Constraint Logic Programming for Fault-Tolerant Distributed Systems

T. Creemers, J. Riera**, and E. N. Tourouta***

** Institute of Problems of Data Transmission, Russian Academy of Sciences, Moscow, Russia*

*** Institut de Robótica i Informática Industrial (UPC-CSIC), Barcelona, Spain*

Abstract—This paper presents key notions of Constraint Logic Programming (CLP), which is a young programming paradigm oriented toward solving difficult discrete highly combinatorial problems by making active use of constraints on the basis of mechanisms of Logic Programming. Being the subject of intensive research all over the world, CLP has already been used successfully in a large variety of application areas. As one of the important applications where CLP demonstrates its potential, we propose CLP-based procedures of solving the problems of optimal resource and task allocation at the stages of design and operation of Fault-Tolerant Distributed Technical Systems.

INTRODUCTION

Constraint Logic Programming (CLP) is a relatively new programming paradigm that integrates the main principles of Logic Programming [1] with active use of constraints on the basis of Constraint Solving Techniques [2]. It forms an elegant and attractive tool for solving difficult discrete combinatorial problems as found in scheduling, resource allocation, placement problems, routing, configuration, financial and production planning, etc. Despite its young age, it is a subject of intense research all over the world, while having already shown its potential in a large number of fielded, industrial applications [3]. CLP-technology, owing to its power to handle these complicated problems, can be effectively applied for developing methods of high fault-tolerance achievement for modern complex technical systems at the stages of their design and operation. Highly dimensional discrete optimization problems are typical for these areas.

Today's industrial systems, made up of a great many Operational Units (OUs), are more and more complex and demanding. Improper operation of such a system caused by failures of OUs may not only result in reduced efficiency of a system with important associated cost but also in accidents with catastrophic consequences. Therefore, by now, ensuring an extremely high level of dependability, survivability, and safety of industrial systems has become unconditional demand. The main way to address this problem is creating fault-tolerant systems, i.e., those possessing the ability to perform their functions correctly (perhaps at a degraded but acceptable level of operation quality) in the case of failures of a certain number of components. The property of a system fault-tolerance is the basis for ensuring its dependability, survivability, and safety.

One of the promising approaches to assure a high degree of a complex system fault-tolerance in optimal way is based on redistribution of a system tasks over non-faulty OUs using reconfiguration of a system structure and fault recovery. It permits one to use internal, "natural" redundancy inherent to modern modular reprogrammable systems and, as a consequence, can prevent an extremely large overhead to reach the required high level of fault-tolerance. This trend appears to be particularly promising for systems composed of multiple reprogrammable units, those possessing potential ability to reassign the tasks of faulty OUs to non-faulty ones using natural redundant resources of OUs. The vast majority of modern industrial systems belong to this class.

This approach was primarily adopted to an extent in some fault-tolerant computing systems [4–6] and was considered theoretically in [7, 8]. However, these works and projects do not touch upon some important aspects (task execution time ordering, concurrency and interactions of tasks, functional degradation; i.e., the possibility for a system to reject some tasks, etc.) and do not address the problem of optimized Task Redistribution (TR), i.e., such that would guarantee a system's functioning with a predetermined level of fault-tolerance at the optimized figures of merit of its operation quality accounting for cost, resource, and real-time constraints.

The general concept of *Fault-Tolerance oriented Optimal Static Task Redistribution (OSTR/FT)* and a set of formal methods to implement this concept for a variety of distributed systems have been proposed in [9, 10]. The concept calls for arranging a system tasks execution in such a way that, in the event of occurring faults of some OUs, the tasks can be reassigned in an optimal way for execution into the remaining non-

faulty OUs, thereby reactivating proper functioning of a system. This goal is achieved by means of creating *optimal static Fault-Tolerant Task Allocation (FT-TAll)* which results in task redundancy: scheduling the possibility to execute the same task in different OUs. The problem of finding such FT-TAll is a combinatorial discrete optimization problem under a number of different constraints; i.e., one which is the most suitable for solving by means of CLP.

This concept may be adopted for a distributed system of whatever type or nature (technical, economical, administrative, etc.) comprised of multiple operational units and devoted to performing a predetermined set of tasks, provided that any of these units is capable of executing any task from the given set or from some subset of it.

This paper presents a short overview of the CLP research area and its main features as a powerful tool to solve the problems of distributed systems fault-tolerance achievement at the stages of their design and operation. The key notions of the general OSTR/FT concept as an advanced approach to ensure a high level of fault-tolerance of a Distributed System (DS) during its design are given, and the applicability for technical DS is demonstrated by considering two typical representatives, namely, *Distributed Computing Control Systems (DCCS) and Flexible Manufacturing Systems (FMS)*. Implementation of one of the OSTR/FT techniques on the basis of CLP for creating fault-tolerant DCCS is described using the application example. Effective application of CLP for sustaining fault-tolerant operation of a large technical system is demonstrated by solving the complicated problem of maintenance tasks scheduling on an electric power distribution network.

1. CONSTRAINT LOGIC PROGRAMMING (CLP)

Constraint Logic Programming as a scientific field related to a specific concept for solving discrete highly combinatorial problems and particular programming paradigm results from the merging of two trends of research, namely, Logic Programming and Constraint Solving. The former is the well-known declarative programming paradigm implemented in the programming language Prolog [1]. Logic programming allows separating the specification part of the problem (what needs to be solved) from the procedural part (how to solve it). The procedural part is taken care of by a generic, built-in resolution mechanism, while to a large extent the specification part (usually in Prolog-like syntax) is the only major aspect the application programmer needs to care about. However, one of the most serious limitations of logic programming lies in its computation rule resulting in a "generate and test" procedure and, consequently, in backtracking with the well-known performance problem. It is the root of Prolog's inefficiency.

