Paweł SITEK, Jarosław WIKAREK*

IMPLEMENTATION OF DECLARATIVE FRAMEWORK FOR DECISION SUPPORT SYSTEM IN SCHEDULING PROBLEMS

Abstract

Scheduling problems appear frequently at different levels of decisions. They are usually characterized by many types of constraints, which make them unstructured and difficult to solve (NP-complete). Traditional mathematical programming approaches are deficient because their representation of constraints is artificial (using 0-1 variables). Unlike traditional approaches, constraint logic programming (CLP) provides for a natural representation of heterogeneous constraints. In CLP we state the problem requirements by constraints; we do not need to specify how to meet these requirements. In this paper we propose a declarative framework for decision support system (DSS) for constrained search problems implemented by CLP and relational SQL database. We illustrate this concept by the implementation of a DSS for scheduling problems with external resources in different production organization environments.

1. INTRODUCTION

Today's highly competitive business environment makes it an absolute requirement on behalf of the managers to continuously make the best decisions in the shortest possible time. *Learning from mistakes* has left its place to *one strike and you're out'* reality. That is, there is no room for mistake in making decisions in this global environment. Success depends on quickly allocating the organizational resources towards meeting the actual needs and wants of the customer. Decision problems involve various numeric and non-numeric constraints, some of which are conflicting with each other. Occasionally, decision-makers do not have complete information on the situation. Thus they perform 'what-if' and goal-seeking analyses involving constraints. In order to succeed in such an unforgiving environment, managers and decision makers need integrated 'intelligent' decision support systems (DSS) that are capable of using a wide variety of models along with data and information resources available to them at various internal and external repositories. An important aspect of decision support systems studies is to develop techniques for automatic or interactive decision analysis in a complex real-world situation.
In this paper we present the use of constraint logic programming as a tool for such decision support systems in constrained search problems, focusing on the model representation and analyses. Constraint Logic Programming (CLP) is a declarative modelling and procedural

* dr inż, dr inż,, Technical University of Kielce, Control and Management Systems Section, 1000-PP 7, Kielce, Poland, E-MAIL: sitek@tu.kielce.pl, j.wikarek@tu.kielce.pl

programming environment that integrates qualitative /heuristic knowledge representation of logic and quantitative/algorithmic reasoning into single paradigm.
The original contribution of our approach consists of a declarative framework for scheduling problems, developed within the constraint logic paradigm together with relational SQL database, and the development of a constraint logic solver for scheduling problems with external resources in different production organization environments.

2. DECLARATIVE PROGRAMMING AND ENVIRONMENTS – SQL, CLP

Declarative programming is a term with two distinct meanings, both of which are in current use. According to one definition, a program is "declarative" if it describes *what* something is like, rather than *how* to create it. For example, HTML, XML web pages are declarative because they describe *what* the page should contain — title, text, images — but not *how* to actually display the page on a computer screen. This is a different approach from imperative programming languages such as Pascal, C, and Java, which require the programmer to specify an algorithm to be run. In short, imperative programs explicitly specify an algorithm to achieve a goal, while declarative programs explicitly specify the goal and leave the implementation of the algorithm to the support software (for example, an SQL select statement specifies the properties of the data to be extracted from a database, not the process of extracting the data).
According to a different definition, a program is "declarative" if it is written in a purely functional programming language, logic programming language, or constraint programming language. The phrase "declarative language" is sometimes used to describe all such programming languages as a group, and to contrast them against imperative languages.
These two definitions overlap somewhat. In particular, constraint programming and, to a lesser degree, logic programming, focus on describing the properties of the desired solution (the *what*), leaving unspecified the actual algorithm that should be used to find that solution (the *how*). However, most logic and constraint languages are able to describe algorithms and implementation details, so they are not strictly declarative by the first definition.
Constraint Logic Programming as a declarative modeling and procedural programming environment is increasingly realized as an effective tool for decision support systems [4, 5, 6]. CLP is suitable for Decision Support Systems (DSS) because [1, 5]:

- CLP is a very good tool for the development of knowledge base that has expertise and experience represented in terms of logic, rules and constraints. This tool allows the knowledge base to be built in an incremental and accumulating way (it is suitable for ill-structured or semi-structured decision analysis problems).
- Constraints naturally represent decisions and their inter-dependencies. Decision choices are explicitly modeled as the domains of constraint variables.
- CLP can serve as a good integrative environment for the decision analysis that has different kinds of model.
- Decision analysis requires a number of computational facilities which this tool can provide.

3 CONCEPT OF DSS BASED ON DECLARATIVE PROGRAMMING FOR SCHEDULING PROBLEMS

The presented in (2) advantages and possibilities of declarative programming environment for decision support make it interesting for decision support in SMEs. The decision support system for production scheduling has been presented as an example of implementation of DSS with declarative programming. Building decision support system for scheduling, covering a variety of production organization forms, such as job-shop, flow-shop, project, multi-project etc., is especially interesting. The following assumptions were adopted in order to design the presented scheduling processes of the decision support system (see Fig.1):

- The system should possess data structures in relational model that make its use possible in different production organization environments
- The system should make it possible to schedule the whole set of tasks simultaneously, and after a suitable schedule has been found, it should be possible to add a new set of tasks later, and to find a suitable schedule for both sets without the necessity to change initial schedules.
- The system should regard:
  - Additional (external) resource types apart from machines, e.g. people, tools, etc.
  - Temporary inaccessibility of all resource types.
  - The processing times dependent on the starting time of jobs, allocated additional resources, etc.
- The decisions of the systems are the answers to appropriate questions formed as CLP predicates.

The range of the decisions made by the system depends on data structures and asked questions. Thus, the system is very flexible as it is possible to ask all kinds of questions (write all kinds of predicates). In this version of DSS the questions which can be asked are the following:

- What is the minimum number of people necessary for assigned makespan and proper schedule? (predicate *opc_d(L,C)*).
- What is the minimum makespan at the assigned number of people and proper schedule? (predicate *opc_g(L,C)*).
- Is it possible to order new tasks (both orders and projects) for the determined makespan? (predicate *opc_s(L,C)*).
- What is minimum makespan at the assigned number of people for new tasks? (predicate *opcd_g(L,C)*).
- What is the minimum number of people necessary for assigned makespan for new tasks? (without changing the schedule of basic set of tasks) (predicate *opcd_d(L,C)*).
- Is it possible to order tasks for the determined makespan ? (predicate *opcd_s(L,C)*).
- Is it possible to order tasks for the determined makespan where the processing time of job depends on allocated number of people? (predicate *opcd_s1(L,C)*).

