems.

We have argued that a course syllabus combining distributed systems with CP-nets provides an interesting and rewarding framework for teaching undergraduates. Analysing complex concurrent systems demands a systematic approach such as CP-nets. Teaching CP-nets requires illustrative interesting real-world examples for which the area of distributed systems contains numerous examples — especially topics like concurrency control, communication, and protocols contain a plethora of interesting issues suitable to be explained by means of CP-nets. The improvement of the integration of the two parts of the course, distributed systems and CP-nets, is an ongoing task. But we have gained useful experience so far. A good textbook, which introduces general concepts on synchronisation, communication, and resource sharing, is still required.

Acknowledgements. Kurt Jensen is the principal lecturer on the course described in this paper. He exhibits a strong commitment to teaching. We thank Liam Bannon, Kurt Jensen, Jens B. Jørgensen, and Lars M. Kristensen for discussions and comments on earlier versions of this paper. The sample solutions of the project assignments were provided by Henry M. Lassen, Mads Torgersen, and Johanna Wiederhold.

This work has been supported by a grant from the Danish Research Council STVF.

References

1. W. Brauer, editor. *Net Theory and Applications: Proceedings of the Advanced Course on General Net Theory of Processes and Systems*, volume 84 of *Lecture Notes in Computer Science*, Hamburg, October 1979. Springer-Verlag.

2. Coloured Petri Nets at the Computer Science Department, University of Aarhus, Denmark. World-Wide Web: http://www.daimi.aau.dk/CPnets/.
3. Design/CPN Online. World-Wide Web: http://www.daimi.aau.dk/designCPN/.
4. W. Feurzeig, G. Lukas, and J.D. Lukas. *The LOGO Language: Learning Mathematics Through Programming.* Entelek, Inc., Portsmouth, NH, 1977.
5. K. Grønbæk and R.H. Trigg. Design Issues for a Dexter-based Hypermedia System. *Communications of the ACM, Vol. 37, 2*, 1994.
6. International Telecommunication Union; Telecommunication Standardization Sector (ITU-T). ITU-T Recommendation Z.120: Message Sequence Chart, Geneva, Switzerland, 1993.
7. M. Jantzen. Structured Representation of Knowledge by Petri Nets as an Aid for Teaching and Research. In Brauer [1], pages 507–517.
8. K. Jensen. Coloured Petri Nets — Basic Concepts, Analysis Methods and Practical Use. Volume 3, Practical Use. To be published by Springer-Verlag.
9. K. Jensen. *Coloured Petri Nets — Basic Concepts, Analysis Methods and Practical Use. Volume 1, Basic Concepts.* Monographs in Theoretical Computer Science. Springer-Verlag, 1992.
10. K. Jensen. *Coloured Petri Nets — Basic Concepts, Analysis Methods and Practical Use. Volume 2, Analysis Methods.* Monographs in Theoretical Computer Science. Springer-Verlag, 1994.
11. K. Jensen, S. Christensen, P. Huber, and M. Holla. *Design/CPN Reference Manual.* Computer Science Department, University of Aarhus, Denmark, 1996. Available from [3].
12. T.A. Joseph and K.P. Birman. Reliable Broadcast Protocols. In S. Mullender, editor, *Distributed Systems*, Frontier Series, chapter 14, pages 293–318. ACM Press, 1989.
13. N.A. Lynch. *Distributed Algorithms.* Morgan Kaufmann Publishers, Inc., 1996.
14. O.L. Madsen, B. Møller-Pedersen, and K. Nygaard. *Object-oriented Programming in the BETA Programming Language.* Addison Wesley, 1993. World-Wide Web: http://www.mjolner.com/.
15. H. Oberquelle. In Teaching and in Terminology Work. In Brauer [1], pages 481–506.
16. L.C. Paulson. *ML for the Working Programmer (2nd Edition).* Cambridge University Press, 1996.
17. A.S. Tanenbaum. *Modern Operating Systems.* Prentice-Hall International, 1992.