Constraint Logic Programming originated from the attempt to overcome the difficulties of logic programming by enhancing Prolog-type languages with constraint solving mechanisms. Constraint manipulation and propagation have been studied in Artificial Intelligence in the late 1970s and early 1980s to improve search procedure efficiency [11–13]. A set of techniques like local-value propagation, data-driven computation, forward checking (to prune the search space), and look ahead summarized under the heading *Consistency Techniques* has been developed for solving constraints. Development of CLP has been strongly influenced by the work on consistency techniques. The use of these techniques for improving the search behavior of a logic programming system has been advocated in [14] where the active use of constraints to prune the search tree in an *a priori* way has been proposed instead of using constraints as passive tests leading to a "generate and test" and standard backtracking behavior. The usage of consistency techniques in CLP is systematically described in [2]. These consistency techniques, combined with the possibility of declaratively expressing constraints from various domains, have resulted in a class of highly efficient CLP-languages with high expressive power [15]. In CLP-languages, the underlying resolution mechanism has been enhanced with constraint-solving techniques, which at any time will enforce consistency between the problem variables.

Research on CLP in Europe was primarily concentrated at the European Computer Industry Research Centre (ECRC) in Munich and has resulted in probably the most widely known CLP languages, CHIP (Constraint Handling in Prolog) [16] and ECLiPSe (ECRC Common Logic Programming System). These languages, based *on local propagation techniques* over variables with *finite domains*, have shown one of the most powerful paradigms and are penetrating most rapidly into various industrial application domains. Basically, in languages like CHIP and ECLiPSe, problem modeling consists of defining a set of problem *decision variables*, each having a finite set of possible, discrete values (*the variable's domain*), and stating the constraints which must hold between these problem variables. *The problem-solving process* then proceeds as follows: on each change in the domain of a variable (i.e., on each *reduction* of the set of possible values), the "*demons*" associated with each constraint the variable is involved in get woken up and a *consistency algorithm* is executed to make all the constraint's variables consistent again; i.e., inconsistent values are removed from other variables' domains. This, in turn, will wake up other constraints' demons and further domain reductions are performed until the "*constraint network*" reaches some kind of stable state (or fixpoint) in which all domains are locally consistent with respect to the constraints. At this point, the logic program can make a choice (e.g., try assigning a value to a variable), and the *constraint propagation* mechanism can go on again as before. If at any point inconsistency is

detected, the process will backtrack to the last choice point made by the logic program and make an alternative choice.

One should note here the substantial reduction in the amount of backtracking needed to reach a solution with respect to classical methods where constraints are used in *a passive "generate-and-test"–like* way. Letting constraints propagate their consequences as much as possible before making any choice is an *active* way of using constraints, reducing drastically the amount of backtracking (or the number of choice points), and pruning the search space *a priori*. In this context, the paradigm is sometimes called "*constrain-and-generate*" to indicate that constraints are used at the beginning and values are only generated afterwards when the search space is reduced maximally.

In the case of optimization problems, the following strategy is implemented on top of the above general scheme. The value of the objective function is represented as a domain variable C. After constraint propagation and making the right choices, an initial solution satisfying all constraints is found. In this solution, the domain variable associated to the objective function, will have been instantiated to some value C_1: the cost of the initial solution. Then, the process is restarted from scratch but with one additional constraint added to its constraint store: $C < C_1$. Forcedly, the solution of this second iteration will have a lower cost, which in turn can be translated into a new constraint. This process continues until in some iteration inconsistency is detected. At that point, one can be sure that the last found solution has the lowest possible cost.

Thus, CHIP provides constraint solvers for finite arithmetic, linear rational, and Boolean domains. Moreover, the user has the possibility to define his own constraints and control their execution. There are several software products based on CHIP technology that are commercially available from the companies COSYTEC, Bull, and ICL. Many features of ECLiPSe are the same as for CHIP, but CHIP's constraint solvers are hard-coded in the language C, whereas ECLiPSe's are written in itself for easier modification. Generalized constraint propagation technique are also used in ECLiPSe.

One of the founding works on CLP was carried out by J. Jaffar and J.L. Lassez at Monash University in Melbourne [17]. They have presented the CLP(X) system, which was later specialized for several computation domains: CLP(R) for real linear arithmetic (at Monash University, IBM Yorktown Heights research facility, and Carnegie Mellon University), CLP(Q) for rational numbers, and CLP(Z) for integers. The CLP-language Prolog-III for the domains of linear rational arithmetic, Boolean terms, and finite strings or lists was developed by A. Colmerauer (one of the fathers of Prolog) at the University of Marseille [18]. Several CLP systems have been developed for different computation domains, such as Trilogy; non-Prolog-based system for integer arithmetic (from "Complete Logic Systems" in Vancouver, Canada); CAL, the first CLP-language for non-linear constraints [19]; and its parallel version GDCC [20] (from ICOT, Tokyo), which are able to operate in the domains of non-linear real equations, Boolean constraints, linear rational arithmetic, and some others.

The CLP scheme was further generalized into the framework of concurrent constraint programming which accommodates all operation on constraints that can be defined as closure operations and therefore significantly extends the scope of CLP languages by enabling issues such as concurrence, control, and extensibility at the language level. This trend results in the language cc(FD), which is a successor to the finite domain part of CHIP. It was applied for solving two practical combinatorial problems, test-pattern generation, and car sequencing [21].

The advantages of CLP technology over traditional techniques and, hence, the reasons for its industrial success lie in the next major points: (a) declarative problem statement much closer to a natural one in which the programmer does not have to care about finding algorithms to solve a problem (instead he can concentrate on what has to be solved); (b) allowing rapid prototyping, which in industry means gaining a lot of money: where traditionally a software project would take one year, a CLP solution will take about two months; and (c) CLP programs are much more maintainable and easier to modify and extend, which is also a major cost factor in software development. Thus, these advantages can be summarized in huge money-saving in software development and maintenance. The potential of CLP-technology lies in its ability to handle exactly those difficult combinatorial problems that are hardest for conventional programming techniques: NP-complete search problems where the time needed for search increases exponentially or worse with the problem size. Constrained search problems like scheduling, allocation, layout, fault diagnosis, and hardware design are typical representatives of this class. The traditional approach for solving these problems requires substantial effort for the development of specialized programs in procedural languages that are hard to maintain, modify, and extend. A large number of constrained search problems have been solved by means of CLP systems resulting in drastically decreased development time while achieving a similar efficiency. A wide range of applications demonstrates flexibility of CLP to adapt to different problem areas [3]. Among the most relevant of these applications, we can mention solutions for circuit design, network problems (cable layout in buildings), warehouse distribution planning, operational control of water-distribution networks, distributed banking [22], personnel assignment problems [23], production planning and scheduling problems (aircraft assembly) [24], workshop scheduling [25], transport problems, database query optimization, military command and control systems, and portfolio management problems. One of

the most recent and important applications, namely, solving a large intricate problem of reconfiguration and maintenance scheduling on power-distribution networks [26], is described in the Section 3 of this paper.

