Coordinating Distributed CLP-Solvers in Medical Appointment Scheduling*

Markus Hannebauer[1] and Ulrich Geske[2]

[1] think-cell Software GmbH, Invalidenstr. 34, D-10115 Berlin, Germany
mhannebauer@think-cell.com
[2] Fraunhofer FIRST, Kekuléstr. 7, D-12489 Berlin, Germany
Ulrich.Geske@first.fraunhofer.de

Abstract. Research on monolithic logic-based systems has usually left out an important aspect of information technology systems supporting medical processes — distribution. Often social restrictions build up by questions of authority prohibit the implementation of global systems with omniscient view. Hence, IT in medical domains has to obey these restrictions by being distributed as well. The direct consequence is the need for communication and coordination. In this paper, we present an industrial-size case study in medical appointment scheduling that is envisaged to be solved by combining the strength of CLP for local (internal) problems with the strength of coordination for external problems. The essence of this approach is the realization of a multi agent system (MAS) consisting of CLP-based agents.

1 Introduction

Applications of information technology in medical domains are confronted with at least two aspects. The first aspect is the support of complex decisions in diagnosis, therapy and administration. Usually, these decisions include some kind of planning, scheduling or reasoning on expert knowledge. Research on Artificial Intelligence has made many successful contributions to these fields in general by introducing action planning, intelligent scheduling and expert systems. Logic Programming and Constraint Logic Programming (CLP) are among the most influential paradigms in this context. Nevertheless, research on monolithic logic-based systems has usually left out the second important aspect of information technology in medicine — distribution.

Almost all medical processes are distributed spatially and among several individuals. Social restrictions build up by questions of authority prohibit the implementation of global systems with omniscient view. Hence, IT in medical domains has to obey these restrictions by being distributed as well. The direct consequence is the need for communication and coordination. The notion of an *intelligent agent* [10, 11] is a recent concept that tries to incorporate the

* revised version; originally published in the Proceedings of the 12th International Conference on Applications of Prolog. Prolog Association of Japan, September 1999

merits of sophisticated AI methods with strong communication and interaction capabilities.

In this paper, we present an industrial-size case study in medical appointment scheduling that is envisaged to be solved by combining the strength of CLP for local (internal) problems with the strength of coordination for external problems. The essence of this approach is the realization of a multi agent system (MAS) consisting of CLP-based agents. After describing the case study, our general strategies and the given variables and constraints in section 2, we focus on the coordination of distributed CLP solving in section 3. Discussion of related work and prospects conclude this article.

2 Medical appointment scheduling

2.1 Case study — ChariTime

CLP and intelligent agents have traditionally been applied to the control and optimization of industrial transport and production processes. In contrast to that, our research is more involved with human processes in the domain of administration and health care management. Several researchers from GMD FIRST, Humboldt University Berlin and Technical University Berlin are currently carrying out a case study at the cardiological clinic of Charité Berlin, Europe's biggest hospital. The cardiological clinic of Charité consists of five wards with a capacity of altogether over 80 patients, four outpatients' facilities, in which different types of medical consulting are done in parallel, and eight diagnostic units, some of which with several workplaces. The diagnostic units perform over 100 diagnostic examinations each day. These examinations are requested by the wards, the outpatients' department and other clinics of Charité.

The present problem is the coordination between the requesting and serving units. Spatial and organizational distribution of the named units results in distributed knowledge, distributed control and hence suboptimal patient throughput and resource usage. Traditional monolithic systems often scale purely in measure of process instantiations and they usually ignore the problem of restricted information distribution. Therefore, a more local and flexible architecture is needed to control the requesting and serving processes in diagnosis.

We have decided to design and realize a truly distributed multi agent system which will (hopefully) run on 25 to 30 computers all over the whole cardiological clinic. This system is called ChariTime. It shall be permanently active to allow the dynamic allocation of actors and resources to diagnostic tasks, while coping with failures and emergency cases. We have modeled the given problem as a coordinated problem solving task among autonomous but benevolent agents.

2.2 General strategies

A usual problem of socially embedded IT systems is acceptance. The introduction of IT is often a management decision as it is in our case. Especially scheduling appointments automatically can have major impact on every day's work. Therefore, in ChariTime we have introduced a variety of properties that can be adjusted by the users and influence the way in which the system schedules appointments.

A key property is the possibility to restrict the number of fixed appointments for every type of examination for each day. Fixed appointments are preferably given to outpatients. Fixed appointments that are not used for outpatients are given to ward patients. Ward patients that have not been assigned a fixed appointment are registered in a priority queue that can be used by the diagnostic unit to flexibly fill the time between fixed appointments.

