Distributing Problem Solving on the Web using Constraint Technology.

Marc Torrens, Rainer Weigel and Boi Faltings

Artificial Intelligence Laboratory

Swiss Federal Institute of Technology in Lausanne (EPFL)

IN-Ecublens, CH-1015 Lausanne, Switzerland

Abstract

WWW servers have to serve many clients simultaneously and thus cannot provide intelligent services. We present an approach where intelligent problem solving is distributed so that compute-expensive tasks are carried out on the client side. To this end, we have implemented a library of constraint satisfaction techniques, called the JAVA constraint library, which allows composing applets that solve CSPs. We present the library and show several examples of applications.

Keywords: Intelligent Internet agents, Problem Solving on the Web, Product Configuration, Constraint-based Reasoning.

1 Distributing Problem Solving

As a result of the spread of the world-wide web, interactive information servers are becoming more and more commonplace. Browsing through databases requires quick response times which are difficult to achieve when users interact directly with a server. Bandwidth restrictions and server overload are the two major problems when users are browsing information directly on a

Web server. In addition, very often information on the web servers are not well-structured to fit certain complex queries.

Consider for example web pages of an airline company. They often provide access to their flight databases by requiring information about departure and destination airports. The result is a list of possible flight connections. A more difficult query where one plans a business roundtrip visiting several cities in an arbitrary order would be much more complicated and cumbersome using the standard interface. This is true in particular if one would like to minimize the hours of flight, costs, etc.

In oder to answer such queries a system must provide problem solving capabilities. Our general approach is the following: The answer of the query is considered to be a solution to a constraint satisfaction problem (CSP). The formalisation as a CSP supports a natural decomposition of the task into two subtasks: generation of the CSP on the server, and solving the CSP on the client. The advantages of the approach are that

1. if there are many solutions to the problem, then the CSP can be regarded as a compact representation of the solution space. It represents it using a minimum amount of information that needs to be send to the client, and

2. finding the solutions requires only computing power of the client and thus avoids server overload. Furthermore, browsing through a large number of possible answers to a query is independent of the server and can be done locally.

The architecture supporting this methodology is shown in Fig. 1. The client sends a request containing the user constraints to the server. The server will access databases in order to generate the corresponding CSP taking into consideration the constraints of the user. The CSP is packaged with search algorithms for finding the solutions to form an agent which is returned to the user. In this way the user can browse through the different solutions by interacting with the agent locally.

We decompose the process into two parts:

- the information server compiles all relevant information from the database and the user constraints (query) into the corresponding CSP. The CSP is a compact representation of all solutions that the problem can have given the initial restrictions of the user.

- the server sends an agent consisting of the CSP and search algorithms to the client. This allows the user to browse through all the possible solutions. Since the agent executes on the client, response time can be very fast and the user can compare different alternatives without placing unnecessary load on the server.

Building the CSP requires only a small fraction of time compared to solving the CSP, so having the agent executed on the client significantly reduces server overload. After having generated the CSP there is no longer need of accessing the server, so the agent sent by the server is completely *autonomous*. This architecture is shown in Fig. 1.

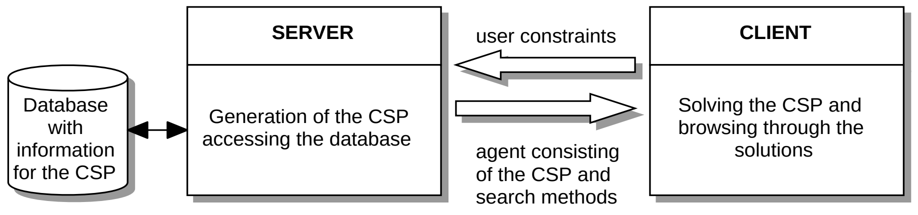

SERVER
user constraints
CLIENT
Database with information for the CSP
Generation of the CSP accessing the database
Solving the CSP and browsing through the solutions
agent consisting of the CSP and search methods

Figure 1: *A client-server architecture for distributing problem solving on the Web.*

In the next Section we introduce the Java Constraint Library (JCL) that supports our architecture described above.

2 Java Constraint Library

We implemented the Java Constraint Library (JCL) which allows us to package constraint satisfaction problems and their solvers in agents on the Internet. We will first give a brief introduction to constraint satisfaction techniques and then describe JCL. Finally we describe the **JCL shell**, a tool to solve CSPs on the Internet by means of a Java *applet*, and present a resource allocation problem to illustrate the usage of JCL.

2.1 Constraint Satisfaction Problems (CSPs)

CSPs are ubiquitous in applications like configuration [10, 9], planning [12, 7, 3], resource allocation [1, 11], scheduling [8, 2], timetabling [6] and many others. A CSP is specified by a set of variables and constraints among them. A solution to a CSP is a set of value assignments to all variables such that all constraints are satisfied. There can be either many, 1 or no solutions. The main advantages of constraint based programming are as following:

- It offers a general framework within which many real world problems can be stated in a succinct and elegant way.
- A constraint based representation can be used to synthesize solutions of the problem as well as for verification purposes (i.e. showing that a solution satisfies all constraints).
- The nature of the representation allows a formal description of the problems as well as a declarative description of search heuristics.