$C=C_{max}$-makespan, L- manpower

These questions are just examples of questions that the present system can be asked. New questions are new predicates that need to be created in CLP environment. Two types of questions are asked in the system (Fig.1):

- About the existence of the solution (eg., is it possible to carry out a new task in the particular time?, etc.)
- About a particular kind of the solution: find a suitable schedule fulfilling the performance index, find the minimum scheduling length-makespan, find the minimum number of people to carry out the task, etc.

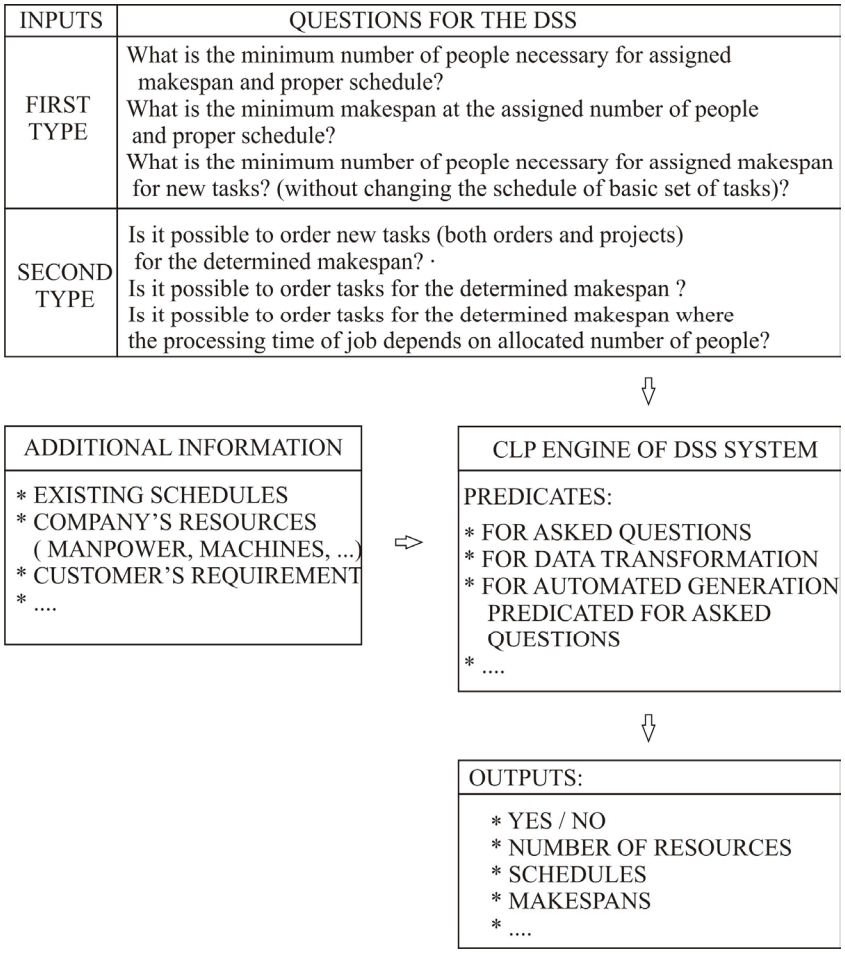

Fig.1 Concept of DSS based on declarative programming for scheduling problems

4 IMPLEMENTATION OF DSS WITH DELCARATIVE PROGRAMMING FRAMEWORK

We propose ECL^{i}PS^e [9] as a platform to decision support in scheduling problems. ECL^{i}PS^e is a software system - based on the CLP paradigm - for the development and deployment of constraint programming applications. It is also ideal for developing aspects of combinatorial problem solving, e.g. problem modeling, constraint programming, mathematical programming, and search techniques. Its wide scope makes it a good tool for research into hybrid problem solving methods. ECL^{i}PS^e comprises several constraint solver libraries, a high-level modeling and control language, interfaces to third-party solvers, an integrated development environment and interfaces for embedding into host environment. The ECL^{i}PS^e programming language is largely backward-compatible with Prolog and supports different dialects. It provides, however, an extended set of basic data types (byte strings, unlimited precision integer and rational numbers, double precision floats and double precision intervals).
Data structures were designed in such a way that they could be easily used to decision problems in a variety of scheduling environments, which is job-shop, flow-shop, project or multi-project. The obtained flexibility resulted from the use of relational data model. The implementation framework is shown in fig.2.

