A MODEL OF THE DISTRIBUTED CONSTRAINT SATISFACTION PROBLEM AND AN ALGORITHM FOR CONFIGURATION DISIGN

Leonid B. Sheremetov

Centro de Investigacion en Computacion,
Instituto Politecnico Nacional,
Unidad Profesional Adolfo López Mateos,
México, D.F., C.P. 07738
e-mail: sher@pollux.cenac.ipn.mx

Alexander V. Smirnov

St.-Petersburg Institute for Informatics and Automation of the Russian Academy of Sciences (SPIIRAS)
SPIIRAS, 39, 14th line, St.-Petersburg, 199178, Russia
e-mail: smir@mail.iias.spb.su

ABSTRACT

The objective of this research is to develop an approach to representing and satisfying constraints during the cooperative configuration design of complex objects with hierarchical structure. Designers often collaborate in product development, so our constraint satisfaction mechanism is based upon multi-agent technology, permitting communication among participating applications coupled with localized solution methods. A system to be configured is divided into fragments; each of them based on local knowledge about template component compatibility. In this case each fragment development task is treated as an agent with embedded constraint-satisfaction facilities. An agent is considered as a computational process with expertise about a limited portion of a design problem, capable of achieving specific goals and communicating with other agents. Configuration design process is formulated as a distributed dynamic constraint-satisfaction problem, accomplished by a number of agents in parallel. The algorithm of distributed search on the dynamic constraint network is discussed. The developed algorithm is well suited to collaborative design because it operates incrementally and without global information about the constraint network. The architecture of Concurrent Cônfiguration Design Advisor - a distributed agent-based expert system for configuration design of complex objects with hierarchical structure - is presented. The discussion is illustrated with the examples from FMS configuration design application domain. Current and future work on the expert system implementation is considered.

Key words: agents, distributed dynamic constraint satisfaction, configuration design, manufacturing system.

INTRODUCTION

Design of large-scale systems involves consideration of hundreds or thousands of often competing concerns such as manufacturability, testibility, cost, etc. The combinatorial complexity of this problem is enormous and, as such, certain models of problem domain knowledge representation and heuristic rules are to be employed to reach an acceptable configuration within a reasonable time frame. It implies a need for new algorithms and program structures able to perform simultaneously configuring procedures. As so, distributed knowledge processing and concurrency become a fundamental requirement for CAD, providing cooperation between engineering systems [3, 7, 8].

This paper discusses the configuration design problem. Configuring is the construction of a technical system according to the requirements of a specification by selecting, parameterizing and positioning instances of suitable component types from a given catalogue [11]. A formulation and representation of configuration design as a distributed dynamic constraint-satisfaction problem (DCSP) and its implementation in the Concurrent Configuration.

Design Advisor (CCDA) - distributed multi-agent expert system (ES) - are considered. In the agent oriented approach an ES is represented as a collection of loosely-coupled autonomous agents that organize synchronous and asynchronous communications among themselves by passing messages based on project model specifications [1, 13, 19, 20]. In this paper an agent is considered as a computational process with expertise about a limited portion of a design problem, capable of achieving specific goals, and communicating with other agents. These agents use a set of operations and heuristics to navigate through the space of possible designs. CCDA agents are distributed functionally and geographically.

In this paper we describe the model and the algorithm of distributed search, discuss agent-based architecture of the CCDA, review our initial experiments in distributed configuration design and outline future directions. Illustrations are made from the Flexible Manufacturing System (FMS) configuration design domain.