A number of solutions based on CLP technology are being used by large industrial corporations such as Michelin and Dassault, the French national railway authority SNCF, Siemens, the airline companies SAS, Swissair, Cathay Pacific, the Hong Kong international terminal, and the harbor of Singapore. A large number of applications are currently under development in the frame of national and European projects.

2. DISTRIBUTED SYSTEMS FAULT-TOLERANCE ACHIEVEMENT USING CLP-TECHNOLOGY AT THE DESIGN STAGE

2.1. Fault-Tolerance Oriented Optimal Static Task Redistribution (OSTR/FT)

In general, a Distributed System (DS) is treated as a set $\mathbf{H} = \{M_i\}$, $i = 1, \ldots, n$, of n interconnected Operational Units (OUs). It executes a fixed job, i.e., a set of *tasks* $\Omega = \{U_j\}$, $j = 1, \ldots, L$, each unit M_i executing a fixed subset $\mathbf{J}_i \subset \Omega$ of tasks. Permanent faults of OUs may occur with the known failure rates. A *structural state (s-state)* of a system is defined as a binary vector $\mathbf{s}_\nu = \{\sigma_i\}$, $i = 1, \ldots, n$, where $\sigma_i = 0$ if OU M_i is non-faulty, $\sigma_i = 1$ if M_i is faulty; $\mathbf{s}^0 = \{0, 0, \ldots, 0\}$ is the *initial* s-state; and any other s-state $\mathbf{s}_\omega$ is called distorted.

Fault-tolerance of a DS is treated as its capability to execute a given job with acceptable degradation in operation quality provided that, because of OUs faults, a system can transit into any of the distorted s-states of the given set $\mathbf{S}^\omega$ determined by the maximal acceptable number of faulty OUs, let d. Generally, there are two types of system degradation, functional and temporal, estimated through *functional* and *temporal* measures of a system operation quality [9, 27]. The required level of a DS fault-tolerance is achieved by means of rational redistribution of the tasks of a system job in each distorted s-state of the set $\mathbf{S}^\omega$ for executing them by non-faulty OUs. This redistribution is attained by means of *optimized static Fault-Tolerant Task Allocation (FT-TAll)*, which is formed using *basic* and *additional* ones. Task Allocation (TAll) is described by a binary matrix $\mathbf{X} = |x_{ji}|$, $j = 1, \ldots, L$, $i = 1, \ldots, n$, where $x_{ji} = 1$ if a task U_j is allocated in OU M_i; i.e., a program module of U_j is loaded into a storage of OU M_i, otherwise $x_{ji} = 0$. A *Basic TAll* $\mathbf{X}$ is the allocation of the *basic tasks*, i.e., those to be executed in the initial s-state: each task $U_j \in \Omega$ is allocated in one and only one OU. An *Additional TAll* $\mathbf{Y} = |y_{ji}|$; i.e., an allocation of spare passive copies of tasks in OUs, is formed as superposition of *Task Assignment (TA)* plans for all distorted s-states $\mathbf{s}_\omega \in \mathbf{S}^\omega$. For each $\mathbf{s}_\omega$ an optimal TA plan $\mathbf{D}^\omega = \left|d_{ji}^\omega\right|$, $j = 1, \ldots, L$, $i = 1, \ldots, n$, is calculated which assigns the tasks $U_j \in \Omega$ to non-faulty OUs for execution in this s-state ($d_{ji}^\omega = 1$ if a task U_j is assigned for execution in OU M_i in the state $\mathbf{s}_\omega$, $d_{ji}^\omega = 0$ otherwise). Then, in each OU M_i, the copies of *all* the tasks are allocated, those to be executed in accordance with each of the TA plans $\mathbf{D}^\omega$. The global FT-TAll is the result of allocating both the basic tasks and their spare copies and is formed by superposing $\mathbf{X}$ and $\mathbf{Y}$, creating allocation $\mathbf{Z} = |z_{ji}|$, where $z_{ji} = x_{ji} \vee y_{ji}$. In the initial s-state, only the basic tasks are executed. When a system transits into distorted s-state $\mathbf{s}_\omega \in \mathbf{S}^\omega$, the spare copies of the tasks of faulty OUs allocated in non-faulty ones are initiated in accordance with the TA plan for this s-state.

Thus, the problem consists in creating the Optimal or Rational FT-TAll that meets the requirements for a system fault-tolerance (e.g., guarantees proper execution of a system job in each s-state from the given set $\mathbf{S} = \mathbf{s}^0 \cup \mathbf{S}^\omega$) and optimizes the given figures of merit of a system operation quality observing cost and resource restrictions. The rational TA plans and task schedules for all $\mathbf{s}_\omega \in \mathbf{S}^\omega$ also must be created.

We use the next measures to evaluate a DC.

(A). Functional measures:

the *functional capability* of a system in a state $\mathbf{s}_\nu$

$$E_\nu = \sum_{U_j \in \Omega^\nu} b_j, \qquad (2.1)$$

where b_j is the weight of a task U_j, i.e., some value evaluating significance of the task for a system; $\mathbf{\Omega}^\nu$ is the set of the tasks assigned execution by a system in a state $\mathbf{s}_\nu$;

lost efficiency ("losses") of a system in a state $\mathbf{s}_\nu$ (when the weight of a task U_j is naturally estimated by the cost value ρ_j of lost output of a system resulting from non-execution, i.e., "rejection" of this task):

$$R_\nu = \sum_{U_j \in \Omega^{\nu *}} \rho_j, \qquad (2.2)$$

where $\Omega^{\nu *} = \Omega / \Omega^\nu$ is the set of the rejected tasks for the state $\mathbf{s}_\nu$.