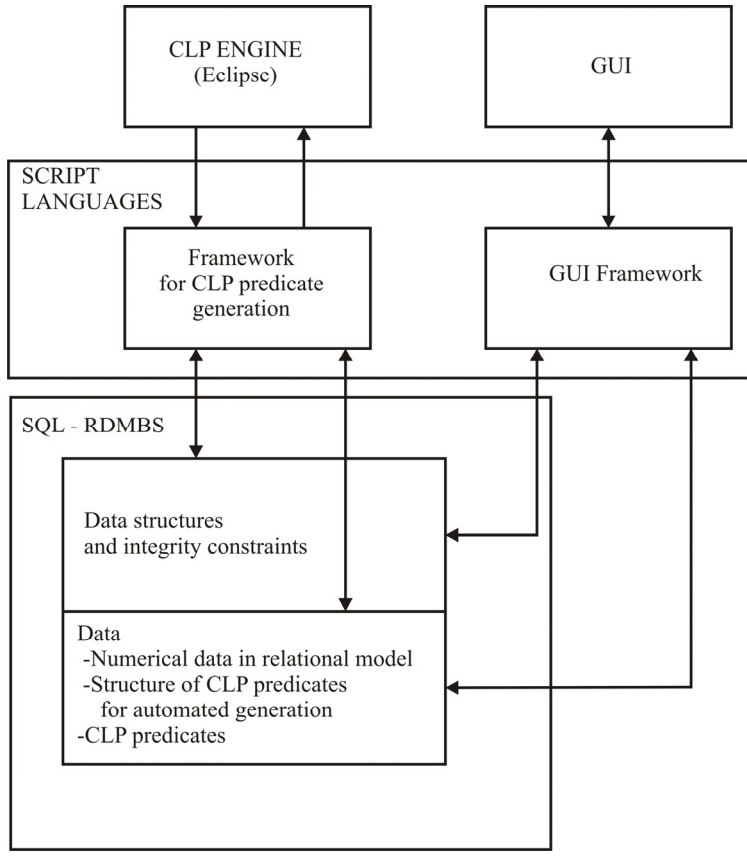

Fig.2 Implementation framework of DSS

The novelty of the proposed approach is in the integration of the CLP methodology with a commonly used relational database model. The scripts started by a CLP engine are generated automatically on the basis of data in the database (numerical values and CLP predicates). The proposed solution makes it possible to easily develop the system (developing and saving in the database the content of additional CLP predicates) and to integrate it with other computer systems based on a relational SQL database. Description of the schema of DSS database has been shown in table 1.

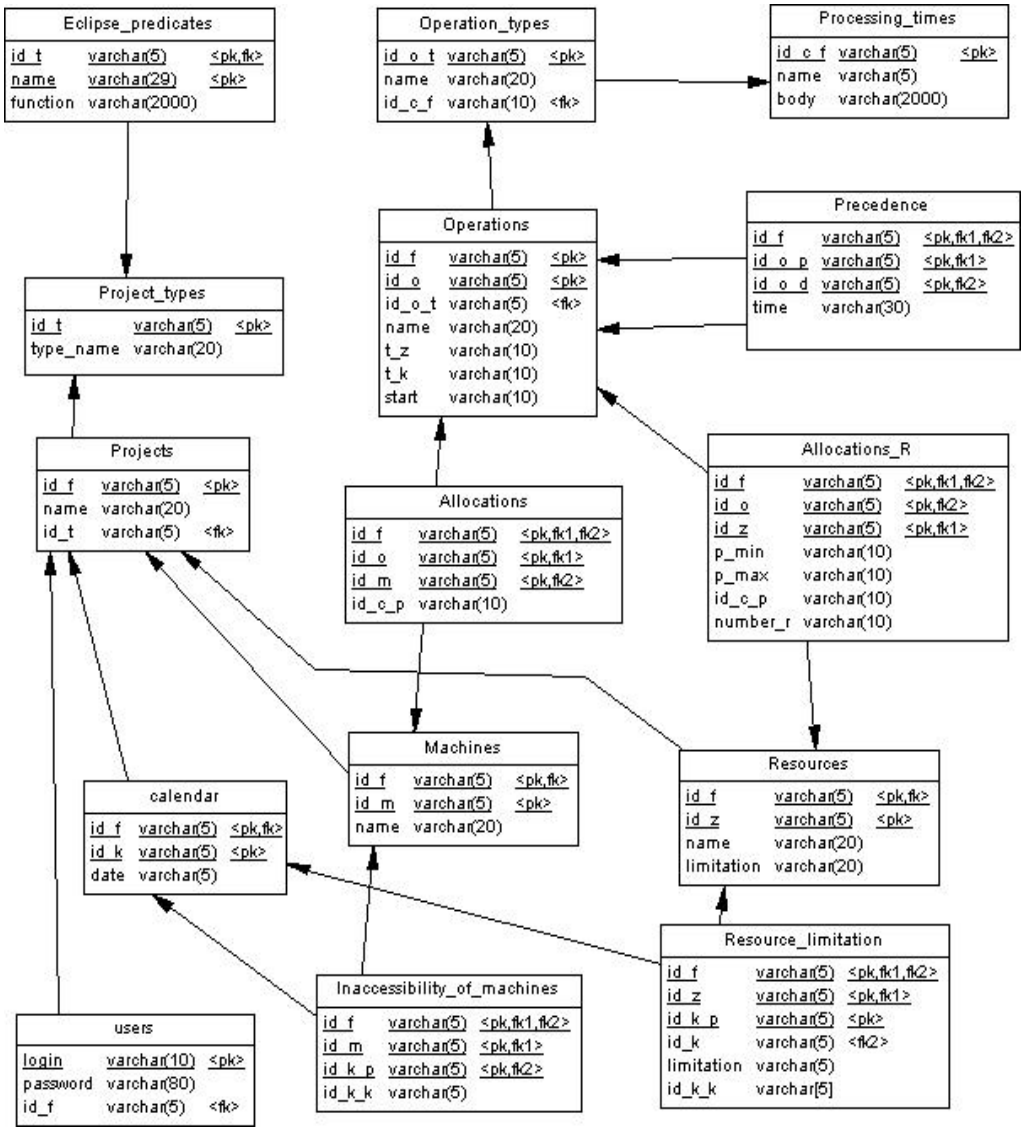

Fig.3 Schema of database of DSS for scheduling problems (Entity Relationship Diagram).

