

# Introduction to constraint programming

## Lecture VI

Per Kreuger [piak@sics.se](mailto:piak@sics.se) &  
Markus Böhlin [bo1a@sics.se](mailto:bo1a@sics.se)

January 2003



### Specification and classification of *global* constraints

- Recently systematic methods to specify and classify global constraints have been developed (Beldiceanu 2000)
- The following slides introduce the main ideas of a formalisation of global constraints used in this work
- The approach uses a more general notion of constraint graph with certain regularity requirements which classification more straightforward

## Characterisation of a global constraint

1. Constraint parameters (variables and type restrictions)
2. Elementary constraint (e.g.  $=$ ,  $\neq$ ,  $\leq$ ,  $<$  etc.)
  - also for compound types
3. Graph generator (*regular structure*)
  - (a) Vertices
  - (b) Arcs (*one type of arc constraint*)
    - (e.g. self, loop, path, clique)
4. Graph property (e.g. clique, no of vert., no of conn. comp. etc)

## Example: all-different

**Vertex generator**  $id(Vars)$

**Arc generator**  $clique(Vars)$

**Arc constraint** =

- Not  $\neq!!!$

**Graph property**  $max\ nsc \leq 1$

- Different constraint for e.g.  $max\ nsc \leq 2$

## Hyper v.s. primal v.s. final graph

Constraint hyper graph	Initial graph	Generic primal graph

- Primal graph is result of applying graph generator
- Final graph is result of enforcing graph property

## Sweep — A generic pruning method

N.Y.I.

# Modeling

- The computational complexity of the task to find a solution to a given problem depends to a large extent on the expressive power of the language used to formalise the problem
- To formulate a mathematical model of some real process is generally a difficult task that requires a thorough understanding of both the problem domain and the methods employed to solve the problem
  - To some extent this is still more of a craft than a science
  - A large body of typical problems with standard models have been identified
  - Early attempts to develop a methodology started to give results

## Modeling examples

1. Puzzles
  - (a)  $N$ -queens (with local & global constraints)
  - (b) Magic squares (?and sequence?)
2. Planning and scheduling

## Example 1 — The $N$ -queens puzzle

- Given a chess-board of size  $N$ , is it possible to place  $N$  queens on the board such that no queen can attack another queen?
- First, observe that each solution to the  $N$ -queens problem must have the queens placed on different rows and columns

## Formalisation

- Let the variables  $x_1, x_2, \dots, x_N$  denote the row position of queen  $i$ , located at column  $i$  and formalise the  $N$ -queens puzzle with the following constraints
  - Each queen placed on a row of the board —  $(\forall i) x_i \in \{1, 2, \dots, N\}$
  - No two queens placed on the same row —  $(\forall i, j) i \neq j \rightarrow x_i \neq x_j$
  - No two queens placed on the same diagonal —

$$\begin{aligned}(\forall i, j) i \neq j &\rightarrow i - j \neq x_i - x_j \\(\forall i, j) i \neq j &\rightarrow i - j \neq x_j - x_i\end{aligned}$$

## Three queens

- As an example, observing some redundancy in the generic formulation above the 3-queens problem can be expressed as the following system of inequalities:

$$\begin{aligned} & x_1, x_2, x_3 \in \{1, 2, 3\} \\ & x_1 \neq x_2, \quad x_2 - x_1 \neq 1, \quad x_2 - x_1 \neq -1 \\ & x_1 \neq x_3, \quad x_3 - x_1 \neq 2, \quad x_3 - x_1 \neq -2 \\ & x_2 \neq x_3, \quad x_3 - x_2 \neq 1, \quad x_3 - x_2 \neq -1 \end{aligned}$$

- This instance of the  $N$ -queens problem can be corresponds with the following constraint graph:

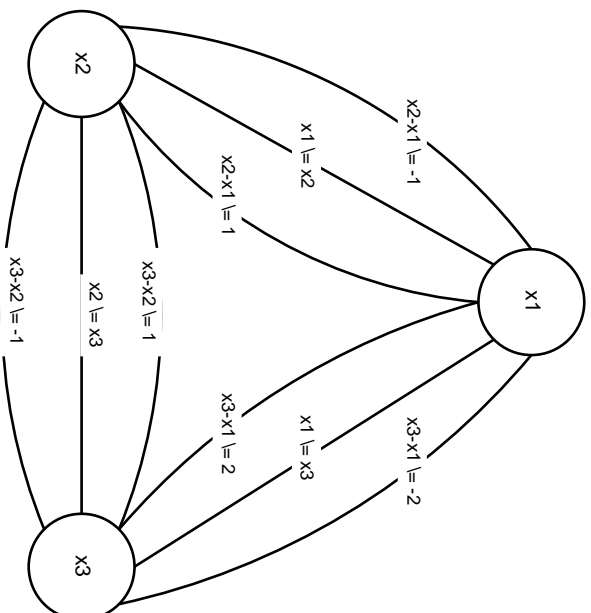


Figure 1: Constraint graph for the 3-queens problem

## Formalisation with global *all-different*

- Using the global constraint *all-different* (introduced by Régim in 1994) which simply state that a set of variables should take distinct values, the formulation can be reduced to:

```
 $x_1, x_2, x_3 \in \{1, 2, 3\}$   
all different ( $x_1, x_2, x_3$ )  
all different ( $x_1 - 1, x_2 - 2, x_3 - 3$ )  
all different ( $x_1 + 1, x_2 + 2, x_3 + 3$ )
```

## *n*-queens in Prolog

```
:- use_module(library(clpfd)).  
nqueens(L,S) :-  
    length(L,Le), domain(L,1,Le), all_different(L),  
    length(L2,Le), Ub is 2*Le,domain(L2,1,Ub),  
    all_different(L2),nqconnect(+,L,1,L2),  
    length(L3,Le), Lb is -Le, domain(L3,Lb,Le),  
    all_different(L3),nqconnect(-,L,1,L3),S=(L,L2,L3).  
nqconnect(-,[],-,[]).  
nqconnect(+,[C|L],I,[C2|L2]) :-  
    C2 #= C+I, I1 is I+1, nqconnect(+,L,I1,L2).  
nqconnect(-,[C|L],I,[C2|L2]) :-  
    C2 #= C-I, I1 is I+1, nqconnect(-,L,I1,L2).  
solve(L) :- nqueens(L,-),labeling([],L).  
/*solve([A,B,C,D]).*/
```

## Example 2 — The magic squares problem

- A magic square of order  $n$  is a  $n \times n$  matrix, containing unique occurrences of the integers  $1, 2, \dots, n^2$
- In a magic square, the sum of each row, column and main diagonal must be equal to the same value
- The problem is to find magic squares for a given  $n$ .

### Formalisation

- Let  $A$  be an  $n$  by  $n$  matrix for which it holds that the values in  $1, 2, \dots, n^2$  occur exactly once (i.e. an all different), and

$$\forall i \in \{1, \dots, n\} : \sum_{j=1}^n A_{i,j} = k$$

$$\forall j \in \{1, \dots, n\} : \sum_{i=1}^n A_{i,j} = k$$

$$\sum_{i=1}^n A_{i,i} = k$$

$$\sum_{i=1}^n A_{i,n-i} = k$$

## Magic squares in Prolog

```
:- use_module(library(clpfd)).
:- use_module(library(lists)).
flatten([], []).
flatten([M1|MR], L) :- flatten(MR, Lp), append(M1, Lp, L).
makelist([], []).
makelist([V1|Vr], [Lr]) :- makelist(Vr, Lr).
transpose([R], L) :- makelist(R, L).
transpose([R1|RR], L) :-
    transpose(RR, Lp), distribute(R1, Lp, L).
distribute([], L, L).
distribute([V1|Vr], [L1|Lr], [[V1|L1]|Lr2]) :-
    distribute(Vr, Lr, Lr2).
diagL([], -, []).
diagL([L1|Lr], N, [R1|Rr]) :-
    nth(N, L1, R1), N2 is N+1, diagL(Lr, N2, Rr).
diagR([], -, []).
diagR([L1|Lr], N, [R1|Rr]) :-
    nth(N, L1, R1), N2 is N-1, diagR(Lr, N2, Rr).
```

```
postsum(_, []).
postsum(K, [E|Lr]) :- sum(E, #=, K), postsum(K, Lr).
magic(N, M, L, K) :-
    length(M, N), findall(E, (member(E, M), length(E, N)), M),
    flatten(M, L), N2 is N*N, domain(L, 1, N2), all_different(L),
    K is (N2*N+N)//2,
    postsum(K, M), transpose(M, MT), postsum(K, MT),
    diagL(M, 1, DL), sum(DL, #=, K),
    diagR(M, N, Dr), sum(Dr, #=, K).
solve(N, (M, K)) :- magic(N, M, L, K), labeling([], L).
```