Appointment relocation is very common in medical appointment scheduling. The request for an appointment is not only prioritized by a basic patient priority but also by medical priorities of every examination requested. This priority is used to determine what we call *appointment modifiability*. Appointments for outpatients are not modifiable, appointments that are part of an examination chain are barely modifiable and appointments for ward patients are easily modifiable. All this allows for relocating already scheduled appointments in favor of a high priority new appointment. In case of relocation, all affected entities are informed and rescheduling is initiated automatically.

To leave some degree of control on appointment scheduling in the hands of the people working in the diagnostic units, we have introduced a threshold for automatic booking of appointments. This threshold represents a certain relative time point in the near future (for example "in one week"). Though the system may automatically propose appointments lying within the time interval defined by this threshold, diagnostic unit users must give final permission for the appointment to become fixed. Appointments lying beyond the time point defined by the threshold are allowed to be automatically fixed. The semantics of this threshold is the representation of the desire for controlling one's own near future, while not really caring for appointments that lie far in the future. Another property in this context is that the future time intervals have to be actively released by the employees of the diagnostic units. This allows to flexibly determine uncommon off times.

Requesters like wards or the outpatients' department can control the scheduling process by providing information on desired time intervals for appointments, priorities and other restrictions like appointments in other clinics. People in the outpatients' department usually receive several alternative appointment possibilities for their requested examinations and can interactively select one or reconsider their desired time interval. People at the wards can select patients to become aspirants for fixed appointments or to be registered in the priority queue.

2.3 Modeling

Appointments in general In the following, we will describe a static variant of our medical scheduling problem. Components of the presented algebraic tuples are accessed by the "." operator. So, $x.name$ denotes the projection on the component with name $name$ of the tuple x.

In medical appointment scheduling we are doing constraint satisfaction and optimization over finite domains. Hence, we have to enforce the existence of a scheduling horizon.

Definition 1 (Horizon). *A* horizon $h = \{0, \ldots, H\}$, $h : 2^{\mathbb{N}} = \mathcal{H}$ *is a finite set of integers that represents the set of possible starting points for appointments. Given a constant number of starting points per day nd the* day horizon *can be defined as* $h' = \{1, \ldots, H \text{ div } nd + 1\}$. □

The central term in our domain are appointments. These objects are used to encapsulate all domain variables, initial domain information and given constants of an appointment. The definition of an appointment bases on the definition of time slots.

Definition 2 (Time slot). *A* time slot *t is a pair*

$$t = (start, duration), \qquad t : (h \times \mathbb{N}) = \mathcal{T}.$$

start is a variable ranging over the horizon h representing the starting point of t. duration is a constant representing the fixed duration of t. □

Definition 3 (Appointment). *An* appointment *a is a 9-tuple*

$$\begin{aligned} a = &\ (id, type, slot, desstart, day,\\ &\ pr, workpl, desworkpl, resource),\\ a\ :&\ (Id \times \mathcal{AT} \times \mathcal{T} \times h \times h' \times\\ &\ \mathbb{N} \times \mathcal{W} \times \mathcal{W} \times \mathbb{N}) = \mathcal{A}. \end{aligned}$$

id is an identifier. type refers to an appointment type (see 2.3). slot refers to a time slot. desstart is a constant that denotes a desired starting time point of the appointment. day is a variable denoting the day of the appointment's starting time point. pr is the appointment priority. workpl is a variable that represents the choice of a concrete workplace for the appointment (see 2.3). desworkpl denotes a desired workplace for the appointment. resource represents the human resource demand of the appointment. □

Using the combination of *desstart* and *priority* we can implement the described strategy of appointment relocation and appointment modifiability. When requesting the scheduling of a set of new appointments the scheduling of recent appointments is also reconsidered. Though recent appointments may be relocated in favor of a high priority new appointment request, usually they should remain at their determined location. This is expressed using the *desstart* component. The *priority* component of such recent appointments has to be high enough to avoid superfluous relocation. This is a matter of defining the optimization criterion which is discussed in 3.4. $desstart = -1$ indicates an appointment that has to be scheduled as soon as possible. *day* is a transformed variable that ranges over h' and is bound to the appointment starting time point by defining the number of starting points per day *nd* and posting the FD constraint *a.slot.start* #= *nd* * (*a.day* - 1) + *Rest* (with $Rest \in \{0, \ldots, nd - 1\}$).

Provider objects In our application context, diagnostic units are providers. Every diagnostic unit provides a unique set of examination types. Hence, there is no choice between diagnostic units for doing a certain examination. Nevertheless, many units have several sub units, called workplaces. In contrast to the diagnostic units, the capabilities of these workplaces are not disjunct. So we have to introduce choice on the concrete workplace in a determined diagnostic unit. This choice is constrained by the appointment types provided by the workplaces. Appointment types are defined as follows.