A finite, discrete Constraint Satisfaction Problem (CSP) is defined by a tuple $P = (X, D, C, R)$ where $X = \{X_1, \ldots, X_n\}$ is a finite set of variables, each associated with a domain of discrete values $D = \{D_1, \ldots, D_n\}$, and a set of constraints $C = \{C_1, \ldots, C_l\}$. Each constraint C_i is expressed by a relation R_i on some subset of variables. This subset of variables is called the *connection* of the constraint and denoted by $con(C_i)$. The relation R_i over the connection of a constraint C_i is defined by $R_i \subseteq D_{i1} \times \ldots \times D_{ik}$ and denotes the tuples that satisfy C_i. The *arity* of a constraint C is the size of its connection. In particular, a constraint is called *binary* if it is between 2 variables (the size of its connection is equal to 2). If all constraints are binary, then the CSP is called binary. Otherwise the CSP is called a *non-binary* CSP.

A solution of a CSP P is a compound label of size n such that all constraints are satisfied simultaneously. A CSP is said to be satisfiable if a solution exists. Depending on the goals one wants to satisfy, one can either try to show that a CSP is satisfiable, i.e., show that a solution exists, find all solutions of a CSP, or find some optimal solutions, where optimality can be defined according to specified domain criteria. A large body of techniques exists for efficiently solving CSPs. For more details see [14].

2.2 The Constraint Library

The library provides services for:

- creating, managing and representing binary discrete CSPs
- applying search and preprocessing algorithms to CSPs

JCL can be used either in a Java enabled browser (applet) or in a stand-alone Java application. In Fig. 2 we present the main components of the JCL environment. The purpose of JCL is to provide a framework for easily building agents that solve CSPs on the Web. JCL is divided into two parts: A basic constraint library available on the network and a constraint shell build on the top of this library, allowing CSPs to be edited and solved. JCL allows the development of portable applications and *applets* using the constraint mechanisms. JCL can be downloaded from `http://liawww.epfl.ch/~torrens`. For more details see [13].

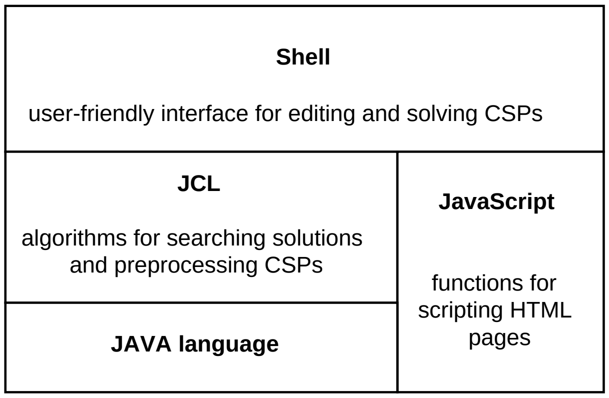
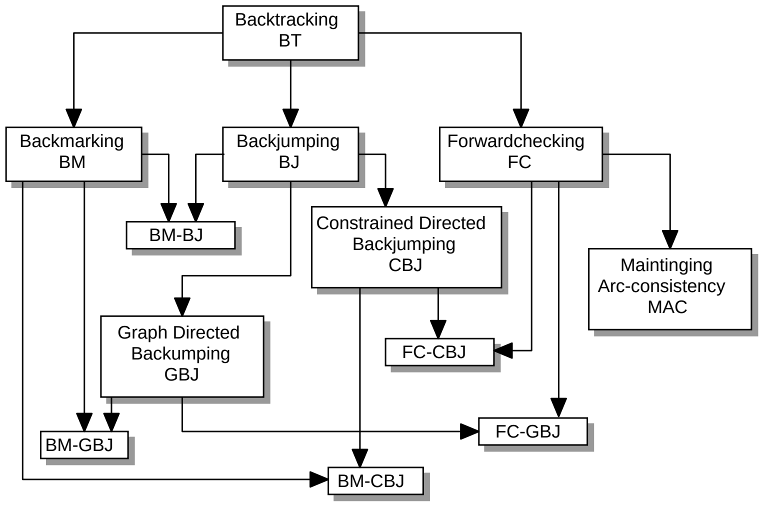

Shell
user-friendly interface for editing and solving CSPs
JCL
algorithms for searching solutions and preprocessing CSPs
JAVA language
JavaScript
functions for scripting HTML pages

Figure 2: *The components of the JCL environment.*

The algorithms

The library contains search and preprocessing algorithms for CSPs. The search algorithms allow us to find the solutions of a CSP, while the preprocessing algorithms are used to simplify a

CSP by eliminating values and compound labels that do not affect its solutions. Several search algorithms are implemented in JCL. There are three main algorithms derived from *Chronological Backtracking* (BT) that are: *Backmarking* (BM), *Backjumping* (BJ) and *Forward Checking* (FC) [4]. Some combinations of them are implemented in [15] and adapted in JCL. Fig. 3 shows a hierarchy of the algorithms in JCL.