Table 1 Description of the database of DSS

Table_name	Table_description	Column	Column_desc
Project_types	the types of possible projects for realization	**id_t**	project_type_id
		type_name	project_type_name
Projects	the specification of separate projects in enterprises	**id_f**	project_id
		name	project_name
		id_t	project_type_id
Processing_times	the list of functions of time calculation	**id_c_f**	function_id
		name	function_name
		body	function_body
Opertaion_types	the list of operation types	**id_o_t**	operation_type_id
		name	operation_type_name
		id_c_f	function_id
Operations	the list of operations to be realized	**id_f**	project_id
		id_o	operation_id
		id_o_t	operation_type_id
		name	operation_name
		t_z	release time
		t_k	critical time
		start	start time
Precedence	defines the sequence of the realized operations	**id_f**	project_id
		id_o_p	operation_id
		id_o_d	operation_id
		time	time between operations
Machines	the specification of available machines for the operation realization	**id_f**	project_id
		id_m	machine_id
		name	machine_name
Allocations	the allocation of operation to machines	**id_f**	project_id
		id_o	operation_id
		id_m	machine_id
		id_c_p	parameters_of_function
Resources	the specification of renewable/external resources	**id_f**	project_id
		id_z	resource_id
		name	resource_name
		limitation	resource_limitation
Allocations_R	the allocation of renewable/external/additional resources to operations	**id_f**	project_id
		id_o	operation_id
		id_z	resource_id
		p_min	min number of allocated resource
		p_max	max number of allocated resource
		id_c_p	parameters_of_function
		number_r	the number of allocated resource
Calendar	the specification of planning/scheduling periods	**id_f**	project_id
		id_k	period_number
		date	starting_date
Inaccessibility_ of_machines	the specification of inaccessibility of machines	**id_f**	project_id
		id_m	machine_id
		id_k_p	number of initial period
		id_k_k	number of final period

Table_name	Table_description	Column	Column_desc
Inaccessibility_ of_resources	the specification of limitation/inaccessibility of machines	**id_f**	project_id
		id_z	resource_id
		id_k_p	number of initial period
		id_k_k	number of final period
		accessibility	number of accessible resources
Type_of_lines		**id_l**	line generation type
		type	type description
		PHP_function	function (in script language)
Gener	describes the process of model generation for Eclipse	**id_f**	project_id
		step	Number of generation step
		id_l	line generation type
		line	line to be made
Eclipse_predicates	the codes for the ready predicates of Eclipse	**id_f**	project_id
		name	name of predicate
		body	code of predicate

5 ILLUSTRATIVE EXAMPLES

After the complete implementation of the DSS into ECL^{i}PS^e and SQL environments, computation experiments were carried out. The job-shop scheduling problem with manpower resources (Example 1) and project (Example 2) were considered.
The proposed illustrative examples cover a wide range of scheduling problems encountered in the SMEs. The examples are selected in such a way that they how two extremely different forms of production organization; repetitive production in the job-shop environment and the unique production including the project. The presented methodology makes solving scheduling problems possible also in indirect methods of production organization. Moreover, the examples are larded with problems of constrained resources (e.g. manpower, specialized machines, etc.) and the dependence of particular jobs processing time on the amount of the allocated resources, for instance.

5.1 Example 1 - The job shop scheduling with manpower resources

In the classical scheduling theory job processing times are constant (Example_1a). However, there are many situations where processing time of a job depends on the starting time of the job in queue or the amount of allocated additional resources (e.g. people) (Example_1b) etc. The parameters of computational examples are presented in table 1. There are 5 jobs, each consist of 6 operations. The job data structures are shown in Fig. 4.

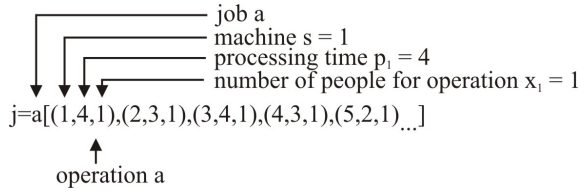

Fig.4 Description of task (job) data structure for job-shop computational example (Example_1a)

Table 2. Parameters of computational examples (Example_1

j∈{A,B,C,D,E}, o∈{1,2,3,4,5,6}, s∈{1,2,3,4,5,6}
j=A[(1,2,4), (2,4,2), (3,4,1), (4,2,1), (5,2,1), (6,3,4)]
j=B[(2,2,2), (3,3,3), (4,4,4), (1,2,3), (5,6,1), (6,3,2)]
j=C[(5,2,3), (4,2,1), (3,3,4), (2,4,3), (6,2,4), (1,4,4)]
j=D[(2,4,3), (3,2,6), (4,3,2), (5,2,3), (6,4,2), (1,4,4)]
j=E[(1,2,3), (3,4,6), (5,4,2), (6,4,2), (4,3,2), (2,2,2)]

The resource occupancy can be interpreted as a job with the fixed start times for all their operations and fixed manpower requirements. For the computational example the following questions (write following predicates) were asked (see section 3):

- *opc_g(_,_)* (see fig. 5, 6).
- *opc_g(8,_)* (see fig. 7,8).
- *opc_d(_,35)* (see fig. 9)
- *opc_s(10,30)* (see fig. 10).
- *opc_s(10,28)* (see fig. 11).

Computation experiments were started on the computer PIV 1,4 GHz, RAM 512 under Windows XP.

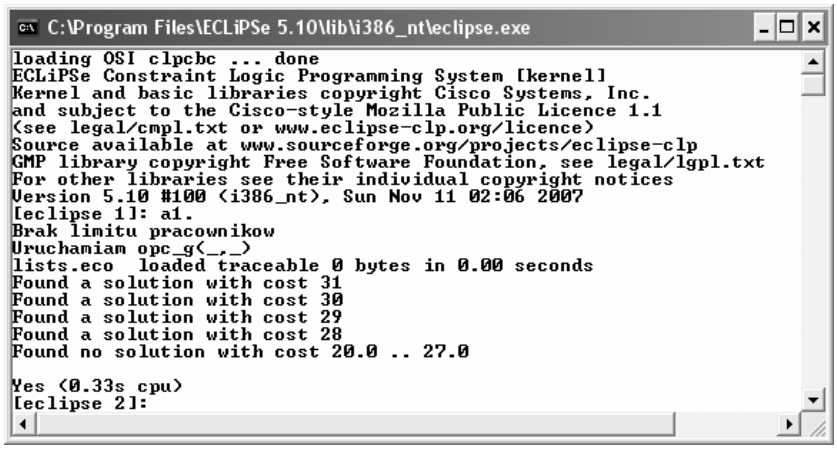

Fig. 5 Answer to the question implemented in predicate *opc_g(_,_)*–result C^*_{max}=28, L=14 (Example_1)