## Resource scheduling

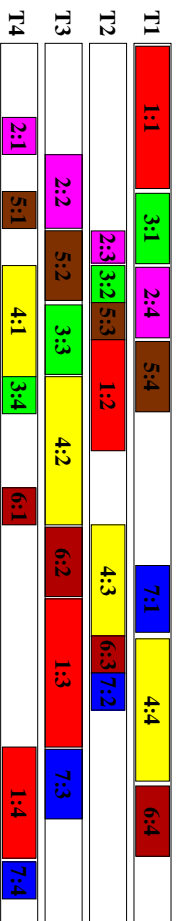
- A scheduling problem consists of a number of *tasks* with restrictions on *start times* and *durations*
- Often the tasks are *partially ordered* into totally ordered sequences
- Such a totally ordered subset of tasks is often called a *job*
- Each task uses one or more *resources* during its duration
- The *Job shop scheduling problem* is a classic and well studied case

## Resources

- Resources are abstractions used to model widely different type of entities. E.g:
  - Processing equipment in a production process
  - Staff or vehicles in a transport net
  - Network resources such as switches, routers and transport links with limited capacity in transport or information networks
- In scheduling resources have discrete capacity
  - a single task at any given time
  - a cumulative limit of simultaneously executing tasks

## Scheduling problems

- To arrange the tasks in time so that no limitations in resources are violated is called *to schedule* the tasks and it is in general a very difficult (NP-complete) computational problem
- Nevertheless the many practical applications for methods in this area make it fairly well studied



Gantt-chart for 6 jobs consisting of 24 tasks using 4 resources

## Resource formalisation

- Many of the best approaches to solving difficult scheduling problems have introduced as global constraints in constraint programming system
- For the classical job shop every resource is encoded with one global serialized constraint

$$\text{serialized}(Starts, Durs)$$

which enforce the condition that for each pair  $\langle i, j \rangle$  of tasks using the resource that

$$Starts_i + Durs_i \leq Starts_j \vee Starts_j + Durs_j \leq Starts_i$$

## Job formalisation

- Each job is encoded as a total order of tasks

$$Starts_i + Dur^r s_i \leq Starts_j$$

for each pair  $i, j$  such that task  $j$  follows immediately after task  $i$  in some job

- Note that
  - the number of inequalities used to represent the job is linear in the number of tasks
  - a single condition is used to represent the quadratic number of disjunctions representing the nonoverlap of the resource

## A larger planning problem:

### Production planning in the rail industry

- The following examples are taken from work done on scheduling and resource planning of rail traffic
- Production planning in the rail industry can be seen as planning the movements of trains according to a specification of train *trips* where each trip must be allocated
  - Track time slots
  - Vehicle resources
  - Travelling personnel

## Trip specification

- We assume that the trips needed to supply a certain transport requirement has been predetermined and is represented as a fixed set of trips with requirements on departure, traversal and arrival times, vehicle and personnel
- For each trip we assume furthermore that the path it will traverse in the track network is given
- This specification can be expressed using a specialised language allowing expansion of cyclic activities:



24

```
tripspec( pathName: 'ARB-HPBG-0'  
deptimeSpec: t(hour:6 minute:15..45)  
arrTimeSpec: t(hour:7 minute:0..30)  
trvTimeSpec: t(minute:40..50)  
locSpecs: locSpecs(  
  'ÖR': locSpec(stopTimeSpec:t(minute:2))  
  'ÖB': locSpec(stopTimeSpec:t(minute:2))  
  'KLA': locSpec(stopTimeSpec:t(minute:2)))  
repSpec: repSpec(  
  bnd: week  
  incr: hour(2)  
  dom: [t(day:mon..fri hour:6)  
    t(day:mon..sun hour:[8 10 12 14 16 18 20])] )  
tripType: tt(type:'PX260020')  
)
```



25

## Subproblems

- One can regard the three above mentioned problems as separate
- However they are obviously correlated
- One can solve the complete problem by solving each of the problems in sequence (A “waterfall” process) e.g:
  1. Tracks slots
  2. Vehicles
  3. Personnel
- We will look in more detail on two of the problems and their combination in the following slides