Definition 4 (Appointment type). *An* appointment type *at is a 4-tuple*

$$at = (id, StartRange, MaxPerDay, ChangeTimes),$$

$$at : (Id \times 2^{h} \times 2^{2^{\mathcal{A}\times\mathcal{A}\times\dots}} \times 2^{2^{\mathcal{A}\times\mathcal{A}}}) = \mathcal{AT}.$$

id is an identifier. StartRange is a subset of the horizon h and represents possible starting times for appointments of this type. MaxPerDay is a set of constraints with dynamic arity restricting the maximum number of appointments of this type for each day. ChangeTimes is a set of binary constraints enforcing constant buffer times between two appointments of this type. □

MaxPerDay is motivated by the fact that people in the diagnostic units want to control the maximum number of appointments of the same type each day. *MaxPerDay* can be easily implemented by using several ECL^{i}PS^e `atmost` constraints. *ChangeTimes* is technically motivated by necessary change times between two distinct examinations of the same type. Given the necessary change time *c*, *ChangeTimes* can be implemented by a set of $a_1.slot.start$ + $a_1.slot.duration$ + c #<= $a_2.slot.start$ #\/ $a_2.slot.start$ + $a_2.slot.duration$ + c #<= $a_1.slot.start$ constraints.

Besides the different appointment types, every workplace can have individual off times which restrict possible starting times of appointments. Off times are usual time slots with a fixed *start* component.

Definition 5 (Workplace). *A* workplace *w is a triple*

$$w = (id, AppTypes, OffTimes),$$

$$w : (Id \times 2^{\mathcal{AT}} \times 2^{2^{\mathcal{A}\times\mathcal{A}\times\dots}}) = \mathcal{W}.$$

id is an identifier. AppTypes denotes a set of appointment types provided by the workplace. OffTimes is a set of constraints with dynamic arity restricting the starting times of appointments to form a mutually exclusive schedule. □

The definition of a diagnostic unit is now canonical.

Definition 6 (Diagnostic unit). *A* diagnostic unit *u is a triple*

$$u = (id, Workplaces, Resources),$$

$$u : (Id, 2^{\mathcal{W}}, 2^{2^{\mathcal{A}\times\mathcal{A}\times\dots}}) = \mathcal{U}.$$

id is an identifier. Workplaces is a set of workplaces belonging to this diagnostic unit. Resources is a set of constraints with dynamic arity restricting the maximum number of resources available for parallel appointments at the workplaces. □

Resources is motivated by the fact that though there may be several workplaces in a diagnostic unit not all of these may be usable in parallel. For example, staff is assigned to diagnostic units and not to workplaces. Hence, if an appointment requires two technical assistants and one doctor (denoted by the *resource* component of the appointment), no other appointment may be possible, though other workplaces may be free. *Resources* is implemented using ECL^{i}PS^e's `cumulative` constraint over all workplaces of the diagnostic unit.

Alternatives in choosing workplaces can be modeled in CHIP by using the `diffn` constraint, which is very efficient in handling process alternatives. ECL^{i}PS^e provides only the `cumulative` constraint, which can be interpreted as a one-dimensional specialization of `diffn`. Nevertheless, it is well known that `diffn` can be emulated with `cumulative` constraints by introducing choice variables and transforming the given starting time variables to new variables. A new variable is then bound to the sum of the standard starting time variable and the product of the choice variable and the horizon. For example in case of two workplaces in a diagnostic unit, the scheduling horizon doubles and the transformed variables have a domain twice as large as the standard starting time variables.

Requester objects Requesters can be interpreted as representatives of a set of patients. In appointment scheduling, a patient is mainly defined by his/her assigned partially ordered set of (open/fixed) appointments and a basic priority.

Definition 7 (Patient). *A patient p is a 5-tuple*

$$p = (id, Apps, Order, Excl, pr),$$

$$p : (Id \times 2^{A} \times 2^{2^{A\times A}} \times 2^{2^{A\times A\times\dots}} \times \mathbb{N}) = \mathcal{P}.$$

id is an identifier. Apps is a set of appointments. Order is a set of binary constraints over appointments, defining a partial order on the patient's appointments. Excl is a set of constraints restricting parallel appointments for this patient. pr is a priority. □

Since patients can only undergo a single examination per time slot every appointment has to be executed mutually exclusive. *Excl* guarantees this by posting corresponding `cumulative` constraints over the sparse "resource" patient.

3 Distributed CLP solving

3.1 Motivation for distribution

The problem modeled by the variables and constraints mentioned above could be solved completely by a monolithic CLP solver. Though this possibility exists in principle, it is usually not practicable in real medical application setting. The information on variables and constraints is spatially distributed among the many departments of the clinic. Even in case of a monolithic solver one would have to collect all the information from its several sources, transfer it to the solver, solve the problem and again distribute the results of optimization among the different users. Hence, even in case of a central optimizer one has to cope with communication and information consistency problems.