Backtracking BT
Backmarking BM
Backjumping BJ
Forwardchecking FC
BM-BJ
Constrained Directed Backjumping CBJ
Maintinging Arc-consistency MAC
Graph Directed Backumping GBJ
FC-CBJ
FC-GBJ
BM-GBJ
BM-CBJ

Figure 3: *A hierarchy of the search algorithms implemented in JCL.*

The following two preprocessing algorithms are implemented in JCL: *Arc-consistency* (AC) and *Path-consistency* (PC) [5].

2.3 the JCL Shell

The purpose of the shell is to provide a user-friendly interface to the library. The JCL shell can be executed as a stand-alone application or as an applet. The following aspects are taken into consideration:

- CSP definition and generation from scratch,
- algorithm application and
- intermediate and final solution management.

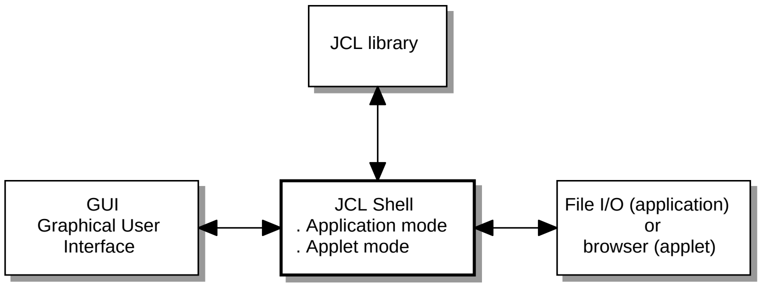
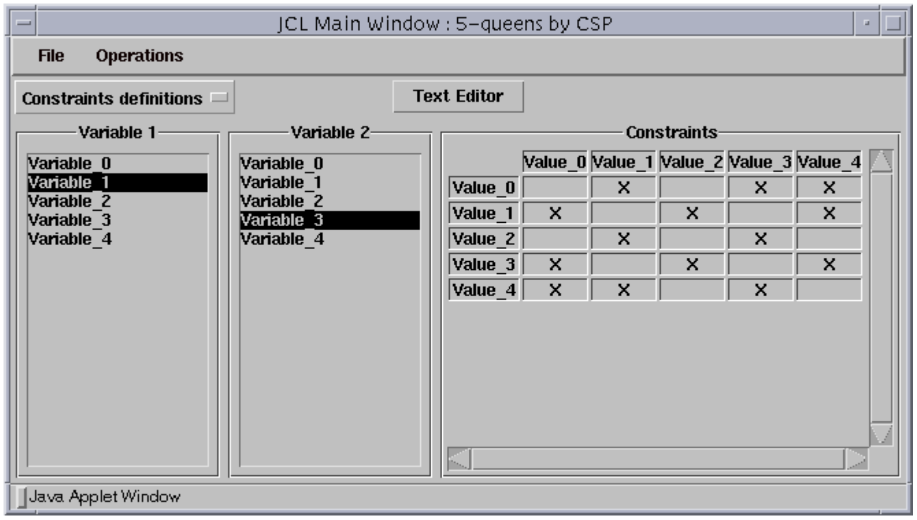


Figure 4: *Relations between JCL library, JCL shell and the external world.*



Variable 1: Variable_0, Variable_1, Variable_2, Variable_3, Variable_4

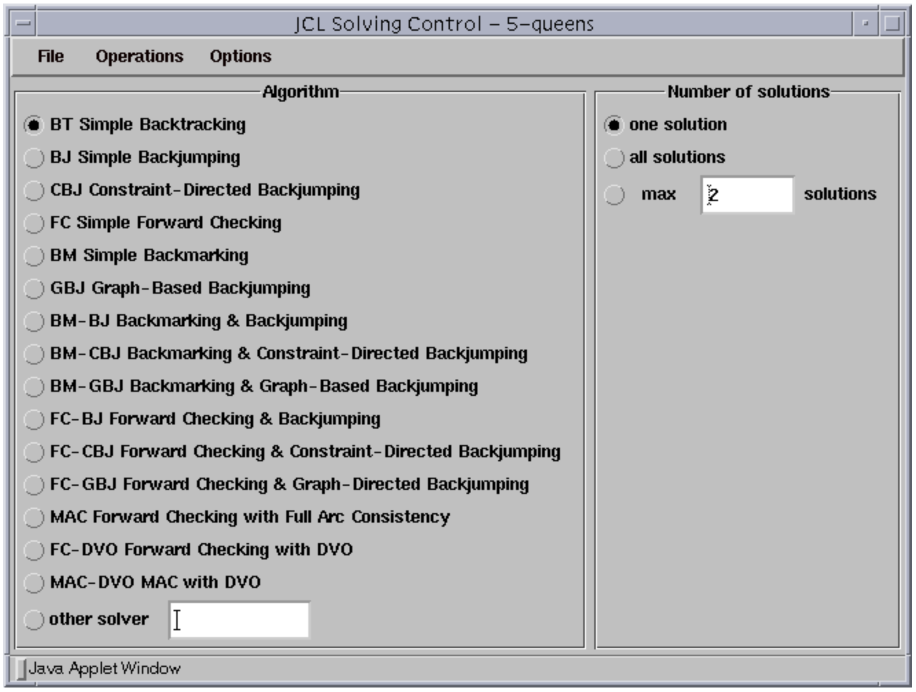
Variable 2: Variable_0, Variable_1, Variable_2, Variable_3, Variable_4