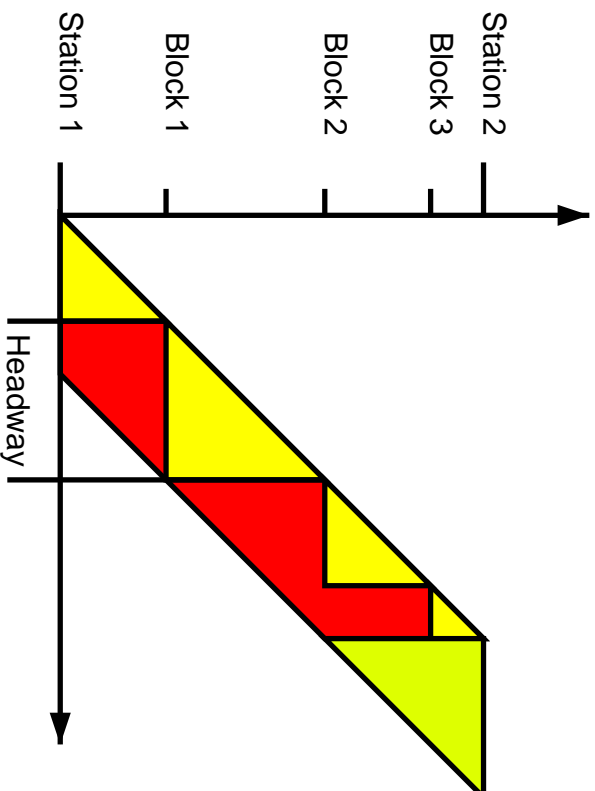
## Track slot allocation

- Track slot allocation can in principle be modeled and solved as classical job shop scheduling problem where
  - Track blocks are unary resources
  - Trains are jobs consisting of sequences of track slot tasks

**Problem:** Very large number of track blocks give practically insoluble scheduling problem

**Alternate approach:** Use abstract track resources consisting of several blocks and use more intricate scheduling model using several durations for each task (headways)

## Headway model



## Constraint model

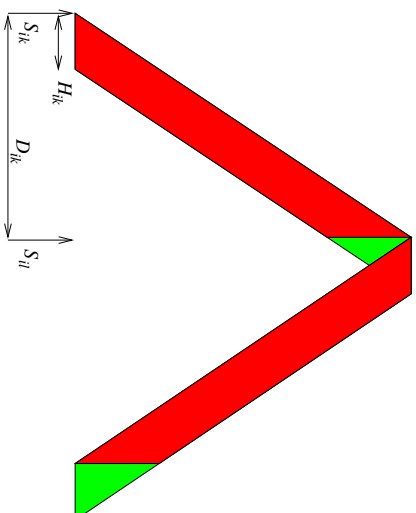
- We wish to enforce conditions that for each track that:
  1. For each pair of trips traversing the track in opposite directions, the intervals during which the traversals take place will not overlap
  2. For each pair of trips traversing the track in the same direction, headway is respected at both departure and arrival locations
- Define a (disjointness) relation  $\diamond$  on pairs  $(S, D)$  of time points and durations as follows:

$$(S_i, D_i) \diamond (S_j, D_j) \Leftrightarrow (S_i + D_i \leq S_j) \vee (S_j + D_j \leq S_i)$$

- Then for each track  $i$  and each pair of trips  $k$  and  $l$  traversing it one of the following two relations must hold:

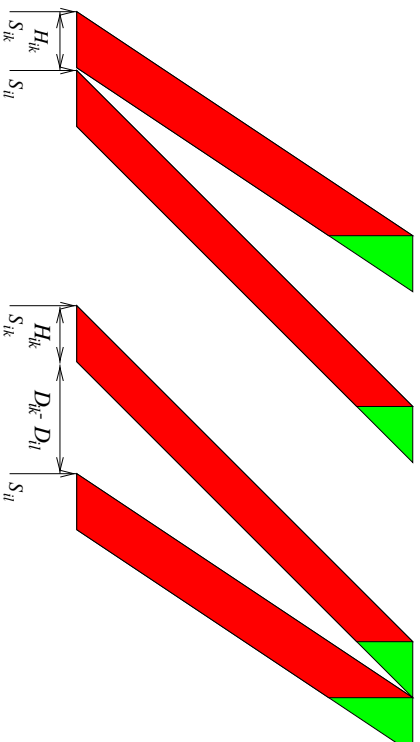
1. If trips  $k$  and  $l$  traverse  $i$  in opposite directions