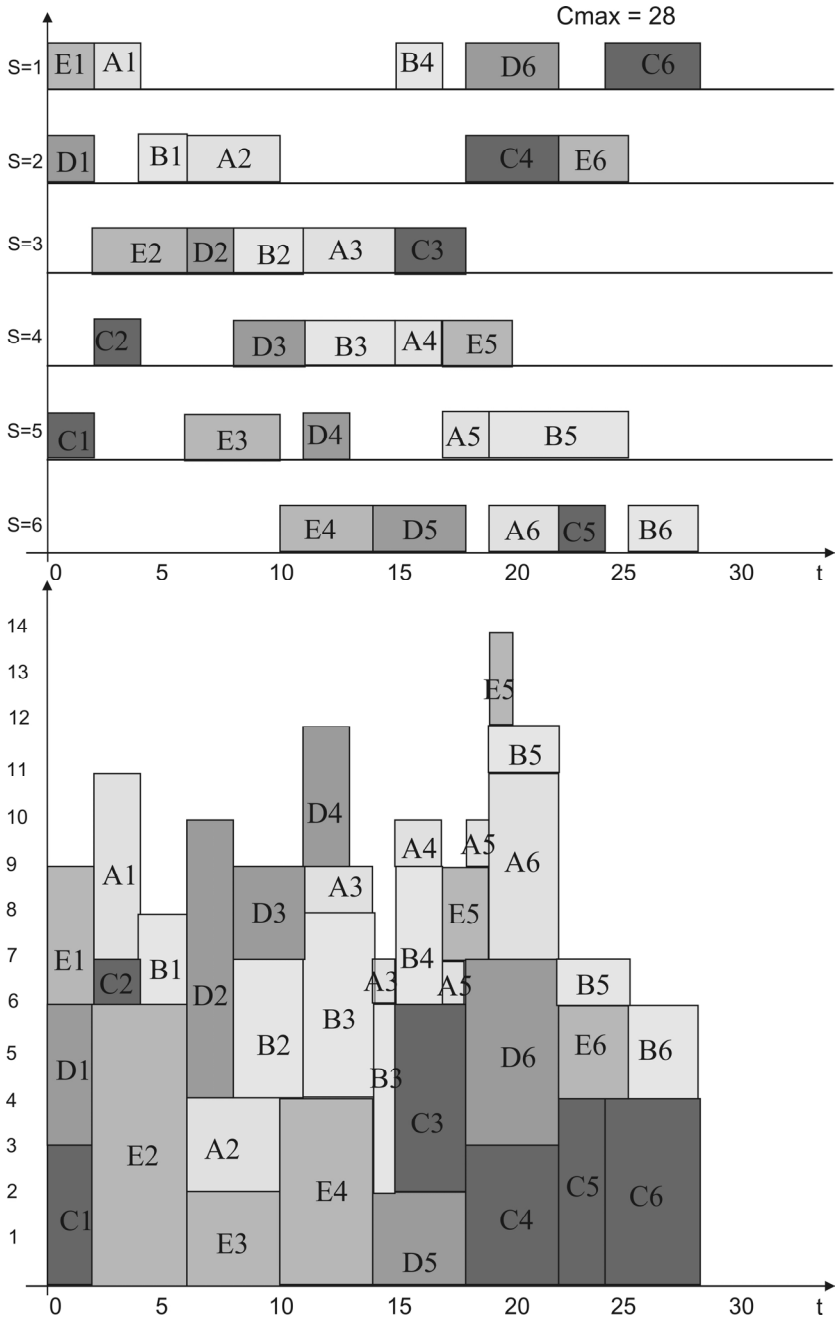

Fig. 6 Gantt's chart for decision from fig.5 (Example_1)

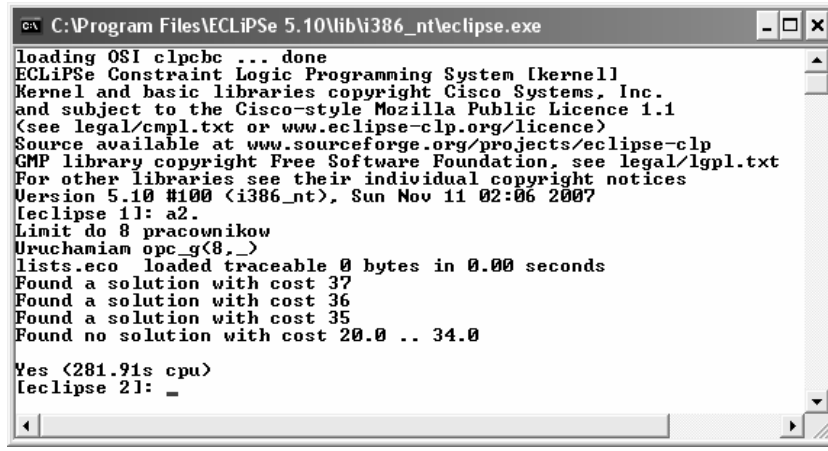

Fig. 7 Answer to the question implemented in predicate *opc_g(8,_)*–result C^*_{max}=35,L=8 (Example_1)