	Value_0	Value_1	Value_2	Value_3	Value_4
Value_0		X		X	X
Value_1	X		X		X
Value_2		X		X	
Value_3	X		X		X
Value_4	X	X		X	

Figure 5: *The constraints editor in the JCL shell.*

Fig. 4 illustrates the relations between the library, the shell and the external world.

Fig. 5 shows how constraints in between variables can be edited using mouse and menus. We can select a pair of variables and then mark the allowed combination of values defining the constraint. Another important window is the *solving control* window shown in Fig. 6. It allows the user choose the algorithm, solution options, displaying options and start the algorithm. The HTML output produces the output in a browser window. The *algorithm panel* permits algorithm selection among the JCL algorithms or other algorithms that could be implemented by the user. While the algorithms are running, a "solving in progress" window is displayed indicating how many solutions have been found so far. In this window one can also suspend, resume or stop the resolution process.



JCL Solving Control – 5-queens

File Operations Options

Algorithm

- [● BT Simple Backtracking]
- BJ Simple Backjumping
- CBJ Constraint-Directed Backjumping
- FC Simple Forward Checking
- BM Simple Backmarking
- GBJ Graph-Based Backjumping
- BM-BJ Backmarking & Backjumping
- BM-CBJ Backmarking & Constraint-Directed Backjumping
- BM-GBJ Backmarking & Graph-Based Backjumping
- FC-BJ Forward Checking & Backjumping
- FC-CBJ Forward Checking & Constraint-Directed Backjumping
- FC-GBJ Forward Checking & Graph-Directed Backjumping
- MAC Forward Checking with Full Arc Consistency
- FC-DVO Forward Checking with DVO
- MAC-DVO MAC with DVO
- other solver

Number of solutions

- [● one solution]
- all solutions
- max 2 solutions

Java Applet Window

Figure 6: *The solving control window.*

2.4 A simple graphical application using JCL

We describe an *applet* for solving Resource Allocation problems. It is a simple application to demonstrate the flexibility and adaptability of the JCL for programming applications using CSP mechanisms on the Web. Resource Allocation (RA) problems can be defined as follows (see [1]): *"Given a set of tasks with fixed endpoints, and given, for each task, a set of resources that can carry out the task, assign one resource to each of the tasks such that no resource is assigned to two different tasks at the same time"*. The *RA* problem can be modeled as a discrete binary CSP where the variables are the tasks, the values are the resources and there are constraints of mutual exclusion between two variables if the corresponding tasks intersect in time. Consider the interval representation of the small example in Fig. 7, where we have 7 tasks $T_1, \ldots, T_7$ and resources $R_1, \ldots, R_5$. Two tasks intersecting in time are not allowed to use the same resource.

The applet of solving *RA* problems can be executed from the Web at: `http://liawww.epfl.ch/~torrens/exercise/exercise.html`. The tasks to be executed are represented as rectangles, and rectangles intersecting horizontally do intersect in time. Each