$$(S_{ik}, D_{ik}) \diamond (S_{il}, D_{il})$$



2. If  $k$  and  $l$  traverse  $i$  in the same direction

$$(S_{ik}, \max(H_{ik}, H_{ik} + D_{ik} - D_{il})) \diamond (S_{il}, \max(H_{il}, H_{il} + D_{il} - D_{ik}))$$

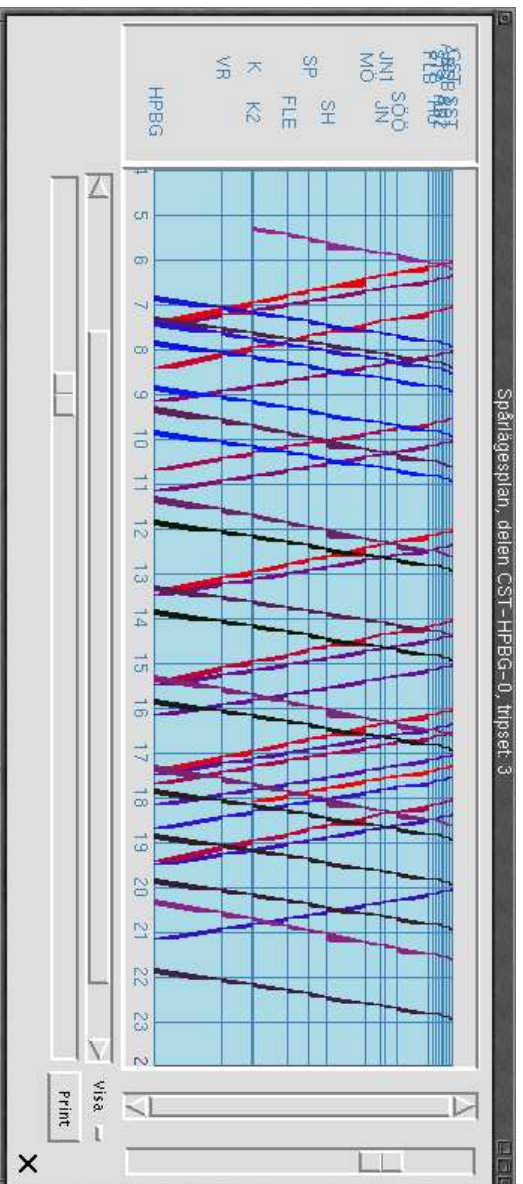


**Note** The max expressions are used to analyse the cases when speeds are different.

## Global constraint formulation

- Use version of serialIze that handles unique setup time between any pair of tasks
- Let task duration be the minimum headway between any two task
- Use the setup parameters to enforce larger time distances between trains with different speeds and travelling in opposite directions
- This model scales to problems of the size of the goods traffic in northern Sweden