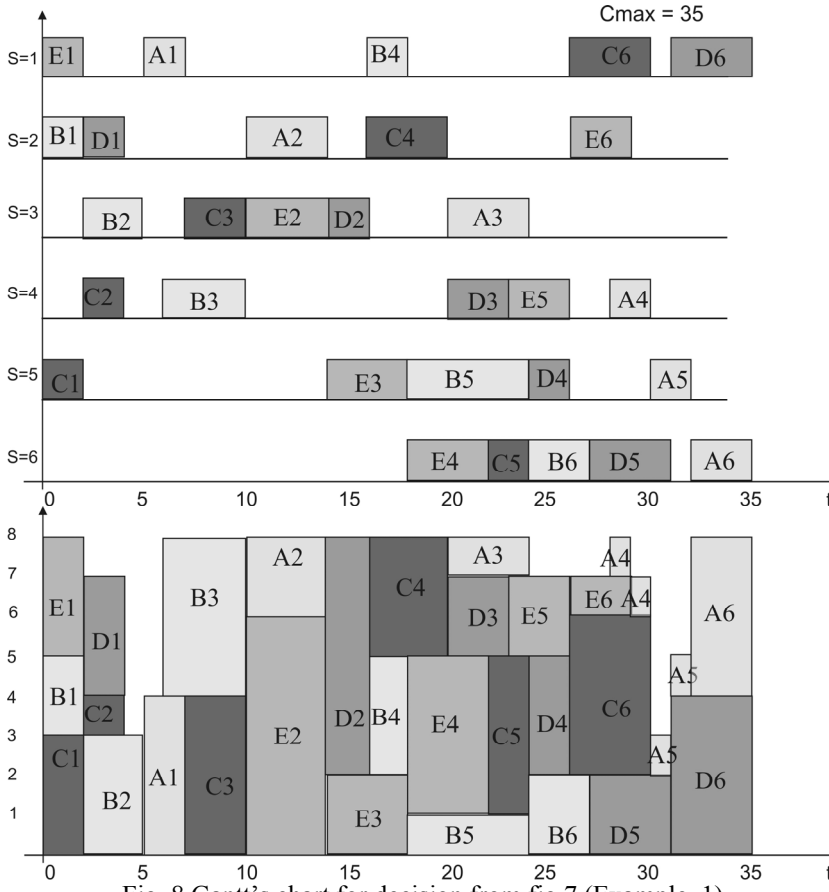

Fig. 8 Gantt's chart for decision from fig.7 (Example_1)

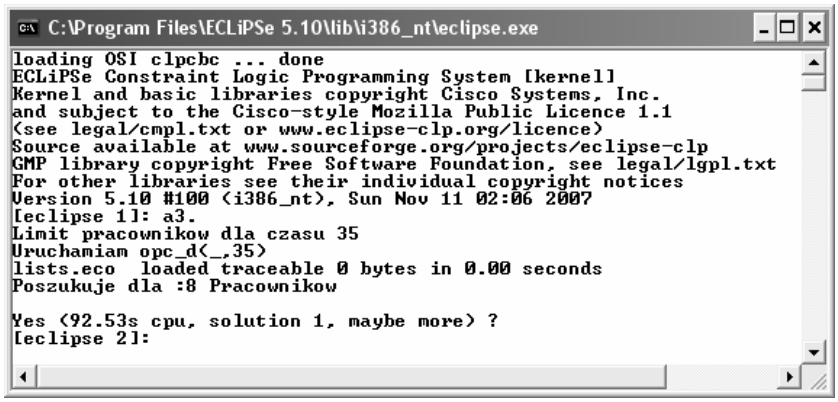

Fig. 9 Answer to the question implemented in predicate *opc_d(_,35)*–result, L_{min}=8 (Example_1)

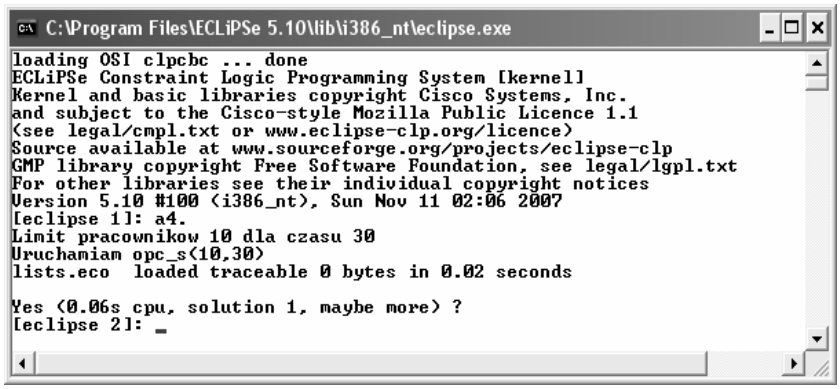

Fig. 10 Answer to the question implemented in predicate *opc_s(10,30)*–result *Yes* (Example_1)

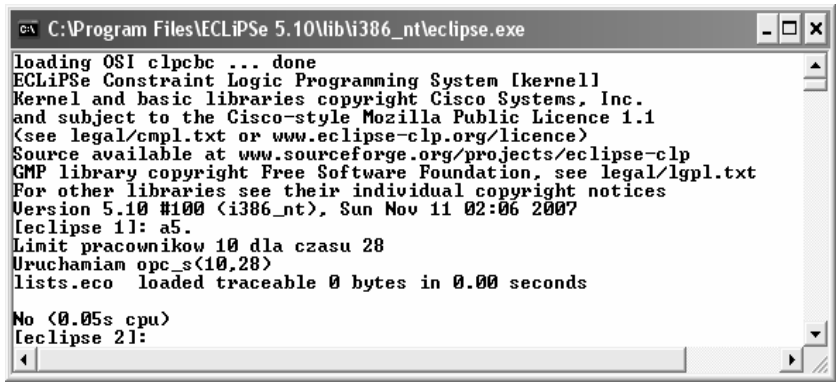

Fig. 11 Answer to the question implemented in predicate *opc_s(10,28)*–result *No* (Example_1)

5.2 Example 2 –-project

A typical modern-day project has a variety of complications not considered in the original PERT/CPM methodology. There are three particular situations:

- You may be able to accelerate the completion of a project by speeding up or "crashing" some of the activities in the project.
- Your ability to finish a project quickly is hindered by limited resources (e.g., two activities that might otherwise be done simultaneously, in fact have to be done sequentially because they both require a crane and you have only one crane on site).
- How long it takes to do each activity is a random variable.

In table 3, we list the activities involved in a simple, but nontrivial, project (building a house, building a bridge, etc.) An activity/operation cannot be started until all of its predecessors are finished. The network activity for this project has been shown in fig.12. To solve this example the DSS with declarative programming (section 4) was used. In this example the processing times of activities are constant (Example_2a, Table 3) or depend on allocated manpower resource (Example_2b, Table 5). The numeric results of these experiments have been shown in table 4 (Example_2a) and table 6 (Example_2b).