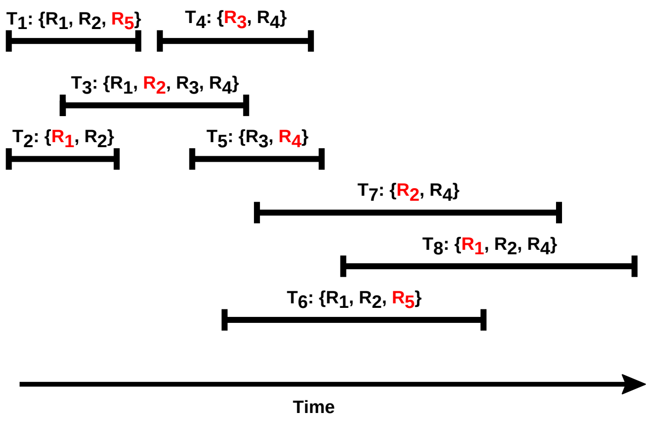
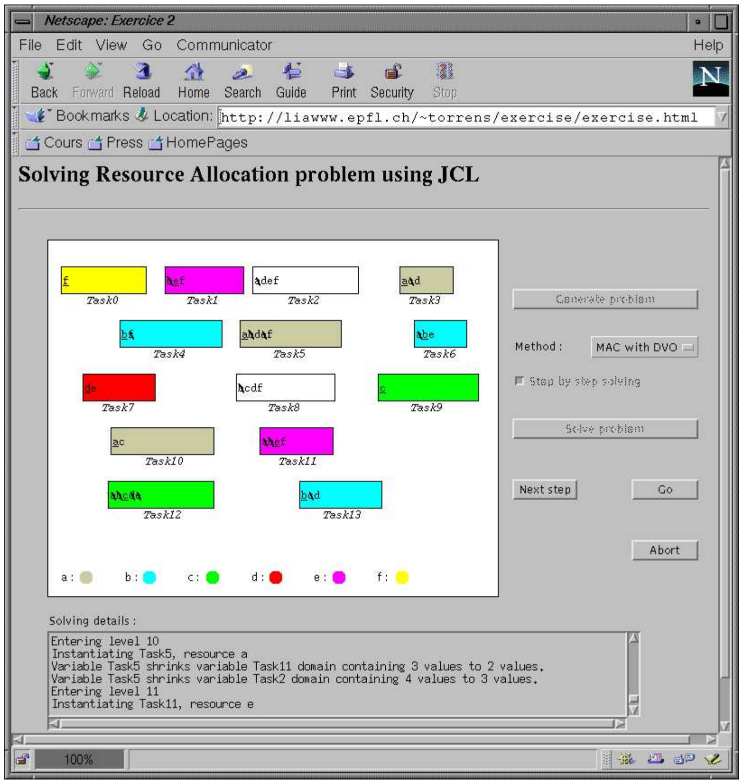

T_1: {R_1, R_2, R_5}
T_4: {R_3, R_4}
T_3: {R_1, R_2, R_3, R_4}
T_2: {R_1, R_2}
T_5: {R_3, R_4}
T_7: {R_2, R_4}
T_8: {R_1, R_2, R_4}
T_6: {R_1, R_2, R_5}
Time

Figure 7: *Interval orders: a schedule of seven tasks whose start time and duration are fixed. For each task, a set of possible resources is shown. A correct assignment of resources to the tasks is illustrated in red.*

resource corresponds to a different color. When a resource is assigned to a task then the task is colored with the color of the resource. The applet allow the user to generate random Resource Allocation problems, select a search algorithm, and solve the CSP. A step-by-step mode allows a user to trace the execution of the algorithms. In Fig. 8 we show the appearance of this applet.


Netscape: Exercice 2
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: http://liawww.epfl.ch/~torrens/exercise/exercise.html
Cours Press HomePages
Task0 Task1 Task2 Task3 Task4 Task5 Task6 Task7 Task8 Task9 Task10 Task11 Task12 Task13
a: b: c: d: e: f:
Generate problem
Solve problem
Next step
Go
Abort
100%

Solving Resource Allocation problem using JCL

Method : MAC with DVO

Step by step solving

Solving details :

```
Entering level 10
Instantiating Task5, resource a
Variable Task5 shrinks variable Task11 domain containing 3 values to 2 values.
Variable Task5 shrinks variable Task2 domain containing 4 values to 3 values.
Entering level 11
Instantiating Task11, resource e
```

Figure 8: *Resource Allocation applet searching a solution.*

3 Applications

3.1 Air Travel Planning (ATP) system

From time to time, we are all faced with the problem of arranging business trips. Typically, we have to meet with a set of people in different cities, each of which has certain days where they are available for a meeting. Transportation schedules impose additional constraints. In the current state of affairs, schedule information can only be obtained by queries to travel agents or WWW servers for particular routes, dates and times. Thus, finding the optimal plan would require separate queries for every part of every alternative itinerary. Since each query implies response times on the order of 1 minute, this makes travel planning very tedious. A better solution is to use a client server architecture distributing problem solving on the Web.

The prototypical business Air Travel Planning (ATP) system is designed to facilitate arranging these kinds of business trips using JCL. An *air travel plan* is a sequence of flights connecting different cities a user plans to visit. Given such a set of cities together with possible time slots to visit each city, the system generates a set of plans. This plan consists of the meeting together with all possible flight connections from which the user can easily select the most preferred one.

