Applying a Distributed CLP Platform to a Workforce Management Problem

Ilias Sakellariou, Fotios Kokkoras and Ioannis Vlahavas
Department of Informatics, Aristotle University of Thessaloniki, 54124 Thessaloniki Greece
Email:{iliass,kokkoras,vlahavas}@csd.auth.gr

***Abstract*— The work presented in this paper concerns the application of CSPCONS, a distributed constraint logic programming platform to a workforce management problem, namely the BT-250-118 problem instance. The latter is a well-studied problem instance in which the requirement is to create sequences of job locations for the technicians to visit (tours), so as to serve as many jobs as possible, minimizing at the same time the travel duration. CSPCONS is a logic programming platform that supports program execution over multiple Prolog processes with constraints. It offers channel-based communicating processes and TCP/IP communication and is based on the CSP model introduced by Hoare. This paper demonstrates its applicability to such complex Distributed Constraint Satisfaction problems.**

I. Introduction

Constraint programming has undoubtedly proved to be a suitable platform for tackling large combinatorial problems. It has been applied with success to a number of industrial applications as for example scheduling, resource allocation, etc. Unfortunately, even with the most advanced techniques, solving such problems is both space and time costly.

However, finding solutions for large problems is sometimes most advantageously carried out as the joint responsibility of multiple agents. For example, a multiple agent approach can be essential when the areas of expertise relevant for solving the problem do not reside in any single agent but are found among multiple heterogeneous agents. Alternatively, a group of identical or similar agents might be employed for the purpose of finding a solution faster than it can be accomplished by a single problem-solver. Such a multi-agent system might be one in which the agents all work in parallel on the entire problem and share hints with each other, or it might be one in which the problem is subdivided among the agents. In the latter case, each agent works on solving its part of the entire problem, within some framework of interaction with the other agents, and eventually the subproblem solutions are recombined in some way to solve the overall problem.

When a problem is subdivided among multiple agents, the way in which the agents find, share and use partial results can greatly affect the overall efficiency of the problem solving effort, either positively of negatively. If all communication of subproblem solution is postponed to the termination of problem solving, incompatibilities might force some or all of the agents to redo work already done. Had the agents communicated their partial results earlier, they might have used each others' partial solutions to direct their efforts, so that upon completion the final solution would be globally consistent.

On the other hand, early communication of subproblem solutions or partial solutions can steer other agents in a counterproductive direction. If a communicated solution that is ultimately inconsistent with any global solution is incorporated by another agent and used to guide its problem solving, that agent can spend a lot of time until it discovers that there is no solution in that direction. Alternatively, a consistent solution might be found, but it might be of lower overall quality than if the agent had worked more independently earlier.

Workforce scheduling problems belong to this class. In the general case, such a problem can be considered as a multi-TCTSP (Time Constrained Traveling Salesman Problem). A TSP involves selection of the shortest route among a number of locations to be visited, subject to the constraints that the person must return to the starting point at the end of his/her journey, and can visit each location only once. A time constrained TSP involves additional time related constraints that need to be satisfied (e.g. some jobs must be done in the afternoon).

CSPCONS is a logic programming platform for building multi agent systems. It is an extension of the Communicating Sequential Prolog II (CSP-II), a version of Prolog that is based on the notion of communicating sequential processes. CSPCONS supports independent CLP processes each having its own constraint store that communicate through message exchange over channels, both between processes that reside in the same host and on different hosts over TCP/IP networks. The system can be easily extended to support new algorithms for constraint solving thus permitting the addition of new constraint domains. The current version includes a library for constraint satisfaction over finite domains (FD) and a linear solver. The combination of the channel based communication and constraint satisfaction, all under the logic programming framework, offers a powerful platform for the rapid implementation of any agent based CSP application.

This paper presents the application of CSPCONS to a workforce management problem instance presented by BT. It involves finding an assignment of sequences of repair jobs to technicians, each technician belonging to some base. The solution must assign as many jobs as possible and keep the travel duration to a minimum. The approach adopted to solve the problem can be applied to all problems that involve allocating technicians to jobs that are distributed over a geographic area.