(B). Temporal measure: the next vector measure for each s-state $\mathbf{s}_\nu$

$$\mathbf{T}_\Sigma^\nu = \{T_{\Sigma i}^\nu\}, \quad i = 1, \ldots, n, \qquad (2.3)$$

where $T_{\Sigma i}^\nu$ is total execution time of all the tasks of the set Ω_i^ν assigned for execution in the unit M_i in the state $\mathbf{s}_\nu$; i.e.,

$$T_{\Sigma i}^\nu = \sum_{U_j \in \Omega_i^\nu} \tau_j,$$

where τ_j is execution time of the task U_j.

2.2. Distributed Technical Systems

Two typical classes of DS may illustrate the applicability of the OSTR/FT concept combined with CLP-technology for today's technical systems.

Distributed Computing Control Systems (DCCS). At present, computing control systems are crucial integral parts of demanding industrial systems in many application fields. A fault of such a system may result not only in decreased efficiency of the controlled industrial object but also in a catastrophic event. As a rule, these systems are topologically or functionally distributed; composed of multiple *Processing Modules* (*PM*), each containing a processor, individual memory, and appropriate interfaces; and connected through a communication structure (e.g., bus interconnections or local area network). The variety of DCCSs ranges from systems for controlling some local processes or installations up to total-plant hierarchical systems involving several functional levels: technological process control, supervision of individual operating units, and overall plant functioning optimization. Generally, a DCCS executes both real-time and non-real-time jobs composed of the tasks of different types such as periodical, randomly initiated, and strictly interconnected communicating tasks. The OSTR/FT techniques for this variety of jobs are summarized in [27].

The Flexible Manufacturing System (FMS) [28] consists of a number of interconnected *numerically controlled (NC) machining centers* (which process various workpieces simultaneously) and has a hierarchical control structure. An NC-center is made up of several NC-machines and a control block which includes a Computerized Numerical Controller (CNC). Automatic machining of a workpiece is performed by NC-machines that execute typical job shop operations and automatic tool changes in accordance with geometric and technological instructions, which are coded and stored as NC-programs in the CNC. The OSTR/FT concept may be applied not only to the control system of an FMS (in the same manner as for general DCCS) but also to its manufacturing equipment. In fact, an FMS may be considered as a set $\mathbf{H} = \{M_1, \ldots, M_n\}$ of interconnected OUs, i.e., NC-machining centers or NC-machines. Each of these OUs is capable of performing machining operations of the same type, which are treated as tasks of an FMS. A particular set of machining operations performed by each OU M_i is determined by a set of NC-programs assigned for execution in this OU. These programs may be stored either in individual storage of each OU or in common storage of a system. To achieve an FMS fault-tolerance with respect to failures of OUs, i.e., NC-machining centers or NC-machines, OSTR/FT may be employed by means of allocating several spare passive copies of the same NC-program into different OUs. In the case of faults of some OUs, the machining operations initially assigned to them will be initiated in those non-faulty OUs that store the spare copies of the corresponding NC-programs. Implementation of FT-allocation of NC-programs requires not only additional memory in the control part of an FMS but some reorganization in workpiece and tool supply subsystems resulting in resource redundancy, which may be taken into account by means of resource constraints. Time constraints must reflect restrictions for overall time of machining the given workpiece and the reduced number of machines result in increased load for them.

2.3. CLP-Based Procedure of Creating Optimal Fault-Tolerant Task Allocation

The techniques previously proposed for solving the FT-TAll problem [27] are based on various methods of discrete optimization and scheduling, often necessarily powered by heuristics, and in general do not provide optimal, but acceptable, "rational" solutions. The common characteristic of any one of these techniques consists in creating the rational Task Assignment plan (as the solution of an optimization problem) *separately for each s-state* from the given set **S**. The global solution, a rational FT-TAll, is obtained as the result of superposing these TA plans for all the states of **S**. In this case, besides spending a great amount of time in solving the large number of "local" optimization problems, the obtained global solution is not guaranteed to be optimal. Even the local solutions may be sub-optimal if they were obtained through the application of heuristics.

We believe that CLP can be an efficient instrument for solving the problem of creating the optimal FT-TAll. Instead of finding FT-TAll by means of a "state-by-state" approach, CLP allows one to tackle the global optimization problem for the set **S** as a whole. The optimal FT-TAll plan and the individual plans for each state of **S** are obtained simultaneously by means of a procedure of constraint propagation. All constraints involved in the global problem are taken into account at once. Depending on the nature of the desired objective function, solutions found can be guaranteed to be globally optimal. The latter will probably constitute the main merit of applying CLP to FT-TAll problems, i.e., the possibility to find better solutions which are out of reach of the classical approaches. We consider FT-TAll creation for one of the common types of DCCS, namely, for a hierarchical system for technological process control, using the next statement of the problem.