A second argument for distribution is the poor scalability of central solvers. As soon as not only one clinic of Charité Berlin would be connected to the appointment scheduling system, a solver would be needed that would have to compute the solutions for all connected clinics. As experience with CLP shows, problems get quickly too complex to be solved efficiently with such an approach. Since constraint satisfaction is known to be NP-hard, the only way to cope with this complexity problem is to partition the problem and accept inevitable suboptimal solutions. At this point, distribution is a should-have.

A third argument against a central solver is privacy. Social structures especially in hospitals have created a heterogeneous field of competencies and influences. No director of a single clinic would accept transferring all his or her clinic's appointment data to another clinic for global optimization. Even less she or he would accept automatic control over her/his clinic's appointments from a central instance. For acceptance, there have to be secure interfaces between realms of competency that only let pass authorized and restricted information. Decisions on appointments have to be done at the same locations of competency where examinations will take place in reality. At this point, distribution is already a must-have.

A last argument for distribution is redundancy and responsiveness. A crash of a central solver or missing connectivity would influence the whole hospital leading to chaos. Master/slave concepts raise the amount of communication overhead by caching and mirroring. In contrast to that, the crash of a single optimizer in one diagnostic unit would influence only that unit and its neighbors.

Despite these advantages of distribution, such systems have also major disadvantages. The complexity that has been saved within the several solvers is transfered to the coordination process. Due to this fact, investigations on todays distributed solver systems often report poor optimization results or vast communication overhead. The traditional approach to distributed problem solving is to design the distribution aspects off-line by statically assigning certain roles and competences to specific agents. Thus, the problem space is statically distributed among a more or less fixed set of agent types. Our approach differs from this by trying to allow on-line modification and reconfiguration of the MAS structure. Together with measures for internal problem solving complexity and communication overhead, the system shall automatically adapt to the current problem structure by melting and splitting problem solving knowledge, tasks and skills. A step towards this target are *composable agents*. This term denotes entities that are built of certain independent components, which represent pieces of knowledge, goals and problem solving capability. By exchanging their components these entities can dynamically reconfigure to fit the current situation better. More detailed information on composable agents and dynamic reconfiguration can be found elsewhere [8,9]. In this article we will focus on the coordination aspects for controlling the interaction of distributed CLP solvers.

Though the structure of distribution is not fixed in our approach to distributed CLP solving, one has to start the system with an initial distribution. Following natural distribution and competency areas we have defined agents that care for patients, agents for requesters in general and agents for diagnostic units. Figure 1 shows a distributed constraint graph with two patient agents and two diagnostic unit agents. As can be seen in the figure, not only variables (appointments a_1 to a_7) are denoted by nodes and designated to agents but also complex constraints (the *cap* node denotes all constraints mentioned above under the provider's view). Thin lines illustrate constraints within an agent. Thick lines illustrate constraints between agents. Dashed lines mean partial order time constraints, solid lines capacity constraints.

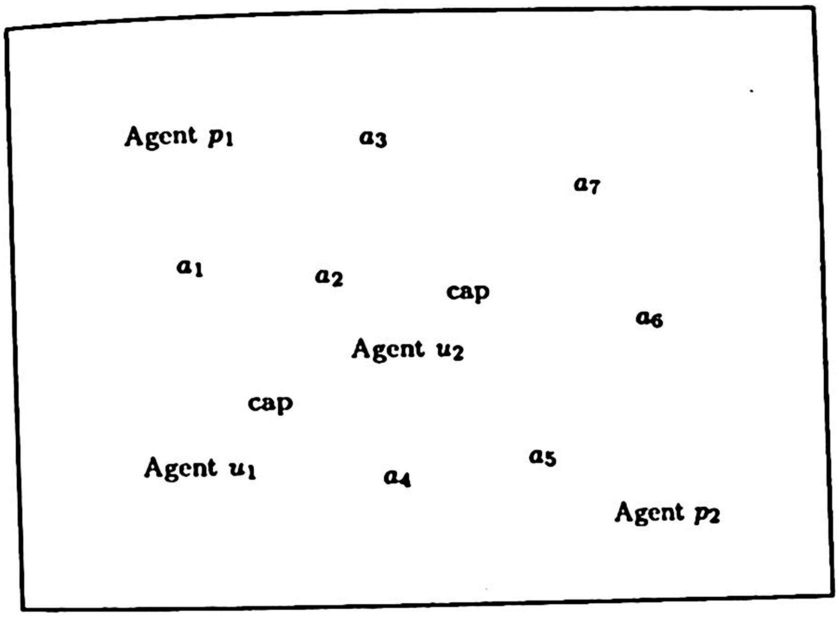

Fig. 1. Distributed Constraint Graph

3.2 Implementation tools

While (constraint) logic programming languages are well suited for knowledge-based computing, like reasoning, planning and optimization, they are not well suited for distributed computing. Therefore, we have decided to use a hybrid tool set of languages each of which fits best to the specific problem.