The rest of the paper is organized as follows. Section II briefly presents related work in the field of distributed constraint satisfaction. The CSPCONS platform is briefly presented in Section III. Section IV describes in detail the workforce management problem instance and the approach we have followed to solve it. Finally conclusions and future work are stated in section V.

II. Distributed Constraint Satisfaction Problems

A constraint satisfaction problem (CSP) consists of finding an assignment of values from a given domain to a set of variables, such that a set of constraints on the variables is satisfied. More formally a constraint satisfaction problem consists of:

- a set of variables X $x_1, x_2, ..., x_n$
- a set of domains D each associated with a variable $D_1, D_2, ..., D_n$
- a set of constraints that impose restrictions on the values that the variables can take. A constraint $R_k(x_{k1}, ..., x(_{km}), m < n$ can be defined by a predicate on the Cartesian product $D_{k1} \times D_{k1} \times ... \times D_{k1}$ and is true on a subset of this product.

A distributed constraint satisfaction problem is a CSP in which the variables/constraints are distributed over some network of agents. Agents are constraint solvers which cooperate to solve the original problem. The need for distributed constraint programming applications derives mainly from two facts: a) more efficient implementations, in terms of execution time, can be achieved by decomposing the original problem into subproblems and b) representing problems that are naturally distributed is significantly facilitated, as for example production planning in a factory in which independent departments must meet their local constraints and at the same time co-operate to achieve global constraints.

A number of approaches have been reported to the literature that address the issue of building distributed constraint programming applications. In the sequel we will restrict our presentation to systems that belong to the logic programming framework that are closely related with the CSPCONS approach.

The CIAO language[1] follows a blackboard architecture that is described in [2]. CIAO is a system based in Prolog extended with constraints, parallelism and concurrency. The distributed execution facilities are based on the Linda library for implementing communication between processing units (referred to as workers), i.e. it adopts a blackboard architecture and the use of attributed variables[3].

A different approach to solving CSP problems in parallel has been proposed by Tong and Leung in [4]. Their model, called Firebird, is based on an extension of the Andorra principle and is an attempt to build a concurrent constraint logic programming system on a massively parallel SIMD computer, that will exploit OR-Parallelism. In Firebird execution interleaves between *indeterministic derivation steps* that consist of guard tests, commitment and spawning in the same manner as committed-choice languages and *non-deterministic derivation steps* which consist of setting up a choice point on a domain variable and attempting all the alternative values in its domain in an OR-parallel manner.

To our knowledge no language that combines communicating sequential processes, to the extent that CSPCONS does, with constraints has been proposed in the literature till now.

III. The CSPCONS Platform

The CSPCONS [5] platform is based on the CSP-II , the communicating Sequential Prolog II that is being developed since 1995 [6],[7]. CSP-II is an excellent platform for building any agent based distributed logic programming applications since it offers advanced communication facilities all under the logic programming framework. The platform has already been successfully employed in the development of a distributed expert system for the management of a TCP/IP based WAN [8].

CSPCONS is in fact an extension of CSP-II that inherits all its advanced features and at the same time supports constraint logic programming. Both platforms have extended features like modularity, multitasking, real-time programming and of course network communication.

The main feature of the CSP-II and the CSPCONS systems is that they support the communicating sequential process [9] programming methodology in a Prolog environment. Processes run in parallel and communication between them is achieved through message passing over channels. This process-based model allows an elegant implementation of any parallel and distributed algorithms.

The channel-based communication has been extended with networking capabilities over the TCP/IP protocol, thus providing the ability to establish connections between applications residing in different hosts across the Internet. Furthermore, under this schema a plethora of features are provided such as communication with foreign applications, an interface to relational data base systems, real-time programming methods like cyclic behavior, reaction to predefined events, timed interrupts, etc.

A. CSPCONS *Processes*

CSPCONS processes are defined as the execution flow of a Prolog goal and every process has its own Prolog execution environment and dynamic database. Thus the progress of a process is independent of the execution of other processes. This separation of dynamic databases ensures that CSPCONS processes may influence on each other only by the specified communication techniques, i.e. channels, events and interrupts, or through external objects like files.