The input data for ATP system is a set of meetings, where every meeting is described by the place and the possible time-slots for the different days the meeting could take place (see Table 1 and Fig. 9). We formulate the problem of finding a travel plan as a binary CSP. For each meeting M_i there exists a constraint variable and the domain are the flights in between cities where the meetings could take place. The constraints require that flights are available such that the person can attend all the meetings. They can roughly be formulated as follows: a *flight action* from meeting j to meeting i can be accepted if meeting j finishes before the departure time and the plane arrives before meeting i starts. A solution of the CSP corresponding to the business travel problem can be seen as a sequence of flights in between the cities of the meetings. For each meeting, one of the possible days must be assigned and it must be guaranteed that then exists at least one flight connection between consecutive meetings. Consider as example the travel data

M	City	Time-Slots for November		
M1	AMS	1^{st} 12h-16h	3^{th} 13h-15h	
M2	BCN	1^{st} 12h-15h	2^{nd} 13h-17h	
M3	LON	2^{nd} 12h-15h	8^{th} 11h-14h	
M4	GVA	2^{nd} 9h-12h	4^{th} 9h-12h	5^{th}10h-15h
M5	PAR	5^{th} 8h-12h	8^{th} 8h-12h	
M6	BER	6^{th} 15h-18h	8^{th} 10h-16h	
M7	FRA	4^{th} 8h-12h	7^{th} 8h-12h	

Table 1: *Input Data to be send to the server.*

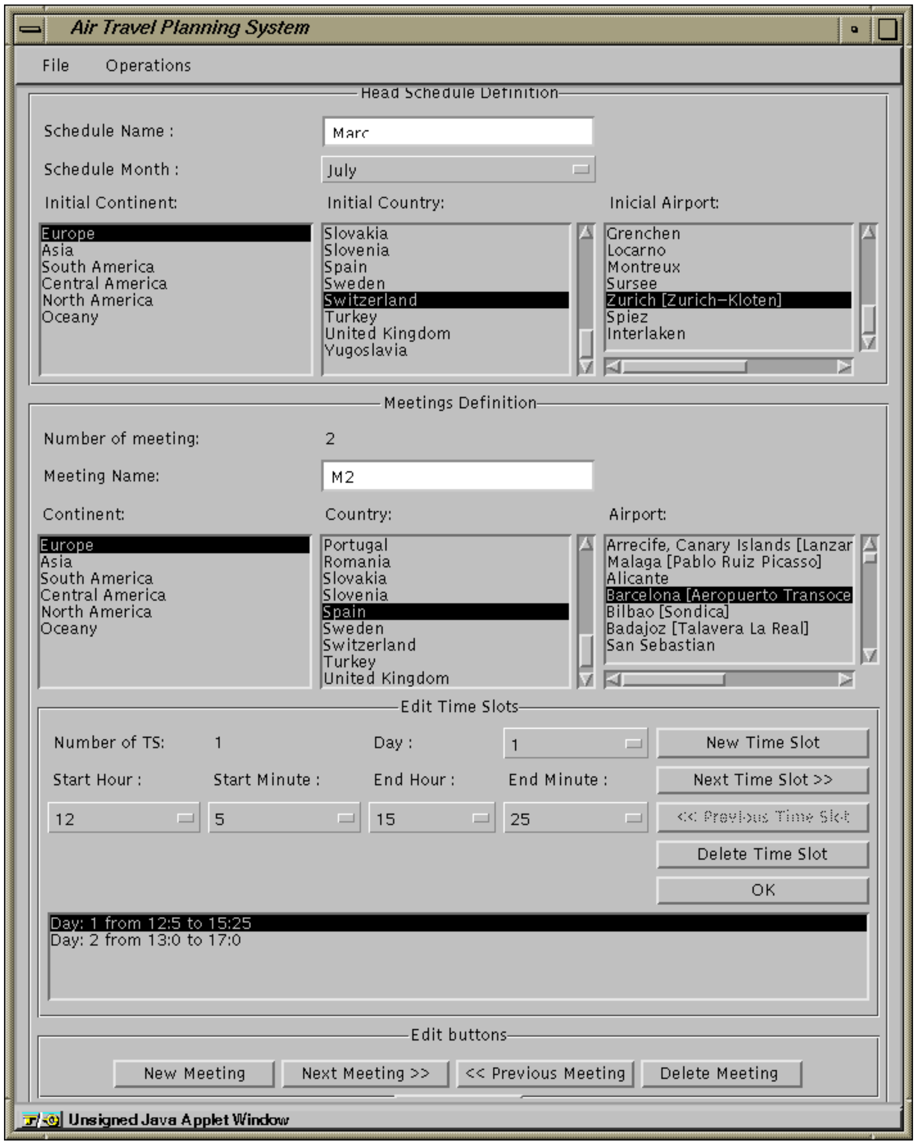
from Table 1.