In Charité Berlin we are facing a pure Windows NT environment. Hence, we are using Microsoft's *Distributed Component Object Model* (DCOM, a CORBA like object request broker) and Message Queue Server (MSMQ, a fully transparent email service for programs) as communication foundations that provide useful abstractions from usual communication protocols, like TCP/IP. Access to DCOM and MSMQ is best done via Visual C++ providing the speed we need. User interfaces can be rapidly developed using Visual Basic. For the internal realization of problem solving agents we use ECL^iPS^e since it is highly expressive, efficient and free of charge for research projects. Additionally, ECL^iPS^e functionality can be linked into C++ code via a Dynamic Link Library (DLL). To put it into a nutshell, the system is controlled by C++ code, displays data with VB code and solves complex problems with ECL^iPS^e code.

3.3 Mechanisms for coordination

Algorithm 1 solve(A)

```
{Compute all neighbors holding constraints on $A$.}
$N \leftarrow$ comp_neighbors($A$);

{Collect declarative information on all constraints imposed on $A$ from all neighbors
and store it in the set of external constraints $C_{ext}$.}
$C_{ext} \leftarrow \emptyset$;
for all $n \in N$ do
  $C_{ext} \leftarrow C_{ext} \cup$ requ_ext_constr($n, A$);
end for

{Compute internal constraints on $A$, compute the local optimization function $z$ and
find a solution $s$ to $A$ internally w. r. t. $C_{ext}$ and $C_{int}$.}
$C_{int} \leftarrow$ comp_int_constr($A$);
$z \leftarrow$ comp_int_opt_crit($A$);
$s \leftarrow$ solve_int($A, C_{ext}, C_{int}, z$);

{The information on external constraints is not static (committed) and asynchronous
solutions of other agents might have invalidated $C_{ext}$ and thus $s$. Hence, the agent
must inform all neighbors on its new solution to $A$. Nevertheless, it assumes that the
problem is solved and sets the state of $A$ optimistically. To react to asynchronous
information on $A$ the agent calls a monitoring function for each $a \in A$.}
for all $n \in N$ do
  inform($n, A, s$);
end for
for all $a \in A$ do
  $a$.solved $\leftarrow$ true;
  newThread (monitor($a$));
end for
```

In the case of distributed problem solving, it should be the same whether the constraints between appointments are internal to a unique agent or externally distributed among several agents. The coordination protocol should allow for both. The following algorithms describe parts of a first approach to a flexible coordination protocol for distributed CLP solvers. Algorithm 1 implements an external request for constraints on the current given set of appointments A that have to be scheduled. This algorithm describes the view of a requester.

The algorithm has three stages. In the first stage, the agent collects information on constraints restricting its set of appointments A. This stage could be called *external constraint propagation*. This is done by determining all neighbors of the agent that hold constraints on A and then requesting declarative descriptions on their constraints. The degree of declarative description is dependent on social restrictions mentioned above. A neighbor that belongs to another competency realm may only answer with a restricted domain on A. A neighbor from the same competency realm may provide more information (for example all constraints described above, which would correspond to providing information on the whole current schedule) to allow for a higher quality solution. The advantage of CLP in this case is the possibility to encode even complex constraints in relatively simple string expressions that can easily be exchanged among agents.

In the second *internal propagation and search* stage, the agent collects its own internal constraints on A, computes a local optimization criterion on A and finally uses its internal solving capability to find a labeling for A[1].

In the third *monitoring* stage, the agent informs all neighbors on its solution and starts monitoring on the single appointments in A. This is shown by algorithm 2. Monitoring primarily means watching the acceptance of other agents for the proposed solution. In case of a `NO-GOOD` message from one of the neighbors, the agent retracts its solution to a and restarts the solving process for a.

Obviously, this coordination protocol is not complete, because it may leave out certain solutions. Nevertheless, it is correct and can avoid cycles by defining a dynamic priority order over agents for keeping an agent from permanently retracting its solutions. And it is efficient since it does no backtracking, but rather a kind of backjumping. Apparently, this protocol works like a usual central CLP solver going through the stages of constraint propagation, labeling and monitoring, but all this in a distributed asynchronous manner.

3.4 Example for solving an internal problem

The previous subsection has assumed the existence of an internal problem solver that can obey external as well as internal constraints. If all external constraints are communicated in a CLP syntax they can easily be incorporated to a local CLP solving process. We will focus on the side of a provider to give an example for this. In the above described coordination scenario a requester would first of all request information on constraints on a set of open appointments A. The task of the provider is to answer the request for constraints lying on A according to its knowledge. The provider is assumed to be benevolent, so it will answer honestly. Nevertheless, it will follow its own optimization strategy in making

[1] Since there are usually no deadlines in medical appointment scheduling, we assume that there will be always a solution for A if the horizon is large enough. The difference is only the solution's quality.