In general, each CSPCONS process can have active instances of several different solvers, as for example an FD and a Linear solver. However the set of constraints and domain variables maintained by instances of a solver that belong to different processes are independent of each other, resulting to a communicating sequential CLP system. On a single processor machine a time-sharing scheduler controls the concurrent processes.

This process independence makes the platform a excellent tool for prototyping any agent based application: agent societies are developed as communicating CSPCONS processes, can be fully tested on a single host and then with minor extensions to the code can be ported to their target environment.

Processes are identified by a unique system-wide symbolic name. Two kinds of processes are provided:

- self-driven or normal processes, which is the most usual kind.
- event-driven or real time processes.

A *self driven* process is characterized by its (Prolog) goal; after its creation, it will begin the execution of this goal. The non-fatal termination of a self-driven process is determined by the termination of its goal. At the moment of its termination the process disappears from the CSPCONS system and will never reappear.

A *real time* process is characterized by one goal for the initialization, one goal for the event handling and by the description of the events that trigger its execution. The initialization goal is executed once and provides the means for performing any necessary setup actions. After the successful termination of the initializing goal the process switches to a cyclic behavior. From that moment on it is controlled by the incoming events. For every real time process, the incoming events are gathered in a separate first-in-first-out input queue, from which the process consumes them by initiating its event-handling goal. The number of events that real time processes can be triggered for is unlimited. The successful termination of a process is signaled by the failure of its event-handling goal. Such termination is considered as regular; it does not affect the overall success or failure of the application.

Inter-process communication is achieved by synchronous messages or by event passing. Messages are passed through *communication channels*. A *message* can be any Prolog term except a single unbound variable, however compound terms containing unbound variables are allowed. Communication channels act as system-wide available resources, identified by unique names and may appear and disappear dynamically during the program's lifetime. A channel implements an one way communication between two processes. In such a connection, one process has the sending end of the channel and the other the receiving end. The total number of channels in the system and the number of the channels a process can be connected to are unlimited.

As stated, *events* serve for triggering real time processes and are also identified by system-wide unique names. They can be generated explicitly by built-in predicates or implicitly by the internal clock of the CSPCONS scheduler. The latter allows to invoke execution of the real-time process in specific time intervals. The number of the available events in a program is unlimited. It should be noted that every occurrence of an event may have an optional data argument that can be used to provide some additional information. The event data is an arbitrary Prolog term, except the case of a single unbound variable.

Finally it should be noted that processes can backtrack, however communication is not backtrackable.

B. TCP/IP Communication

As a natural extension of the original inter-process channel concept, the external communication conceptually consists of message streams. In order to facilitate speed-up of external communication, asynchronous message passing is introduced as an option. The *send* operation in this case remains blocking but the condition for continuing execution is the availability of sufficient buffer space instead of the commencement of the matching *receive* operation.

For the Prolog programmer the communication environment appears as a homogeneous address space (community) in which all fellow applications (partners) are accessed via channel messages. A separate mechanism is introduced for connecting channels to other CSPCONS applications. Two notions are introduced in this mechanism: the port and the connection.

A *port* represents an incoming message substream. This entity should not be confused with the normal TCP/IP port. A CSPCONS port is the entry point of all incoming messages for the local application. It is explicitly created by a corresponding predicate and a local channel is associated with it at the time of its creation. The application receives all messages through that channel. A parameter set during port creation determines the size of the message buffer so that asynchronous communication can take place.

A *connection* is the representation of an outgoing message stream. It is also explicitly created by the programmer and is associated with a partner's port. There it forwards all outgoing messages that it receives from a specific local channel of the sender application. All previous information is defined at the creation of the connection, including a parameter indicating the number of messages stored in the connection buffer.

In order to be able to communicate with a partner, a configuration process has to be performed using a special built-in predicate. Though this, all necessary network information of the partner is defined, i.e. its name, port, IP address or host-name, IP port it listens to, etc. Although this operation requires detailed knowledge of the partner's network information, it provides a more versatile connection schema. We are currently considering the idea to introduce some sort of naming service in a future version, however this will not require modifications of the current communication model, since it will be added in the form of a simple Prolog library.

At any given time the status of each agent in the community is known to all fellow agents via an alert mechanism: changes in the status of an agent trigger network related events to the fellow agents participating in the community, thus informing them of the change that took place, as for example the non-availability of an agent.