PROBLEM FORMULATION BASED ON A DCSP MODEL

Constraint satisfaction problems (CSPs) consist of: a set of variables, the domains for the variables, and the constraints on the variables [18]. A static constraint network (SCN) (V,dom,C) involves a set of variables $=\{i,j,\ldots,\}$, each taking value* in its respective domain, $dom(i),dom(j),\ldots$, and a set of constraints *C*. Each constraint $c(i_1,\ldots,i_q)$, constraining the subset $(i_1,\ldots,i_q)$ of *V*, is a set of tuples, subset of the Cartesian product $dom(i_1)\times\ldots\times dom(i_q)$, that specifies which values of the variables are compatible with each other. A dynamic constraint network (DCN) *S* is a sequence of static CNs $S_{(0)},\ldots,_{(\alpha)},S_{(\alpha)},S_{(\alpha+1)},\ldots$, each resulting from a change in the preceding one imposed by "the outside world". The solution for the CSP is a value assignment for each variable such that all the constraints are satisfied.

The problem of consistency-based configuration design was formulated in [15]. It maps directly into CSPs. The variables and equations of the model become the variables and constraints of the CSP, with the addition of variables representing the state of the various components, and the variables and values representing the observations. The search problem then becomes one of finding an assignment of normal/abnormal to the attributes of the components that is consistent with the observations.

Let us formally define the configuration problem as CSP, given: A set of possible designs:

$$D=\{d_1,d_2,\ldots,d_n\};$$

A set of preferably independent attributes that describe a design:

$$A=\{a_1,a_2,\ldots,a_n\}.$$

Then a complete design is an n-tuple:

$$d_i=(d_i.a_1,d_i.a_2,\ldots,d_i.a_n),$$

where $d_i.a_k$ is a value of an attribute for this particular design. A set of complete designs semantically forms possible worlds, where design constraints are believed by an agent to be satisfied. So a set of possible designs can be represented in the form of attribute intervals for each tuple element in the following way:

$$[a_k]=[\min(d_i.a_k),\max(d_i.a_k)].$$

A constraint is a relation over a subset of the attribute space that defines feasible designs. A constraint is satisfied when no design lies outside it's feasible region. Examining the bounds of the attribute space we can easily detect which designs lie outside the region and are to be removed. For each constraint c_j defined over a subset A_j of design attributes: $A_j\subseteq A$, a constraint-based decomposition can be defined as the subset of the attribute space specifying the feasible region for c_j. An unary constraint applies only to one node. A binary constraint involves two nodes. This notion is extensible to an n-ary constraint over *n* nodes. Multiple unary constraints on a node can be equivalently represented by a single unary constraint that is their conjunction. The same technique can be applied to represent multiple binary constraints between two nodes as a single binary constraint between them. In the case, when a constraint precondition is not always true (it can be a function of other component attributes), we have DCN.

Complex design *D* is decomposed into fragments. A fragment is a component type specification. It can specify an atomic class (low-level objects, such as machine tools, automated guided vehicles -AGV, robots) or can be decomposed further, specifying a high-level class such as FMS on the shop floor level, FMS cell, etc. Given that, the model of a complex design *D* can be considered as a set of DCN, each of them defining a model of a fragment but also considered as a node of DCN of the upper level (fig1).

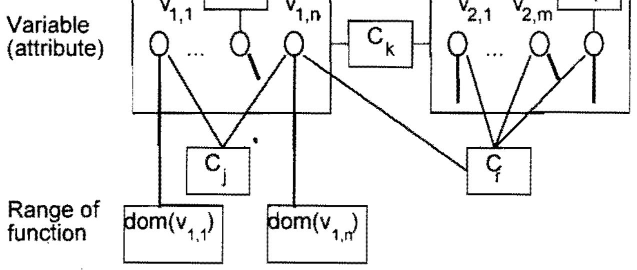


Figure 1. Multilevel constraint network: c_n, c_p - unary, c_j, c_k - binary constraints.

Let F be a set of fragments:

$$F = \{f_1, f_2, \ldots, f_m\}.$$

The multi-valued attribute space domain of f_j is defined as follows:

$$[f_i] = ([f_i . a_1], [f_i . a_2], \ldots, [f_i . a_m]).$$

Then a solution of DCSP is an assignment $f_i = \{f_i . \alpha_1, f_i . \alpha_2, \ldots, f_i . \alpha_n\}$, $\alpha_k \in [a_{jk}]$, $\forall [\alpha_{jk}] \in [A_j]$, $f_i \in F$, that satisfies the set of constraints C.