Algorithm 2 monitor(a)

```
{Compute all neighbors of constraints on $a$. Wait for message or time events triggered
by $a$.}
$N \leftarrow$ comp_neighbors($a$);
$e \leftarrow$ wait_for_events($a$);

{React to the triggered event according to its type.}
select $e$
  {In case of another agent sending a "no-good" message
  retract $s$ and inform all appropriate neighbors that $a$
  is unsolved again. Adjust state of $a$ accordingly.}
  case NO-GOOD:
    for all $n \in N$ do
      inform($n, a, \perp$);
    end for
    $a$.solved $\leftarrow$ false;
    newThread (solve({$a$}));

  {Handle other messages according to $a$. }
  case ?:
    ⋮
end select
```

proposals for A. In our case, this optimization strategy tries to change as few recent appointments as possible, but to schedule the appointments in A as soon as possible in favor of the requester. Given the set of recent appointments A_R, the set of requested appointments A and a weight for penalizing displacement *displ* this optimization strategy can be formalized by

$$z = \sum_{a \in A_R} [(a.slot.start - a.desstart) \cdot a.pr \cdot displ] + \sum_{a \in A} [(a.slot.start - a.desstart) \cdot a.pr] .$$

This value has to be minimized.

As usual in CLP, for reaching completeness not only the presented constraints have to be posted but all free variables have to be labeled by a heuristic search procedure. Starting time variables *a.slot.start* and workplace choice variables *a.workpl* are the free variables of our problem. Heuristics for choosing the next variable to label and for choosing the next value to assign to this variable are manifold and have been reported in several papers on CLP. In most cases, heuristics tailored to the application domain are most successful, since they can incorporate specific domain knowledge.

We have designed our labeling heuristics to fit the demands set by the optimization function presented above. Since we use a branch-and-bound method for optimization it is most important to find solutions with expected high optimality first to tighten the bounds on the searched solution early. That means to label the variables starting with the highest optimality solution candidates even though risking inconsistency. Then we stepwise deviate from this solution until consistency is reached. In our example that means to start labeling variables with high priority first, because thereby they won't be subject to backtracking soon. In general all workplace choice variables are labeled first, because they often have much tighter domains than the starting point variables. For value ordering we use a special strategy that labels starting point variables initially from their desired value (*a.desvalue*) and in case of backtracking cyclically around this value (one value left, one value right, two values left, two values right ...). First simulations show that the combination of these variable and value ordering heuristics speed up the search for good solutions remarkably in comparison to standard labeling strategies.

The results of optimization on the provider's side can be reached back to the requester for constraining the choices on A. The requester will then calculate its own optimization function, for example minimizing the patient's stay in hospital or trying to create examination chains, and search for an externally and internally consistent good solution. This solution is handed back to the providers and they can decide whether the conditions under which they gave their proposals still hold. In this case, the reported appointments are fixed and go into the recent appointments. Otherwise, the providers can send NO-GOOD messages, thus restarting the process.

4 Comparison to other DCSP approaches

Most related work in Logic Programming and CLP has considered parallel evaluation of goals (and-/or-parallelism) or concurrent approaches using a shared store [17]. These approaches are interesting, but they are not fully applicable to the distributed setting found in medical appointment scheduling. More appropriate are approaches for solving problems known as *Distributed Constraint Satisfaction Problems* (DCSPs). Though there are DCSP models commonly excepted by several researchers, there are also some alternative models that allow for a different view on the problem and such for different algorithms.

An excellent, yet a little out-dated overview to DCSP models and algorithms is given in [15] and [12]. The authors identify four basic elements in solving DCSPs: centralized or decentralized control, shared or separated search space, message-passing or shared memory and termination detection. In this sense, our research tends to do coordinated problem solving with decentralized control, shared search space, message-passing and without termination detection. The latter is due to the fact that termination detection is not so important in dynamic problems, since new tasks may arise on any time. Luo et al. also present an interesting classification of DCSP solving approaches. They distinct variable-based approaches (in which every agent cares for a subset of variables), domain-based approaches (in which ever agent cares for a subset of values for a unique variable) and function-based approaches (in which costly computations in centralized CSP solving are distributed to speed them up). Our concepts are designed to solve problems variable-based, since this is the only approach to allow for social and natural borders between subproblems. In [12, 14, 13] the authors propose different algorithms to solve DCSPs variable-based, domain-based and function-based. They all assume a binary DCSP and are hence based on simple constraint representations via no-good-sets.