Let the required level of a system fault-tolerance be specified by the given maximum acceptable number d of PMs, which may be at fault during a system operation. Then, our problem is the following: assuming that the number of faulty PMs does not exceed the given value d, create such an FT-TAll which guarantees execution of a system job with *minimal losses in each state* $\mathbf{s}_\nu \in \mathbf{S}$ (2.2), provided that the given constraints for the

temporal measure T_Σ^ν (2.3) and for memory volume of each PM are satisfied. The problem is formulated as follows:

$$R_\nu = \sum_{U_j \in \Omega} \rho_j d_{jf}^\nu \longrightarrow \min, \quad \forall \mathbf{s}_\nu \in \mathbf{S}, \qquad (2.4)$$

where $d_{jf}^\nu = 1$ if the task U_j is rejected in the state $\mathbf{s}_\nu$, i.e., is assumed to be assigned to fictitious PM M_f; otherwise, $d_{jf}^\nu = 0$;

$$T_{\Sigma i}^\nu = \sum_{U_j \in \Omega} \tau_j d_{ji}^\nu \le T_{\Sigma i}^*, \quad i = 1, \ldots, g_\nu; \ \forall \mathbf{s}_\nu \in \mathbf{S}; \qquad (2.5)$$

$$V_i = \sum_{U_j \in \Omega} v_j z_{ji} \le V_i^*, \quad i = 1, \ldots, n, \qquad (2.6)$$

where g_ν is the number of non-faulty PMs in the state $\mathbf{s}_\nu$; v_j is the memory volume for the task U_j program module; and z_{ji} is an element of the binary matrix of FT-TAll $\mathbf{Z}$ given by $z_{ji} = \bigcup_{\mathbf{s}_\nu \in \mathbf{S}} d_{jl}^\nu$, where $\cup$ denotes logical function OR; $T_{\Sigma i}^*$, and V_i^* are the maximal allowable values (for a PM M_i) of total execution time of all the tasks in the state $\mathbf{s}_\nu$ and of individual memory volume.

Note that the expressions (2.4) and (2.5) are given for each state $\mathbf{s}_\nu \in \mathbf{S}$, while the memory constraint (2.6) must be satisfied for the global FT-TAll accounting of all the states. Our previous solution using traditional optimization techniques implies creation of rational TA plans separately for each s-state $\mathbf{s}_\nu \in \mathbf{S}$ using a simplified memory constraint for each $\mathbf{s}_\nu$. Most likely, this solution is not optimal.

Now, we present the solution of this problem by means of CLP technology using the application example: a simplified fragment of DCCS of a gas distribution s stem composed by 7 PMs (with the parameters given by Table 1) in a hierarchical structure (figure). The system executes 21 tasks with the given parameters (Table 2): memory requirement v_j, execution time τ_j, losses ρ_j, and a set $\mathbf{H}_j$ of PMs in which the task U_j is allowed to be allocated (for technological reasons).

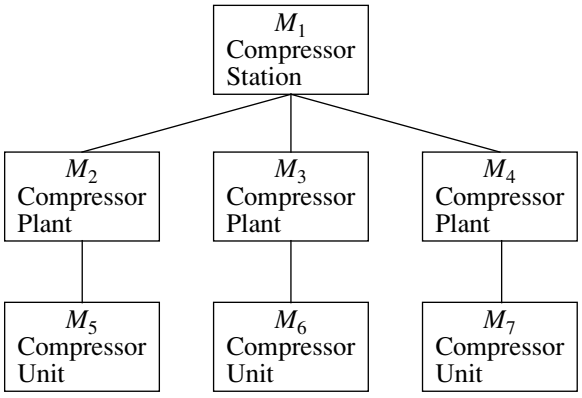

Figure.

The problem statement (2.4)–(2.6) for the set $\mathbf{S}$ determined by the given value $d = 1$ was implemented using the CLP language CHIP, v.5 from Cosytec. Since description of the complete program is believed to be far out of the scope of this paper, we give here some taste of how the particular problem aspects have been modeled.

Decision variables and search space. The basic decisions that have to be made in the above exposed problem consist of determining the assignment of tasks to PMs. Therefore, for each task, we introduce the assignment variable with the domain of values, which ranges from 1 to 8 (assigning to 1 of the 7 PMs or rejecting the task). In the final solution, these domains will have been reduced through constraint propagation to one single value. The number of these assignment variables will be one for every task in every stale; i.e., $21 \times 8 = 168$. As a consequence, the size of the unconstrained search space is 8^{168}, which is way beyond the capabilities of most classical search methods. The active use of *problem constraints* to *a priori* prune away large parts of this search space is a real necessity.

Forbidden assignments, either due to failure of a PM in a certain state or simply because of technological reasons (Table 2), are stated straightforwardly such as:

T31#\= 3 %Task 1 in state 3 cannot take value 3 (PM3 is faulty)% or **not in (T11, 2, 7) %Task 1 in state 1 cannot take values from 2 to 7%**.

Losses and optimization. The losses introduced by a task in a certain state are represented as a *domain variable*, which has two values: 0 when the task is not rejected, or some value dependent on the task when a task is rejected. To make the link between the assignment variable for a certain task and its loss variable R, the built-in symbolic constraint **element/3** is used, for example,

element(T11, [0, 0, 0, 0, 0, 0, 0, 500], R11).

As an optimization criterion, we want to minimize the maximal losses in any one of the 8 states where the losses R_ν in a state $\mathbf{s}_\nu$ are defined by a constraint of the kind:

R1 #= R11 + R12 + R13 + R14 + R15 + R16 + … .

The optimization predicate **min_max/2** is used to state the minimization criterion:

min_max(labeling(…),
[R1, R2, R3, R4, R5, R6, R7, R8]).

Time and memory constraints. Both time and memory constraints are of cumulative nature: as tasks are assigned to PMs, a resource of limited availability (either time or memory) is used, constraining the

Table 1. Parameters of the modules

Module $\mathbf{M}_i$	$\mathbf{M}_1$	$\mathbf{M}_2$	$\mathbf{M}_3$	$\mathbf{M}_4$	$\mathbf{M}_5$	$\mathbf{M}_6$	$\mathbf{M}_7$
Allowable Memory V_i^*	64	32	32	32	16	16	16
Allowable Total Execution Time $T_{\Sigma i}^*$	1.6	0.3	0.3	0.3	0.15	0.15	0.15

assignment of other tasks. This kind of constraint is typical in scheduling problems and is effectively handled by some powerful global constraints. The time constraints can efficiently be modeled through CHIP's **cumulative/8** constraint for resource-constrained scheduling problems. The constrained resource in this case is the execution time used per module. The memory constraint is somewhat more complicated due to the fact that it is a constraint spanning all states, while the time constraint can be stated for each state separately. Its implementation makes use of the global constraints **diffn/6** and **among/5**. It also enforces that a task be assigned to at most 2 different PMs in different states.