A CSPCONS application can also establish communication with a non-Prolog application through an appropriate mediator, that handles all data and protocol conversions. Currently

CSPCONS supports an ASCII mediator for plain text communication and one for communication with a specific network management platform (HNMS).

C. Constraints in CSPCONS

The CSPCONS system consists of two main subsystems: the *solver* and the *core*. The solver is responsible for maintaining the constraint store and performing any constraint related tasks, i.e. is responsible for storing domain variables and the set of constraints as well as for constraint propagation. The core is the extended CSP-II system that keeps track of the active instances of the different solvers, dispatches requests originated by the Prolog program to the appropriate solver instance, and performs other system-related tasks, including ll normal Prolog predicate calls.

The design aim behind such a model was to allow the introduction of new constraint handling facilities easily. The solver is in fact a C linkable library that implements a set of function calls defined by the CLP-Interface of CSPCONS .

The model offers independence of the code concerning constraint handling and provides the means to easily extend the system to support any constraint domain and any algorithm. Currently CSPCONS supports a finite domain solver and a linear equations-disequations solver.

1) The Finite Domain Solver: Finite domains (FD) constraints are one of the most studied areas of constraint programming. This is not surprising since 95% of industrial applications employ FD constraints, undoubted evidence that a large variety of problems that can be modeled using the specific domain. A large number of algorithms have been proposed in the literature that aim to remove a inconsistent values from domain variables in order to prune the search space and allow an efficient solution of large combinatorial problems.

The CSPCONS FD solver was based on the AC-3 [10] algorithm. Although the latter is not considered state of the art, it was selected due to its simplicity.

Currently the solver supports constraints of the form: $x \in \{n_1, n_2, .., n_m\}$ and $exp_1\ R\ exp_2$ where $\{n_1, n_2, .., n_m\}$ is a set of natural numbers, $R \in \{=, \neq, <, >, \geq, \leq\}$ and exp_1, exp_2 are linear expressions on constraint variables. All constraints are posted through the `clp_constaint/1` predicate as shown in the following examples:

```
clp_constraint([X in [1..9],Y in [1..9]]),
clp_constraint([3*X < 2*Y +10]),
```

All unary and binary constraints are handled internally by the consistency algorithm. Higher arity constraints are handled by a bounds consistency algorithm. A set of predicates for labeling including labeling with heuristics, such as the fail-first principle, most-constraint principle, etc is also available.

It should be noted that the implementation has been tested on a variety of benchmarks, including the well-known NQueens, Golomb-rulers, cryptarithmentic and alpha problems and has shown adequate performance.

IV. THE BT WORKFORCE SCHEDULING PROBLEM

In the BT workforce scheduling problem [11] (dataset RD-250-118 [12]), the requirement is to create sequences of job locations for the technicians to visit (tours), so as to serve as many jobs as possible, minimizing at the same time the travel duration.

More specifically (Fig 1), there are 11 BT bases at different locations each one having a number of technicians. There is a total of 118 technicians, each one working on a certain time frame and having a skill factor that affects the time it takes for the technician to accomplish a job. There is a total of 250 jobs, each one located at a certain location and requiring specific time for an average technician to accomplish. In addition, most jobs are constrained over time, that is, some of them are morning jobs, some other must be first in a tour, etc. Finally, for each job, there is a list of engineers associated with it that indicates which of them are qualified to do the job.

The travel time between two locations with coordinates (X_1, Y_1) and (X_2, Y_2) for a technician $i$is given by the following function:

$$TTime_{12} = \frac{|X_1 - X_2| + |Y_1 - Y_2|/2}{8}$$

if $|X_1 - X_2| > |Y_1 - Y_2|$, otherwise the distance is given by the function:

$$TTime_{12} = \frac{|Y_1 - Y_2| + |X_1 - X_2|/2}{8}$$

The total quality of any given solution to the above problem, is calculated by the following cost function:

$$Cost = \sum_{i=1}^{Techs} TTime_i + \sum_{j=1}^{Jobs} (Dur_j + Penalty) \times Flag_i$$

where $Techs$ is the total number of technicians, $TTime_i$ is the total travel time of engineer i, Jobs is the total number of jobs, Dur_j is the duration of job j, $Penalty$ is a constant set to 60 and $Flag_j$ is set to 0 if job j is allocated to an engineer; 1 otherwise. Strictly speaking, the aim of the problem is to minimize the cost function.