An important contribution to DCSP solving has been given by Sycara, Roth, Sadeh and Fox in [20] in which they present *Distributed Constrained Heuristic Search*. They identify important characteristics of collaborative problem solving: global system goal to satisfy all constraints and minimize backtracking (equivalent to computational effort), concurrent and asynchronous variable instantiations, limited communication, incomplete information and potentially major ripple effects of backtracking. They also characterize the design trade-off for a proper level of distribution in a system for a given communication bandwidth, but do not address this problem in the paper. Though their proposal is mainly focussed on job-shop-scheduling they have already used a combination of distributed constraint propagation (in form of communicating resource demands) and distributed heuristic search (called *asynchronous backjumping*). The authors' introduction of special resource monitoring agents and job agents and the according cooperation protocol can be seen as predecessors of the ideas presented in this article. Sycara et al. characterize the effect of different decompositions and their characteristics to be a subject of future research.

Being another classical reference in DCSP, the work of Yokoo and Ishida introduces a DCSP model that simply assigns the variable nodes of a binary CSP graph to the different agents. Hence, this is a variable-based approach. Their main contribution lies in the development of distributed search algorithms, like *asynchronous backtracking* and *asynchronous weak-commitment search*. The earlier versions (collectively presented in [23] and [25]) relied on the assumption, that every agents cares for just one variable. Newer versions ([24]) overcome this restriction by allowing complex local problems. All these algorithms are correct and complete. To coordinate the different forms of asynchronous backtracking, the algorithms establish a static or a dynamic order among agents that determines the cooperations patterns between agents. In their work, Yokoo and Ishida mainly cover search and not so much constraint propagation. Additionally, the assumption of simple binary constraints restricts the applicability in real-world settings. Nevertheless, their coordination procedures have influenced much other work in this field. The same holds for the coordination protocols in this article.

Also in [25] two constraint propagation techniques are mentioned: a filtering algorithm reported in [22] and a hyper-resolution-based consistency algorithm described in [2]. The filtering algorithm achieves arc-consistency by communicating the domains of each process to the neighbors and removing values from these domains that cannot satisfy the given constraints. The hyper-resolution-based consistency algorithm applies a logical transformation rule to combine communicated constraints and information on an agent's domain to form tighter constraints. Both algorithms do not transmit abstract constraint information but concrete domains or no-good-sets of variable labelings that are inconsistent.

Hence, one weakness of these algorithms is the vast amount of communication since enumerating domains or constraints as simple data types can be highly space-demanding. The coordination protocol presented here uses high-level logic description to pass constraints from one agent to another, thus saving communication overhead. This is related to the work presented in [26]. Zhang and Mackworth propose a distributed arc-consistency check that uses an abstract constraint propagation facility and joins the communicated constraints with internal constraints. They also present complexity results for acyclic constraint graphs.

Another pre-processing distributed arc-consistency algorithm DisAC4 is discussed in [21] (see also [16]). It is a distributed version of the sequential AC4 algorithm and assumes that every agent is assigned exactly one variable. By simulating the behavior of several such agents more than one variable can be checked by a single agent.

Another approach to DCSP solving does not try to solve the DCSP with new distributed propagation or search methods but to facilitate existing CSP solvers to solve the problems local to an agent and then to combine the results of these solvers. An early reference on this approach is [1]. They introduce the notion of *interface problems* by partitioning a DCSP along variable nodes and not as usual along constraint arcs. All variable nodes that belong to more than one agent form a new problem — the interface problem. The variable nodes not belonging to the interface problem can be labeled independently from other variable nodes. Such, solving the interface problem and then solving the independent problems eventually using backtracking solves the whole problem. A disadvantage is the need for a global instance for finding the solution to the interface problem and collecting the solutions of the independent problems. Solotorevsky and others ([19]) follows a similar strategy by defining *canonical DCSPs* which consists of a special constraint graph connecting all independent local constraint graphs. Similar to Berlandier and Neveu they use common CSP solvers to solve the partitioned problems. All these authors assume a given partitioning of the DCSP and facilitate a global instance for guiding the solving process.

Solotorevsky and Gudes have applied their DSCP approach to time tabling in a hospital [18]. Decker and Li apply their generalized partial global planning approach to patient scheduling [3]. Despite these research efforts, we do not know any MAS that actually solves the problems we are facing in the ChariTime project.

5 Conclusion

Until now, the concepts presented here are just a vision. Nevertheless, our working groups have achieved promising results in central optimization by constraint-based approaches. Examples are job-shop-scheduling and time-tabling problems [4, 5].

The efforts in distributed CLP solving on the conceptual level and ChariTime on the applicational level are based on recent research in distributed production control [6] and include besides the presented concepts business process modeling based on Petri Nets [7]. The ChariTime team is currently beginning realization and we hope to achieve first results at the end of 1999.