RESULTS

The program using a SparcStation 20 yields the optimal solution after 8 sec.: TA plans for the initial state $\mathbf{s}^0$ and for the distorted states $\mathbf{s}_k$, $k = 1, \ldots, 7$, each corresponding to a faulty PM M_k (Table 3, where each cell (j, k), present the number of a PM for allocating the task U_j in the state $\mathbf{s}_k$ and the global FT-TAll (Table 4). The minimal losses for the states $\mathbf{s}^0$, $\mathbf{s}_1$, …, $\mathbf{s}_7$ are [0, 780, 30, 30, 300, 30, 30, 300].

3. ARRANGING FAULT-TOLERANT OPERATION OF A POWER DISTRIBUTION SYSTEM ON THE BASIS OF CLP-TECHNOLOGY

The exploitation of a power distribution network involves the scheduling of multiple maintenance and unforeseen repair tasks. The main resource is a network subject to topological, economical, and electric constraints. A line section being maintained needs to be isolated from the rest of the network by opening all surrounding switches. This, in turn, would leave other areas of the network de-energized, which is unacceptable in most cases. Hence, these areas have to get their supply via some alternative way, i.e., service needs being restored by closing switches connected to an energized part of the network taking into account overloading of branches, energy losses, and the cost of the necessary switching operations. In case tasks are carried out in the same area, switching operations might be shared among them. In some cases, a valid network reconfiguration might not even exist. Finally, typical scheduling constraints have to be met: resources of limited availability (manpower, vehicles, etc.), due dates, priority relations, etc.

To solve this problem, the prototype scheduler PLANETS (*Planning Activities on NETworkS*) have been developed using the CLP-language CHIP [26]. It generates near-optimal schedules for the tasks to be carried out in one week making sure that, at any moment in time, the network is appropriately reconfigured to guarantee power supply to all consumers. The preparation of such schedules is not trivial. Certainly, the planning engineer is facing a complex decision problem since he needs to consider many constraints and assign values to many variables.

In order to carry out every foreseen maintenance task, it is first of all necessary to *isolate* the affected line section by opening all the nearest surrounding switches. Next, the engineer has to decide through which of the many possible alternative paths of the network he will keep on energizing all consumers affected

Table 2. Parameters of the tasks

Tasks U_j	Losses ρ_j	Memory v_j	Execut. time τ_j	$\mathbf{H}_j$						
				$\mathbf{M}_1$	$\mathbf{M}_2$	$\mathbf{M}_3$	$\mathbf{M}_4$	$\mathbf{M}_5$	$\mathbf{M}_6$	$\mathbf{M}_7$
1	5.0	36.0	1.0	1						
2	0.2	2.0	0.02	1	1	1	1			
3	4.0	1.0	0.1	1	1	1	1			
4	0.15	1.5	0.001	1	1	1	1			
5	0.1	1.0	0.1	1	1	1	1			
6	3.0	1.0	0.3	1	1	1	1			
7	6.0	0.8	0.02	1	1			1		
8	6.0	0.8	0.02	1		1			1	
9	6.0	0.8	0.02	1			1			1
10	0.3	0.5	0.001	1	1			1		
11	0.3	0.5	0.001	1		1			1	
12	0.3	0.5	0.001	1			1			1
13	2.5	2.0	0.1	1	1					
14	2.5	2.0	0.1	1		1				
15	2.5	2.0	0.1	1			1			
16	8.0	0.6	0.03	1	1			1		
17	8.0	0.6	0.03	1		1			1	
18	8.0	0.6	0.03	1			1			1
19	4.0	3.0	0.1	1	1			1		
20	4.0	3.0	0.1	1		1			1	
21	4.0	3.0	0.1	1			1			1

Table 3. Task Assignment plans for the states $\mathbf{s}_v \in \mathbf{S}$

Task U_j	State $\mathbf{s}_v$							
	$\mathbf{s}^0$	$\mathbf{s}_1$	$\mathbf{s}_2$	$\mathbf{s}_3$	$\mathbf{s}_4$	$\mathbf{s}_5$	$\mathbf{s}_6$	$\mathbf{s}_7$
1	1	8	1	1	1	1	1	1
2	1	2	1	1	1	1	1	1
3	1	2	1	1	1	2	1	2
4	1	2	1	1	1	1	1	1
5	1	2	1	1	1	1	1	2
6	1	4	4	4	8	4	4	8
7	2	2	1	2	1	1	2	1
8	3	3	3	1	3	3	1	1
9	4	7	7	7	7	7	7	4
10	2	2	1	8	1	1	8	1
11	3	3	8	1	1	8	1	1
12	4	8	1	1	1	1	1	1
13	2	2	1	1	1	1	1	1
14	3	3	1	1	1	1	1	1
15	4	8	1	1	1	1	1	1
16	5	2	5	2	2	2	2	2
17	6	3	3	6	3	3	3	3
18	7	7	1	1	1	1	1	1
19	5	5	1	5	5	1	5	1
20	6	3	3	6	3	3	3	3
21	7	7	7	1	1	1	1	1

Table 4. Global Fault-Tolerant Task Allocation (FT-TAll)