## A completely determined timetable



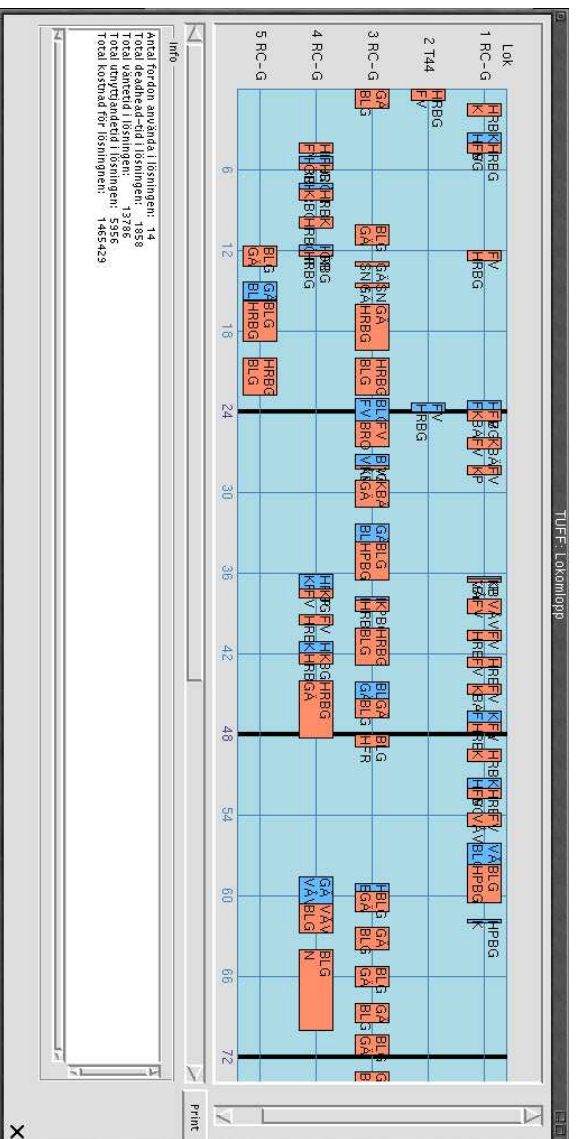
Track time slot schedule

## Engine circuits

- The problem of determining engine circuits is solved by determining, for each engine, a closed path through the network such that each trip is supplied with the type and number of engines it needs
- This includes determining certain passive transports of engines from the arrival location of one trip to the departure location of the next trip in its circuit
- This can be done in two ways:
  - The engine runs with an already scheduled trip — better
  - The engine runs alone as a new trip (that has to be scheduled separately on free time slots) — worse

## Solution of the engine circuit problem

- This problem can be solved by traditional optimisation methods (Network-SIMPLEX) if the timetable has been predetermined
- If not the problem becomes considerably more difficult
- This is because in this case the generation of the vehicle circuit has in addition to partially solve the scheduling problem
- This combined problem is a particularly difficult instance a “travelling sales man” problem
- Recently introduced heuristic methods works fairly well for problems of moderate size



## Engine circuits

- Finally one has to generate personnel circuits
- This problem is superficially very similar to the vehicle problem
- Unfortunately it also contains a large number of complicating additional requirements e.g.
  - People would not like to spend more than a limited time away from home
  - People can work only so long before taking a break
  - etc

## Personnel circuits

- Finally one has to generate personnel circuits
- This problem is superficially very similar to the vehicle problem
- Unfortunately it also contains a large number of complicating additional requirements e.g.
  - People would not like to spend more than a limited time away from home
  - People can work only so long before taking a break
  - etc

## Coordination

- The ordering of the planning activities suggested by the waterfall model is traditionally used in the rail transport industry
- Unfortunately it is suboptimal even if the solution of each of the sub-problems is optimal
  - To see this consider the following example:
    - \* Assume that a specification contains slack in the departure and arrival times for each trip
    - \* Assume that a particular solution to the scheduling problem fix some departures to just before arrivals at the same location
    - This means that the same engine (and personnel) cannot be used to service certain sequences of trips

## An alternative sequence of subproblem solution

- Solve (partially) engine (and maybe personnel) circuit problems before track allocation
  - Unfortunately this makes it much more difficult to find efficient solutions to the engine (and personnel) problems
  - New methods (heuristics, iterative interactive planning ) has to be employed
- We have built an interactive tool for analysis and coordination of these tasks
- The methods we employ in this tool can be thought of as the kernel and a set of tools for implementing general decision support tools

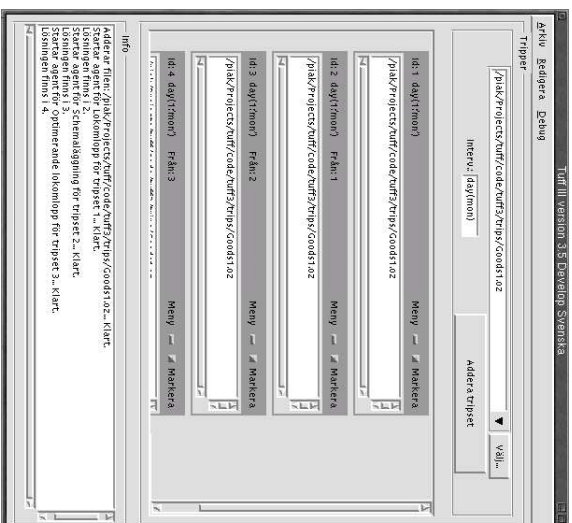


Figure 5: Graphical user interface for a coordination component