References

1. P. Berlandier and B. Neveu. Problem partition and solvers coordination in distributed constraint satisfaction. In *Proceedings of the Workshop on Parallel Processing in Artificial Intelligence (PPAI-95)*, Montréal, Canada, 1995.
2. J. de Kleer. A comparison of ATMS and CSP techniques. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, pages 290–296, 1989.
3. K. Decker and J. Li. Coordinated hospital patient scheduling. In *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS-98)*, Paris, France, 1998.
4. H.-J. Goltz and U. John. Methods for solving practical problems of job-shop scheduling modelled in CLP(FD). In *Proceedings of the Conference on Practical Application of Constraint Technology (PACT-96)*, London, UK, 1996.
5. H.-J. Goltz, G. Küchler, and D. Matzke. Constraint-based timetabling for universities. In *Proceedings of the Eleventh International Conference on Applications of Prolog (INAP-98)*, pages 75–80, 1998.
6. M. Hannebauer. B-DICE — A BDI control environment for manufacturing systems. Master's thesis, Humboldt-Universität zu Berlin, Germany, 1998.
7. M. Hannebauer. From formal workflow models to intelligent agents. In *Proceedings of the AAAI-99 Workshop on Agent Based Systems in the Business Context*, pages 19–24. Technical Report WS-99-02, AAAI Press, 1999.
8. M. Hannebauer, H.-D. Burkhard, J. Wendler, and U. Geske. Composable agents for patient flow control — preliminary concepts. In *Proceedings of the DFG-SPP Workshop "Intelligente Softwareagenten in betriebswirtschaftlichen Anwendungsszenarien"*, pages 223–231. Ilmenau, Germany, 1999.
9. M. Hannebauer and R. Kühnel. Dynamic reconfiguration in collaborative problem solving. In *Proceedings of the Eighth Workshop on Concurrency, Specification and Programming (CS&P-99)*, pages 71–82, Warsaw, Poland, 1999.
10. M. N. Huhns and M. P. Singh, editors. *Readings in Agents*. Morgan Kaufmann Publishers, 1998.
11. N. R. Jennings and M. J. Wooldridge. *Agent Technology — Foundations, Applications, and Markets*. Springer, 1998.
12. Q. Y. Luo, P. G. Hendry, and J. T. Buchanan. Heuristic search for distributed constraint satisfaction problems. Research Report KEG-6-92, Department of Computer Science, University of Strathclyde, Glasgow G1 1XH, UK, 1992.
13. Q. Y. Luo, P. G. Hendry, and J. T. Buchanan. A hybrid algorithm for distributed constraint satisfaction problems. Research Report RR-92-62, Department of Computer Science, University of Strathclyde, Glasgow G1 1XH, UK, 1992.
14. Q. Y. Luo, P. G. Hendry, and J. T. Buchanan. A new algorithm for dynamic constraint satisfaction problems. Research Report RR-92-63, Department of Computer Science, University of Strathclyde, Glasgow G1 1XH, UK, 1992.
15. Q. Y. Luo, P. G. Hendry, and J. T. Buchanan. Comparison of different approaches for solving distributed constraint satisfaction problems. Research Report RR-93-74, Department of Computer Science, University of Strathclyde, Glasgow G1 1XH, UK, 1993.

16. T. Nguyen and Y. Deville. A distributed arc-consistency algorithm. Technical Report 1348, Dépt Informatique, Univ. Cath. de Louvain, Louvain-la-Neuve, Belgium, 1995.
17. V. A. Saraswat. *Concurrent Constraint Programming*. MIT Press, 1993.
18. G. Solotorevsky and E. Gudes. Solving a real-life time tabling and transportation problem using distributed CSP techniques. In *Proceedings of CP-96 Workshop on Constraint Applications*, Cambridge, USA, 1996.
19. G. Solotorevsky, E. Gudes, and A. Meisels. Modeling and solving distributed constraint satisfaction problems (DCSPs). In *Proceedings of the Conference on Constraint-Processing (CP-96)*, 1996.
20. K. P. Sycara, S. F. Roth, N. Sadeh, and M. S. Fox. Distributed constrained heuristic search. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6):1446–1461, 1991.
21. G. Tel. Distributed control algorithms for AI. In G. Weiss, editor, *Multiagent Systems — A Modern Approach to Distributed Artificial Intelligence*, pages 562–569. MIT Press, 1999.
22. D. Waltz. Understanding line drawing of scences with shadows. In P. Winston, editor, *The Psychology of Computer Vision*, pages 19-91. McGraw-Hill, 1975.
23. M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Transactions on Knowledge and DATA Engineering*, 10(5), 1998.
24. M. Yokoo and K. Hirayama. Distributed constraint satisfaction algorithm for complex local problems. In *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS-98)*, pages 372–379, Paris, France, 1998.
25. M. Yokoo and T. Ishida. Search algorithms for agents. In G. Weiss, editor, *Multiagent Systems — A Modern Approach to Distributed Artificial Intelligence*, pages 165–199. MIT Press, 1999.
26. Y. Zhang and A. K. Mackworth. Parallel and distributed algorithms for finite constraint satisfaction problems. In *Proceedings of the IEEE-Symposium on Parallel and Distributed Processing*, pages 394–397, 1991.