Task U_j	Module $\mathbf{M}_i$						
	$\mathbf{M}_1$	$\mathbf{M}_2$	$\mathbf{M}_3$	$\mathbf{M}_4$	$\mathbf{M}_5$	$\mathbf{M}_6$	$\mathbf{M}_7$
1	(1)	0	0	0	0	0	0
2	(1)	1	0	0	0	0	0
3	(1)	1	1	0	0	0	0
4	(1)	1	0	0	0	0	0
5	(1)	1	0	0	0	0	0
6	(1)	1	1	0	0	0	0
7	1	(1)	0	0	0	0	0
8	1	0	(1)	0	0	1	0
9	1	0	0	(1)	0	0	0
10	1	(1)	0	0	0	0	0
11	1	0	(1)	0	0	0	0
12	1	0	0	(1)	0	0	0
13	1	(1)	0	0	0	0	0
14	1	0	(1)	0	0	0	0
15	1	0	0	(1)	0	0	0
16	1	1	0	0	(1)	0	0
17	1	0	0	0	0	(1)	0
18	1	0	0	1	0	0	(1)
19	1	1	0	0	(1)	0	0
20	1	0	1	0	0	(1)	0
21	1	0	0	1	0	0	(1)

by a maintenance task. He has to *reconfigure* the network while keeping it radial, i.e., there may not exist any closed loops, and ensuring that current limits of all lines are not exceeded. The number of disconnected *consumers* has to be minimized. The disconnection of a customer is only justified in certain cases where there does not exist any reconfiguration of the network that can maintain power supply. The amount of unconsumed energy translates directly into profit loss for the company. It is necessary to achieve optimal usage of the company's limited resources involved in maintenance activities. These are basically equipment, mobile technical staff, transportable generators, distribution operators, and the distribution network itself. By properly constructing the schedule, this is possible. The dynamic behavior of the *energy demand* along the week must be considered. Thus, a maintenance task involving the disconnection of a main line carrying high current values cannot be performed at peak hours. The overload produced in neighboring lines as a consequence of the reconfiguration could cause significant damage. Additionally, temporal constraints must be met, such as priorities; *due dates*; and, for some tasks, *a priori* fixed dates. Finally, if one bears in mind the huge size of the electric networks dealt with (they can easily contain 2000 nodes and 800 switches), one understands the combinatorial complexity of the problem the planning engineer is periodically faced with.

The next kinds of constraints are implemented in the scheduler PLANETS by means of CHIP.

Isolation constraints: during job execution, the area surrounding the maintained branch must be in outage state. This is achieved by opening all surrounding switches. The constraint was implemented by means of an extension to the built-in **element/3** constraint, forcing a number of elements in a list of currents (of the maintained branch) or switch states (of the surrounding switches) to be 0, starting at a position indicated by a temporal domain variable (the start time of the job).

Resource constraints: the available amount of resources must be respected at all times. Possible resources required by a maintenance job are vehicles, manpower, etc. The constraint is enforced by a straightforward use of the built-in **cumulative/8**.

Precedence constraints: jobs on ancestor branches must be carried out before jobs on their descendant branches. The constraint translates into a number of inequality relations between temporal domain variables. Apart from this precedence, every job has additionally an absolute priority number. However, these priorities are not considered as "hard" constraints. They

merely act as preferences. Violations represent an additional cost per time slot that can be minimized.

Consumer constraints: at all times there must flow a minimal nonzero current to the consumers. These constraints will propagate changes to the domains of many other current variables and, doing so, will force the network to reconfigure itself in case of an outage. In case such a reconfiguration would be impossible, we allow exceptionally the isolation of some consumers during a period of at most the duration of the particular job. This exception was necessary to avoid failure of the scheduler and getting a *no* answer. It is implemented by the built-in **atmost/3** constraint: the list of currents supplying any of these consumers can have at most d zeros, where d is the duration of the particular job. For all other consumers, the constraint is enforced simply by setting the initial domain of their respective lists of current variables.

Continuity constraints: on all nodes, except the root and consumer nodes, the continuity law of Kirchoff must hold at all times. This constraint says that the sum of all incoming currents in a node must equal the sum of all outgoing currents. It forms the basis for the propagation of current domains through the network. These constraints are linear equations between domain variables.

Switch-behavior constraints: an open switch cannot carry a current. This constraint forms the basis for the topological reconfiguration of the network in all time slots. It is implemented by means of the conditional-propagation construct in CHIP. For all 15 elements of the lists of currents and the lists of switch states, we have

(if S #= 0 then I #= 0),

(if I #\= 0 then S #= 1).

Radiality constraints: at all times the network must be radial, i.e., not contain any closed cycles. This is enforced by searching all possible cycles and stating that, at any time, at least one of the switches in any cycle must be open, using the **built-in atmost/3**.

Overload constraints: in every branch, current must be below its allowable maximum. This constraint translates into simple inequalities on the current domain variables.

Energy-demand constraints: in every consumer branch the domain of the current variable is restricted to a pre-defined profile of energy demands.

Due-date constraints: any job must be finished before its due date. In the normal case, this is a “hard” constraint. However, the user can indicate that it should be treated as a “soft” one, minimizing violations which represent a cost per time slot.

A solution to the overall problem is generated by labeling the temporal and, afterwards, the topological domain variables, yielding a schedule and its associated network reconfigurations. Labeling is embedded in a branch-and-bound process using the min_max/2 meta-predicate. A function representing the *global cost of all necessary switching operations* is minimized. For this purpose, every switch has a nominal state and two associated costs: a cost for changing its state (from 0 to 1 or from 1 to 0) and a cost for staying in a non-nominal state during one time slot. These cost terms are added up for all switches and over all time slots, yielding the global cost.

The PLANETS scheduler and reconfigurator were completely written in CHIP and run on a Sun workstation. For a power distribution network of about 1200 nodes and 400 operable switches and 15 maintenance jobs to be scheduled, the system creates about 22 000 domain variables. The total time to produce the optimal solution with respect to the cost functions at hand is about 2.5 minutes CPU time on a SuperSparc20 with 70% of that time used to create the variables and set up the constraints; the remaining time is needed for variable labeling and branch and-bound minimization.

The developed system improves the exploitation of the power distribution network. From an economical point of view, an extra benefit is obtained by minimizing the total amount of undistributed energy due to forced maintenance outages. The global cost of carrying out the maintenance schedule is lowered due to the efficient use of the finite resources and the elimination of many redundant operations.

CONCLUSION

Thus, CLP-technology, which has emerged during the last decade, has already become a powerful tool for solving difficult combinatorial problems and, nowadays, is being used in a large variety of applications. In view of the results of research and a number of projects, it is clear that CLP is particularly well suited for tackling the problems of optimal resource and task allocation and scheduling. An additional advantage discovered during some projects is that bigger problems come within the reach of smaller programs. For solving a really big problem, it is worth applying CLP; for a small one, it is probably better to use traditional methods. The major advantage of CLP is saving a huge amount of time and resources (and, as a consequence, money) in software development and maintenance.