Table 3 Parameters of Example_2a

activity/operation	processing time	required predecessor	manpower
A	2	-	4
B	3	A	8
C	4	B	4
D	3	B	3
E	2	D	3
F	3	C, E	6
G	4	F	8
H	5	G	4
I	3	G	6
J	2	H, I	8

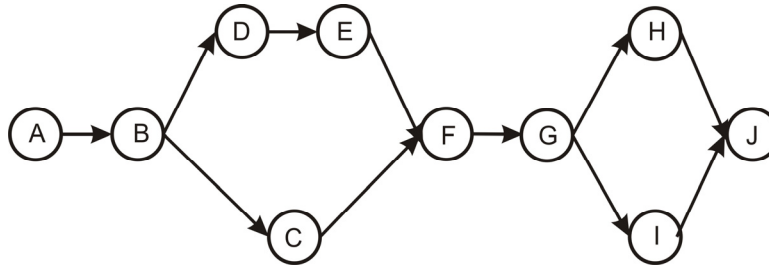

Fig. 12 Activity network (Example_2a, Example_2b)

For the computational example (Example_2a) the following questions (write following predicates) were asked (see section 3):

- *opc_g(_,_)* (see fig.13)
- *opc_d(_,24)* (see fig. 14)
- *opc_g(8,_)* (see fig.15)

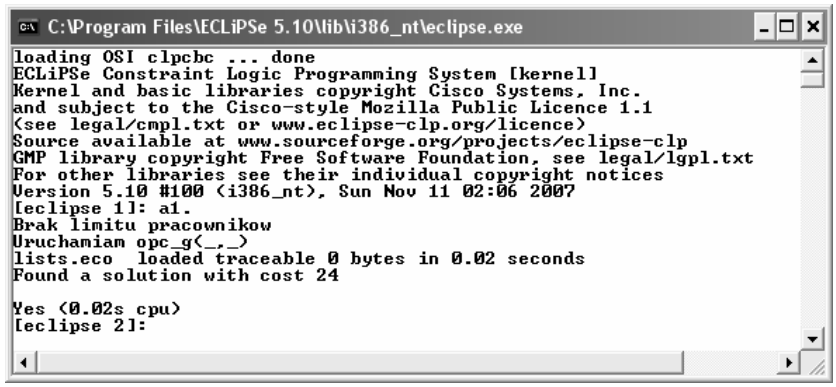

Fig. 13 Answer to the question implemented in predicate *opc_g(_,_)*–result *the shortest time to complete projec=24* (Example_2a)

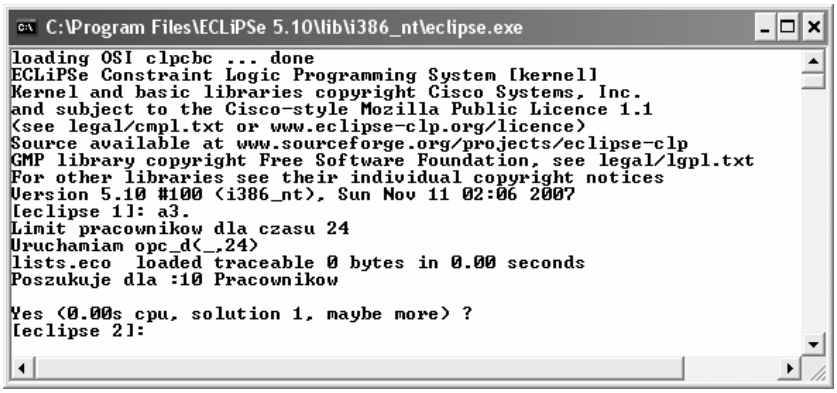

Fig. 14 Answer to the question implemented in predicate *opc_d(_,24)*–result, $L_{min}=10$ (Example_2a)

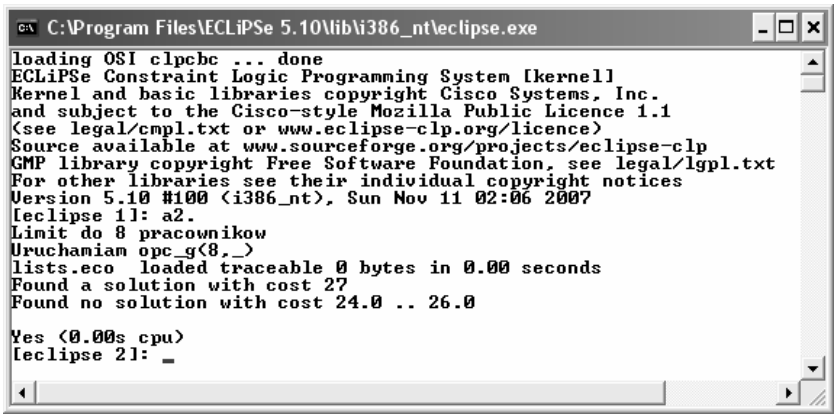

Fig. 15 Answer to the question implemented in predicate *opc_g(8,_)*–result *the shortest time to complete project*=27 (Example_2a)

Table 4 Results of Example_2a

activity/operation	Answer to *opc_g(_,_)* *Start time*	Answer to *opc_d(_,24)* *Start time*	Answer to *opc_g(8,_)* *Start time*
A	0	0	0
B	2	2	2
C	5	5	5
D	5	5	5
E	8	8	8
F	10	10	10
G	13	13	13
H	17	17	20
I	17	17	17
J	22	22	25

Table 5 Parameters of Example_2b

activity/operation	required predecessor	manpower	processing time-PT	Additional manpower/shortening PT			
				0	**1**	**2**	**3**
A	-	4	2	0	1	1	1
B	A	8	3	0	0	1	2
C	B	4	4	0	1	2	3
D	B	3	3	0	1	1	2
E	D	3	2	0	1	1	1
F	C, E	6	3	0	1	2	2
G	F	8	4	0	1	1	2
H	G	4	5	0	1	2	3
I	G	6	3	0	1	1	2
J	H, I	8	2	0	1	1	1

For the computational example (Example_2b) the following questions (write following predicates) were asked (see section 4):

- *opc_g(_,_)* (see fig.16)
- *opc_g(10,_)* (see fig.17)