In the following we present a new methodology for solving the business travel problem on the Web using JCL. The main idea is to generate the CSP corresponding the business travel problem on the server and solve the problem locally on the client (see Fig. 1). The input required from the traveler includes all possible meeting slots. The input is sent to the server in order to build the corresponding CSP, taking flight databases and the user input into account. Then the CSP is packaged together with search algorithms from JCL into an autonomous agent. The CSP can then be solved on the client without having to access the server. This allows the user to browse autonomously through the different solutions. When the user selects a travel plan (a solution) then the actual flights can be presented in the form of a list such that the user can easily select the most preferred one. The example below describes some possible partial solution of the problem shown in Table 1:

- meeting $M1$ (Amsterdam) is scheduled on the 3^{rd} (from 13h to 15h)
- meeting $M2$ (Barcelona) is scheduled on the 2^{nd} (from 13h to 17h)
- meeting $M4$ (Geneva) is scheduled on the 4^{th} (from 9h to 12h)
- meeting $M5$ (Paris) is scheduled on the 5^{th} (from 8h to 12h)

The corresponding flights to this partial solution are shown in Table 2. In order to access flight data for building the CSP, we created a MiniSQL[1] database. We use a Java class library called

[1]Reference at http://Huges.com.au



Head Schedule Definition

Schedule Name : Marc

Schedule Month : July

Initial Continent:	Initial Country:	Inicial Airport:
Europe	Slovakia	Grenchen
Asia	Slovenia	Locarno
South America	Spain	Montreux
Central America	Sweden	Sursee
North America	Switzerland	Zurich [Zurich–Kloten]
Oceany	Turkey	Spiez
	United Kingdom	Interlaken
	Yugoslavia	

Meetings Definition

Number of meeting: 2

Meeting Name: M2

Continent:	Country:	Airport:
Europe	Portugal	Arrecife, Canary Islands [Lanzar
Asia	Romania	Malaga [Pablo Ruiz Picasso]
South America	Slovakia	Alicante
Central America	Slovenia	Barcelona [Aeropuerto Transoce
North America	Spain	Bilbao [Sondica]
Oceany	Sweden	Badajoz [Talavera La Real]
	Switzerland	San Sebastian
	Turkey	
	United Kingdom	

Edit Time Slots

Number of TS: 1 Day : 1

Start Hour :	Start Minute :	End Hour :	End Minute :
12	5	15	25

Day: 1 from 12:5 to 15:25
Day: 2 from 13:0 to 17:0

Figure 9: *The input data window for the ATP system.*

MsqlJava[2] which allows applications or applets to access and manipulate MiniSQL databases. On the server the MiniSQL is running in background. This makes it possible for the applet on the client to access the flight database to build the corresponding CSP. Table 3 describes some rows of the created database.

[2]Reference at http://mama.minet.uq.oz.au/msqljava

Comp	Fly	From	To	Dep	Arr	Dur
IB	4248	BCN	AMS	10:15	12:25	2:10
KL	352	BCN	AMS	7:05	9:25	2:20
SR	724	GVA	PAR	12:15	13:20	1:05
AF	2855	GVA	PAR	14:10	15:15	1:05
SR	726	GVA	PAR	16:15	17:20	1:05
AF	2835	GVA	PAR	17:15	18:20	1:05
AF	2893	GVA	PAR	18:05	19:10	1:05
SR	728	GVA	PAR	18:45	19:50	1:05
AF	2887	GVA	PAR	20:40	21:45	1:05

Table 2: *Possible flights from Barcelona on 2nd (after 17h) to Amsterdam on 3rd (before 13h) and from Geneva on 4th (after 12h) to Paris on 5th (before 8h).*

Code	Flight	Dep.	Time	Ar.	Time	Days
IB	4248	BCN	1015	AMS	1225	1234567
KL	354	BCN	1125	AMS	1345	1234567
KL	356	BCN	1610	AMS	1830	1234567
IB	4262	BCN	1640	AMS	1855	1234567

Table 3: *Description of some rows of the flight database.*

3.2 Product Configuration for Electronic Commerce

In recent years, manufacturing trends have changed from pure mass-production towards a more customer oriented one-of-a-kind production. The main reason for this change is that today's customers have very specific and individual requirements which cannot be satisfied by mass-products. The one-of-a-kind production of many consumer and investment products imposes new challenges concerning the marketing of these products, in particular in Electronic Commerce. With current electronic catalogs, customers are supposed to compose solutions themselves by selecting elements individually. In the future, *product configuration* utilities that synthesize products according to customer's wishes will be indispensable for selling multi-variant products through Electronic Commerce, and will be an essential part of more intelligent electronic catalogs.

The general configuration task can be defined as follows: Given

- a set of predefined components,

- the knowledge of how components can be connected,
- the customer requirements for a specific configuration

find the sets of components fulfilling the user-requirements and respecting all the compatibility constraints. The configuration task can be formalized as a general constraint satisfaction problem (CSP). Two approaches for solving configuration problems represented as CSPs exist:

Standard approach: Using this technique one needs to solve the CSP corresponding to the configuration problem from scratch. That is the customer inputs the requirements and then the constraint solver tries to find the solutions. There are three major problems with this approach. The first is that user requirements may lead to an over-constrained CSP and thus no product can be configured. The second is that there are too many possible products satisfying the requirements and the customer is overloaded with information. The third and most serious problem is that often a user has only a vague idea about the product and cannot really express the input requirements.