On the other hand, creating complex and demanding technical systems that would guarantee extremely high fault-tolerance of operation is one of today’s challenges. Implementation of modern promising concepts of creating highly fault-tolerant systems runs into highly dimensional discrete combinatorial and optimization problems, which are hardest to solve using conventional methods and programming techniques, which, moreover, cannot guarantee the optimal solution. This is the major obstacle for wide application of the new approaches. Employment of CLP technology with appropriate adaptation for solving the problems of fault-tolerance of complex systems seems to be an

effective way out. This paper demonstrates this possibility and exposes both a new promising application area for CLP and a new effective instrument to solve one of the crucial problems, namely, the problem of creating highly dependable systems.

REFERENCES

1. Sterling, L. and Shapiro, E., *Programming with Prolog Language*, Moscow: Mir, 1990.
2. Van Hentenryck, P., *Constraint Satisfaction in Logic Programming*, Cambridge, Mass.: MIT Press, 1989.
3. Wallace, M., Practical Applications of Constraint Programming, *Constraints, International J.*, 1996, vol. 1, no. 1.
4. Rennels, D.A., Fault-Tolerant Computing—Concepts and Examples, *IEEE Trans. Computers*, 1984, vol. C-33, no. 12.
5. Behr, P.M. and Giloi, W.K., The design of Fault-Tolerance in the UPPER System, *Computer Architecture Technical Committee NEWSLETTER* (IEEE Comp. Society), June 1985.
6. Deswarte, Y., Alami, K., and Tedaldi, O., Realization, Validation, and Operation of a Fault-Tolerant Multiprocessor: ARMURE, *16th Annual Int. Symp. on Fault-Tolerant Computing (FTCS'16)*, Vienna, Austria, 1986.
7. Hariri, S. and Raghavendra, C.S., Distributed Functions Allocation for Reliability and Delay Optimization, *IEEE/ACM*, 1986 Fall Joint Comp. Conf., Dallas, 1986.
8. Shatz, S.M. and Wang, J.-P., Models and Algorithms for Reliability-oriented Task Allocation in Redundant Distributed Computer Systems, *IEEE Trans. Reliability*, 1989, vol. 38, no. 1.
9. Tourouta, N., Organization of Task Allocation in Computing Systems That Ensures Their Fault-Tolerance, *Autom. Control and Comput. Sci.*, 1985, vol. 19, no. 1.
10. Tourouta, E.N., Fault-Tolerant Mapping Algorithms onto Hardware Structure of a Multiprocessor System, *5th Annual European Computer Conf. (COMPEURO'91)*, Bologna, Italy, 1991.
11. Montanari, U., Networks of Constraints: Fundamental Properties and Applications to Picture Processing, *In. Sci.*, 1974, vol. 7, no. 2.
12. Mackworth, A.K., Consistency in Network of Relations, *Artificial Intelligence*, 1977, vol. 8, no. 1.
13. Davis, E., Constraint Propagation with Interval Labels, *Artif. Intell.*, 1987, vol. 32, no. 3.
14. Gallaire, H., Logic Programming: Further Developments, *IEEE Symp. on Logic Programming*, Boston, 1985.
15. Cohen, J., Constraint Logic Programming Languages, *Commun. ACM*, 1990, vol. 33, no. 7.
16. Dincbas, M., Van Hentenryck, P., Simonis, H., Aggam, A., Graf, T., and Bathie, F., The Constraint Logic Programming Language CHIP, *Int. Conf. on 5th Generation Computer Systems (FGCS'88)*, Tokyo, 1988.
17. Jaffar, J. and Lassez, J.-L., Constraint Logic Programming, *14th ACM Symp. on Principles of Programming Languages*, Munich, 1987.
18. Colmerauer, A., An Introduction to Prolog-III, *Commun. ACM*, 1990, vol. 33, no. 7.
19. Aiba, A., Sakai, K., Sato, Y., *et al.*, Constraint Logic Programming Language CAL, *Int. Conf. on Fifth Generation Computer Systems (FGCS-88)*, ICOT, Tokyo, 1988.
20. Aiba, A. and Hasegava, R., Constraint Logic Programming Systems—CAL, GDCC and Their Constraint Solvers, *Int. Conf. on Fifth Generation Computer Systems (FGCS-92)*, ICOT, Tokyo, 1992.
21. Van Hentenryck, P., Simonis, H., and Dincbas, M., Constraint Satisfaction Using Constraint Logic Programming, *Artif. Intell.*, 1992, vol. 58, nos. 1–3.
22. Chiopris, C. and Fabris, M., Optimal Management of a Large Computer Network with CHIP, *2nd Int. Conf. on Practical Applications of Prolog (PAP-94)*, London, 1994.
23. Baues, G., Kay, P., and Charlier, P., Constrained-based Resource Allocation for Airline Crew Management, *Int. Conf. ATTIS-94*, Paris, 1994.
24. Bellone, J., Chamard, A., and Pradelles, C., PLANE - An Evolutive Planning System for Aircraft Production, *Int. Conf. on Practical Applications of Prolog (PAP-92)*, London, 1992.
25. Chamard, A., Deces, F., and Fischler, A., A Workshop Scheduler System Written in CHIP, *2nd Int. Conf. on Practical Applications of Prolog (PAP-94)*, London, 1994.
26. Creemers, T., Ros, L., Riera, J., *et al.*, Constraint-based Maintenance Scheduling on an Electric Power-Distribution Network, *3rd Int. Conf. on Practical Applications of Prolog (PAP-95)*, Paris, 1995.
27. Tourouta, E.N., The Methods for Ensuring Fault-Tolerance of Distributed Control Systems, *IFAC Symp. SAFEPROCESS'94*, Helsinki, 1994.
28. Rembold, U., Nnaji, B., and Storr, A., *Computer Integrated Manufacturing and Engineering*, Addison-Wesley, 1993.