A. Related Work

The approaches that have been used so far to tackle the stated problem include simulated annealing [13], genetic algorithms [14], fast local search and guided local search [15], CLP [11], [16], and finally, CLP with distributed patching techniques [17]. [11] uses a constraint-based tour generation algorithm and a heuristic schedule repair technique. Repair techniques are also used in [14]. The fast local search algorithm in [15] helps to improve the efficiency of hill climbing. The guided local search helps local search to escape local optima. Approach [16] avoids a full search and uses heuristics for the generation of the tours, while in [17], the problem is partitioned into subproblems that are solved independently to get a first solution and then distributed patching techniques are used to further improve this solution.

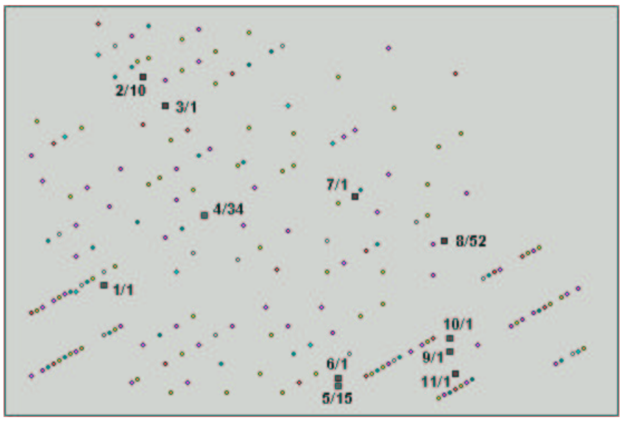

Fig. 1. Spatial visualization of Jobs, Bases and Personnel

TABLE I
WORKFORCE AND SERVICE CAPACITY OF EACH BASE.

BaseID(s)	No of Technicians	Service Capacity (min)
1,3,6,7,9,10,11	1	480
2	10	4800
4	34	16320
5	15	7200
8	52	24960

B. Solving Approach

In order to solve the above problem a three phase procedure was adapted. The first phase concerns partitioning the problem resulting to a number of subproblems of less complexity. This clustering phase is followed by a solving phase in which CSPCONS agents find near-optimal solutions according to the cost function. Then a purely distributed patching technique that follows a distributed bidding strategy was employed to find the final job assignments. Each agent is modelled as a CSPCONS process and all communication is implemented using the communication facilities of the system.

1) Clustering Phase: The idea behind clustering is to partition the problem (data and search space) into sub-problems of less complexity. As far as these sub-problems are independent, partitioning reduces dramatically the execution time needed to solve them, as well. This time saving allows us to use algorithms that are impractical in the case of the whole problem due to their complexity. The overhead is that, when the sub-problems are related with constraints on elements belonging to different sub-problems, additional effort is required to resolve conflicts.

For the case of the BT workforce problem we decided to use a large grained partitioning that is base oriented. This reflects the natural organisation of BT's services that are also base-oriented. Thus, we decided to create 11 sub-problems and assign each of them to one agent/base. Moreover, any idea about uniformly distributing the jobs to bases was immediately rejected because the service capacity varies significantly among bases (see table I).

As a result we decided to use a clustering method that shares the working time (i.e. the jobs) among bases in a way proportional to the ratio of the total service capacity of each base. We also considered the various job types rejecting this way jobs that were impossible to service due to lack of qualified personnel.

A ripple like clustering method was developed in which bases obtain candidate jobs from a common pool in a round-robin fashion. Each agent (base) has an ordering of all the jobs according to their distance from itself. In each step the agent that has the control gets the closest free jobs from the pool. The number of jobs each agent gets in a single step is proportional to $\ln(T)$, where T is the number of the technicians that belong to the base. However small bases ($T \leq 3$) get always two jobs in each step. Additionally, since small bases are next to large ones, the former are allowed to select jobs first. Obviously, as soon as a base covers its service capacity it stops selecting jobs. The overall method terminates when all jobs have been assigned to some base.