The task of finding a solution in a DCN is NP-complete, so a number of local consistency algorithms have been proposed [2, 5, 9]. The most widely used are those achieving arc-consistency, checking the consistency of values for each pair of variables linked by a constraint. We define some basic notions of consistency below [18].

Node-Consistency of a node: A node *i* is node-consistent if and only if all labels in its domain satisfy all unary constraints on that node.

Arc-Consistency of an arc: An arc *(i; j)* is arc-consistent if and only if nodes *i* and *j* are node-consistent and for any label in the domain of node *i*, there exists a label in the domain of node *j* such that all binary constraints on the two nodes are satisfied.

Consistency of a SCN: A SCN is said to be node- or arc-consistent if and only if every node or arc respectively is node- or arc-consistent.

Arc-consistency is very simple to implement and it has good efficiency; it is described in [5]. Solutions that lie outside the arc-consistent space are removed from consideration. It is equivalent to removing designs that violate the constraints, since removed fragments are those that appear in possibly infeasible design.

RECURSIVE SEARCH ALGORITHM FORMULATION

In CCDA configuring of a design is based on multi-level decomposition 'and concurrent fragment design. Each fragment is represented as a DCN, an agent is associated with it to perform a configuring task. We shall call this agent a 'design agent' or D-agent. Each D-agent is oriented to solve a configuration problem on its level, organizing the template solution search. This process is applied recursively to produce a set of sub-fragments that cannot be subdivided further. The procedure of solution generation is depicted in Figure 2. The algorithm explores a design tree looking for nodes that correspond to "solutions". Each time, when called to expand a node, this procedure checks the component data bases to see whether the node in question has a solution. If not, the algorithm searches the template knowledge base (KB) for the design *D*. If no template is found, then it generates an agent of another type - a 'project assistant' or A-agent - to generate a new template. After that it makes recursive calls to the same procedure to expand each of the offspring fragments.

```
Procedure search(P)
begin
    if (solution(P)) then
        score = eval(P)
            report solution and score
    else
        if (search_template(P)) = FALSE then
            generate_template(P)
        endif
        foreach concept $P_i$ of P
            search ($P_i$)
        endfor
    endif
end
```

Figure 2. Recursive formulation of a configuration solution search algorithm.

A parallel algorithm for this problem can be structured as follows. Initially, a single agent, responsible for the entire project configuring, is created for the root of the tree. An agent evaluates its node and then, if that node is not a solution, creates a new agent for each fragment (sub-tree). A channel created for each new task is used to return to the new task's parent all the solutions located in its sub-tree. Hence, new agents and channels are created in a wavefront as the search progresses down the search tree.

Figure 3 depicts the process of top-down configuration design, based on DCN. All the relations between components are described in terms of constraints. Dashed ellipses represent project fragments, for which a solution can't be found in the component data base, and then the process of logical fragment decomposition begins (fragment f_n, sub-levels n_1 and n_2). Each D-agent, associated with f_n at the *n* level (server), uses different heuristics to create *M* fragments of the configuring task by activating *M* agents (clients) at level *n+1* to configure the fragment. One of the main heuristics is based on minimum fragmented structure. Each fragment has maximum close-coupled components, which structure is generated by a constraint network. DCN on each level is created by sub-fragment constraints, generated by means of constraint propagation and the rules of compatibility of components, because concurrently created solutions can't be independent. Each rule of compatibility is represented by first order predicate language and has set of attributes, which are considered by the algorithm of fragment definition. A D-agent utilizes inferencing techniques to reason about feasible D-agent operations over a project and so can be conceptually viewed as a virtual project server. To have an opportunity to choose the best solution from the possible solution set, a set of clients for the same task can be generated, as shown in Figure 3 for the level *n+1*.