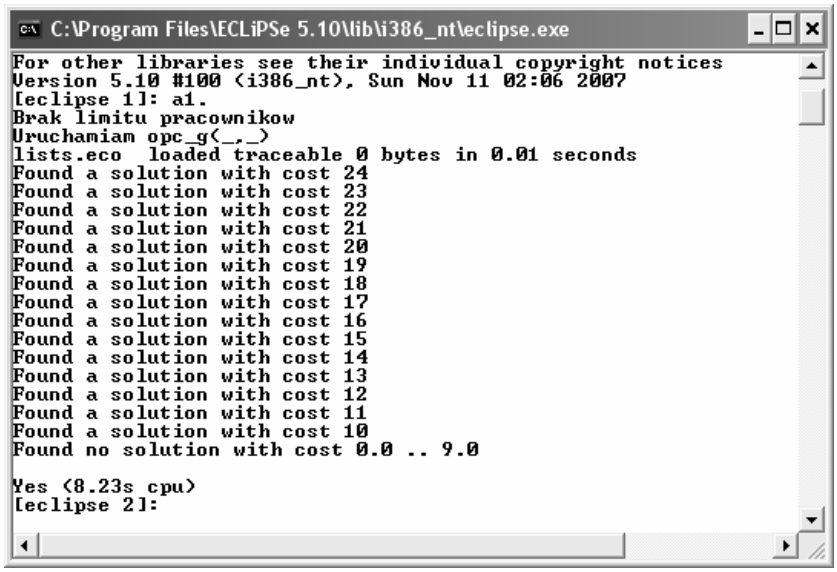

Fig. 16 Answer to the question implemented in predicate *opc_g(_,_)*–result *the shortest time to complete project* =10 (Example_2b)

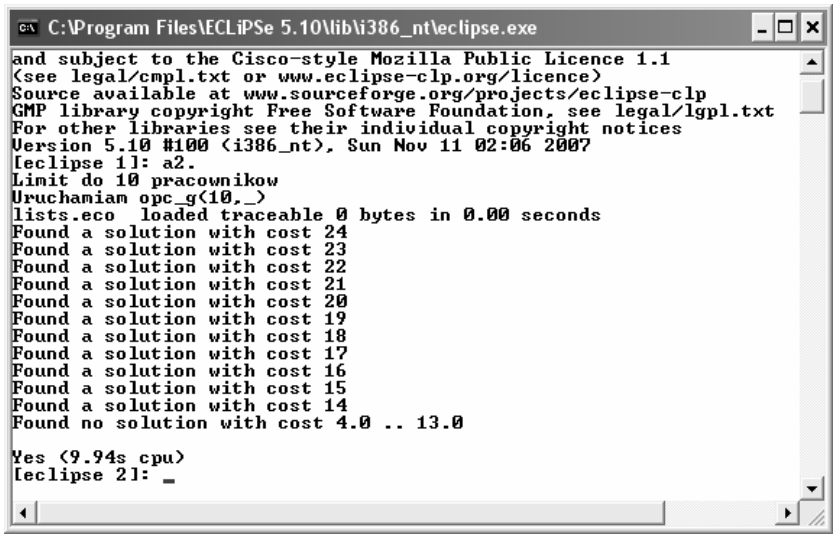

Fig. 17 Answer to the question implemented in predicate *opc_g(10,_)*–result *the shortest time to complete project*=14 (Example_2b)

Table 6 Results of Example_2b

activity/operation	Answer to *opc_g(_,_)*			Answer to *opc_g(10,_)*		
	processing time	manpower	start time	processing time	manpower	start time
A	1	5	0	1	5	0
B	1	11	1	2	10	1
C	2	6	2	3	5	3
D	1	6	2	2	4	3
E	1	4	3	1	4	5
F	1	8	4	1	8	6
G	2	11	5	3	9	7
H	2	7	7	2	7	11
I	2	7	7	1	9	10
J	1	9	9	1	9	13

6. CONCLUSIONS

The proposed approach can be considered to be a contribution to scheduling and especially to scheduling problems with additional/external resources. In many enterprises this kind of resources can have influence on production and delivery schedules. That is especially important in the context of cheap, fast and user friendly decision support in SMEs (Small and Medium Sized Enterprises). Great flexibility of the presented approach and practically unlimited possibilities of asking questions through creating predicates cannot be overestimated. What is more, the whole decision system can be built in one modeling and programming declarative environment, which lowers costs and adds to the solution effectiveness. The CLP-tools fulfill the need of intelligent production management structures and can be based successfully in cases of scheduling problems with external resources. The proposed approach seems to be a viable alternative option for supporting quite a number of decision making processes. The originality of our approach, which achieves the transition from custom imperative programming to declarative programming in a field of scheduling problems, consists in the data structure and CLP implementation. The presented framework can be implemented in many other constrained search problems apart scheduling like planning, routing, placement etc.

REFERENCES

1. LIAO S.Y., WANG H.Q., LIAO L.J.: *An extended formalism to constraint logic programming for decision analysis*, Knowledge-based Systems 15, 2002 , pp 189-202.
2. PEABODY G.: *Interpath connects Customer to SAP Applications via World-class Communications*, Data Center and Support Infrastructure Aberdeen Group 2000.
3. LACITY M.C., HIRSCHHEIM L, WILLCOCKS: Realizing outsourcing expectations, Information Systems Management 11(4), 1994, pp 7-18.

4. BISDORFF R., LAURENT S. *"Industrial linear optimization problem solved by constraint logic programming",* European Journal of Operational Research 84 (1), 1995, pp 82-95.
5. LAMMA E., MELLO P., MILANO M. "*A distributed constrained-based scheduler"*, Artificial Intelligence in Engineering 11,1997, pp 91-105.
6. LEE H.G., LEE G. Yu., "Constraint logic programming for qualitative and quantitative constraint satisfaction problems", Decision Support Systems 16 (1), 1996, pp 67-83.
7. RYU U. Young ."*Constraint logic programming framework for integrated decision supports"* Decision Support Systems 22, 1998, pp 155-170.
8. BENNETT Ch. TIMBRELL G.: *Application Services Providers: Will They Succeed* ?, Information Systems Frontiers, pp 195 – 211, Kluwer Academic Publishers, 2000.
9. http://www.cs.kuleuven.ac.be/