2) Solving Phase: The problem was modeled using finite domain constraints following the same approach that is reported in [16]. To achieve better results in this phase, the "closest first" heuristic [16] was used for labeling jobs.

According to the heuristic when a job is going to be labeled, the candidate engineers are organised into two groups. Those that are in a good direction related to the job and those that are in a bad direction. An engineer belongs to the good direction group if any of the following conditions hold:

- the engineer is currently idle;
- the engineer has been assigned a job that is in the same direction as the labelling job;
- the engineer has already been assigned more than one job, among which are at least two jobs in the same direction as the labelling job.

Two jobs, located at places J_1 and J_2, are in the same direction if the angle $(J_1, Base, J_2)$ is less than 45 degrees. It should be pointed out that this is the best heuristic in terms of the cost function but is a bit expensive due to the direction calculation.

As soon as all agents have completed the solving phase the patching phase is initiated.

3) Bidding Strategy: The motivation behind our approach in this phase is to try and fit unscheduled jobs to *gaps*, i.e. time frames in tours in which technicians are either on a long transition or idle. A gap are filled with an appropriate job resulting in a *patch*. Thus, during this phase each $Agent_i$ does:

1) Sends/receives the unscheduled jobs to/from the other agents.
2) For each $Tech_{ij}$ generates all Gap_{ijk} permutations based on the current tour (named $Tour_{ij}$) of $Tech_{ij}$.
3) For each Gap_{ijk} generates all possible patches (named $Patch_{ijkl}$) using jobs from all of the unscheduled jobs, annotated with the gain to the total cost function obtained by serving this job.
4) Sends/receives all patches $Patch_{ijkl}$ to/from the other agents (bids)

5) Filters out the whole set of patches, removing those that would be better served by another agent according to the broadcasted gain values.
6) Uses the remaining of the patches $Patch_{ijkl}$ to update (patch) the tours $Tour_{ij}$

It should be noted that unscheduled jobs that correspond to patches are inserted to the tour without altering the schedule found in the solving phase. The above procedure continues until no further optimization can be performed. In our case two optimization cycles took place.

C. Results

The system was implemented on a Sun E450 machine running Solaris. The results obtained are shown in table II. The table shows the characteristics of the first solution and the final solution indicating the contribution of each bidding cycle.

TABLE II
SOLUTION DETAILS BEFORE AND AFTER BIDDING

	First Solution	**Final Solution**
Scheduled Jobs	165	190(165+23+2)
Active Techs	63	67(63+4+0)
Total Cost	24912	21948

Figure 2 presents the technician tours that consist the final solution to the problem. Clusters generated during the first phase are indicated also.

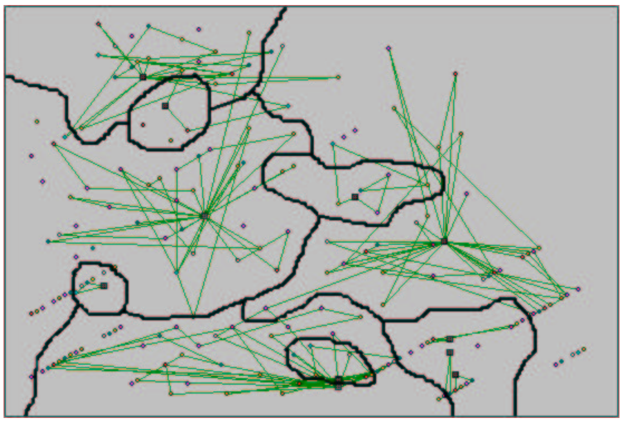

Fig. 2. Final Solution and Clusters Visualized over the Problem Area

V. Conclusions and Future Work

As demonstrated by the impementation of workforce management problem the CSPCONS system offers a suitable platform for developing complex DCSP applications. The system facilities were more than adequate not only to model the problem but also to implement all communication between the agents in an elegant declarative way. Although other platforms also provide communication primitives, those of CSPCONS are high level and significantly reduce the efford required by the programmer, allowing him to focus on the problem itself.