Case-based approach: This approach allows the customer to select a configuration from a set of configurations sold to earlier (possibly fictitious) customers. Then the selected configuration can interactively be modified until a product satisfying all the requirements is found. The case-based method is especially well-suited when there are only a few "standard" products which could be represented in a catalog and when constraint based adaptation methods to modify these standard products are available. This approach avoids all the problems mentioned above.

In both cases, the JCL allows executing the computationally expensive parts of the problem on the customer side. Fig. 10 shows the resulting architecture for the case-based approach.

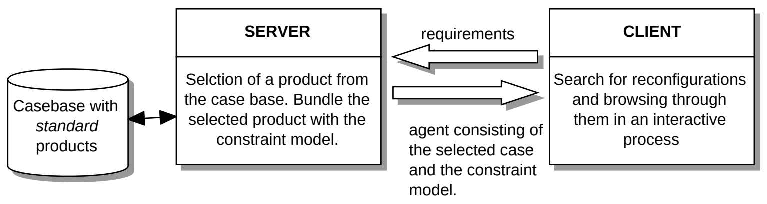

SERVER
Selction of a product from the case base. Bundle the selected product with the constraint model.
CLIENT
Search for reconfigurations and browsing through them in an interactive process
requirements
agent consisting of the selected case and the constraint model.
Casebase with *standard* products

Figure 10: *A client server architecture for distributing product configuration on the Web.*

4 Conclusions

As a result of the spread of the world-wide web, interactive information servers are becoming more and more important. Browsing through databases requires quick response times which are difficult to achieve when users interact directly with a server. We have shown a methodology when agent techniques underlying Java can separate browsing from database access. The key element of this approach is to represent solution spaces of a problem as a CSP. This CSP will be bundled with search engines from JCL that allow the problem to be solved locally on the client without requiring computation resources from the server.

References

[1] Berthe Y. Choueiry. *Abstraction Methods for Resource Allocation.* PhD thesis, Swiss Federal Institute of Technology in Lausanne, 1994.

[2] Mark Fox. Why is Scheduling Difficult? A CSP Perspective. In *Proc. of the 9 th ECAI*, pages 754–758, Stockholm, Sweden, 1990.

[3] Matthew L. Ginsberg. A new algorithm for generative planning. In Luigia Carlucci Aiello, Jon Doyle, and Stuart Shapiro, editors, *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning*, pages 186–197, San Francisco, November 5–8 1996. Morgan Kaufmann.

[4] Grzegorz Kondrak. A Theoretical Evaluation of Selected Backtracking Algorithms. Technical Report TR-94-10, Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada, 1994.

[5] Alan K. Mackworth. Consistency in Networks of Relations. *Artificial Intelligence*, 8:99–118, 1977.

[6] Amnon Meisels, Ehud Gudes, and Gadi Solotorevsky. Employee timetabling, constraint networks and knowledge-based rules: a mixed approach. In *Proceedings of the First International Conference on the Practice and Theory of Automated Timetabling (ICPTAT '95)*, pages 504–510, 1995.

[7] Claude Le Pape. Constraint propagation in planning and scheduling. Technical report, Stanford University, 1991.

[8] Patrick Prosser. Scheduling as a Constraint Satisfaction Problem: Theory and Practice. In *Scheduling of Production Processes Workshop Notes, W7, ECAI-92*, pages 7–15, Vienna, Austria, 1992.

[9] Daniel Sabin and Eugene C. Freuder. Configuration as composite constraint satisfaction. In *Proceedings of the Artificial Intelligence and Manufacturing Research Planning Workshop*, pages 153–161, 1996.

[10] Felix Freyman Sanjay Mittal. Towards a generic model of configuration tasks. In *Proc. of the 11 th IJCAI*, pages 1395–1401, Detroit, MI, 1989.

[11] A. Sathi and M. S. Fox. Constraint-directed negotiation of resource allocations. In L. Gasser and M. Huhns, editors, *Distributed Artificial Intelligence Volume II*, pages 163–194. Pitman Publishing: London and Morgan Kaufmann: San Mateo, CA, 1989.

[12] Mark Stefik. Planning with constraints (molgen: Part 1). *Artificial Intelligence*, 16(2):111–140, 1981.

[13] Marc Torrens, Rainer Weigel, and Boi V. Faltings. Java Constraint Library: bringing constraints technology on Internet using Java language. In *Working Notes of the Workshop on Constraints and Agents, Technical Report WS-97-05, AAAI-97*, Providence, Rhode Island, USA, 1997.

[14] Edward Tsang. *Foundations of Constraint Satisfaction.* Academic Press, London, UK, 1993.

[15] Peter van Beek. *CSPLib : a CSP library written in C language.* University of Alberta, vanbeek@cs.ualberta.