We are currently improving the bidding optimization strategy so that only close to the base patches are generated. This will decrease the effort of the patching algorithm without decreasing solution quality.

It is also in our plans to add problem specific algorithms as extensions to the current FD solver and offer them as built-in predicates, so that programming efford for this class of problems will be further reduced.

References

[1] M. Hermenegildo, F. Bueno, D. Cabeza, M. G. de la Banda, P. Lopez, and G. Puebla, *The CIAO Multi-Dialect Compiler and System: An Experimentation Workbench for Future (C)LP Systems*. Nova Science, April 1999, pp. 65–85.

[2] D. Cabeza and M.Hermenegildo, “Distributed Concurrent Constraint Execution in the CIAO System,” in *Proceedings of the 1995 COMPULOG-NET Workshop on Parallelism and Implementation Technologies, U. Utrecht / T.U. Madrid*, September 1995.

[3] M. Hermenegildo, D. Cabeza, and M. Carro, “Using Attributed Variables in the Implementation of Concurrent and Parallel Logic Programming Systems,” in *Proceedings of the 12th International Conference on Logic Programming*, L. Sterling, Ed. Cambridge: MIT Press, June 13–18 1995, pp. 631–646.

[4] B.-M. Tong and H.-F. Leung, “Data-parallel concurrent constraint programming,” *The Journal of Logic Programming*, vol. 35, pp. 103–150, 1998.

[5] I. P. Vlahavas, I. Sakellariou, I. Futo, Z. Pasztor, and J. Szeredi, “C SPCONS: A Communicating Sequential Prolog with constraints,” in *Methods and Applications of Artificial Intelligence, Procs of the 2nd Hellenic Conference on AI, SETN 2002*, ser. Lecture Notes in Computer Science, vol. 2308. Springer, 2002, pp. 72–84.

[6] I. Futo, “Prolog with Communicating Processes: From T-Prolog to CSR-Prolog,” in *Proceedings of the 10th International Conference on Logic Programming*, D. Warren, Ed. The MIT Press, 1993, pp. 3–17.

[7] I. Futo, “A Distributed Network Prolog System,” in *Proceedings of the 20th International Conference on Information Technology Interfaces, ITI 99*, 1998, pp. 613–618.

[8] I. Vlahavas, N. Bassiliades, I. Sakellariou, M. Molina, S. Ossowskia, I. Futo, J. S. Zoltan Pasztor, I. Velbitskiyi, S. Yershov, and I. Netesin, “ExperNet: An Intelligent multi-agent system for wan management,” *IEEE Intelligent Systems*, vol. 17, no. 1, pp. 62–72, 2002.

[9] C. A. R. Hoare, “Communicating Sequential Processes,” *Communications of the ACM*, vol. 21, no. 8, pp. 666–677, Aug. 1978.

[10] A. K. Mackworth, “Consistency in Networks of Relations,” *Artificial Intelligence*, vol. 8, no. 1, pp. 99–118, 1977.

[11] N. Azarmi and W. Abdul-Hameed, “Workforce scheduling with constraint logic programming,” *British Telecom Technology Journal, British Telecom Laboratories, Ipswich*, vol. 13, no. 1, 1995.

[12] “RD-250-118 Data Set,” British Telecom Laboratories.

[13] S. Baker, “Applying simulated annealing to the workforce management problem,” British Telecom Laboratories, Ipswich, Tech. Rep., 1993.

[14] C. Muller, E. Magill, and D. Smith, “Distributed genetic algorithms for resource allocation,” Strathclyde University, Glasgow, Tech. Rep., 1993.

[15] E. Tsang and C. Voudouris, “Fast local search and guided local search and their application to british telecom’s workforce scheduling problem,” Department of Computer Science, University of Essex, Colchester, Tech. Rep. CSM-246, 1995.

[16] R. Yang, “Solving the workforce management problem with constraint programming,” in *Proceedings of Practical Applications of Constraint Technology*, 1996.

[17] F. Kokkoras and S. Gregory, “D-WMS: Distributed workforce management using CLP,” in *Proceedings of the 4th International Conference on the Practical Application of Constraint Technology, PACT 98*. Practical Application Company Ltd., March 1998, pp. 129–146.