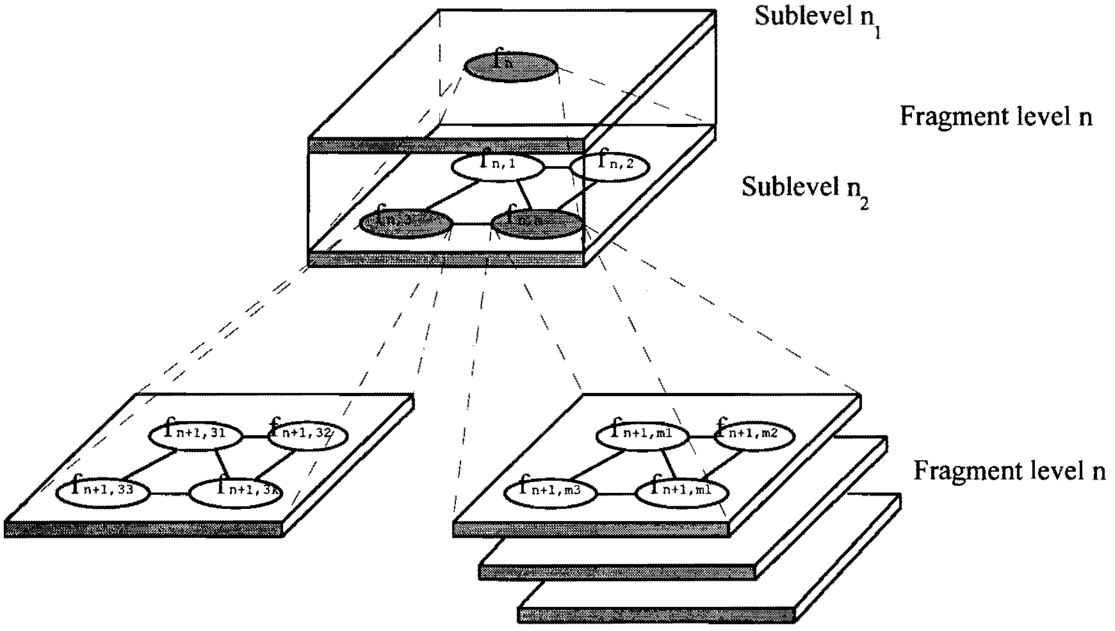


Figure 3. Hierarchical dynamic constraint network

A template is a model of a fragment. It can be of an elementary level or can contain other concepts with interrelations between them. It also can be considered as a structured model of a problem domain. Being object-oriented, it contains also a set of defined operations on the corresponding data type. A component is an instance of a template. It has valued attributes and can be considered as a solution. Template generation can be viewed as a knowledge acquisition procedure, performed by domain experts and supported by an A-agent. A project server (an arbitrary decomposable D-agent) receives an expert assignation query from one of its clients (from *n+1* level) and organizes the remote access to the A-agent in the distributed environment. ES contains the data base with the expert's experience, professional level and problem domain. The expert assignation task is solved by a D-agent (server), while generating an A-agent.

The developed algorithm is well-suited to collaborative design because it operates incrementally and without global information about the constraint network. The solution process starts with known values or fully constrained fragments. These fragments, represented by nodes in the constraint network that have no remaining degrees of freedom, notify any adjacent constraints to consider them as inputs. The constraints then determine if sufficient inputs are available to propagate information to additional fragments. This process continues in a depth-first manner. If the model is well constrained and no cycles exist in the constraint network, local propagation will generate a complete solution. If the system is over-constrained, there will be redundant or conflicting constraints. In the latter case, the conflicts must be resolved by retracting or temporarily relaxing constraints. Under-constrained problems will be characterized by propagation being completed before all parts of the network have been visited.

D-AGENT FUNCTIONAL STRUCTURE AND INTERNAL KNOWLEDGE REPRESENTATION MODEL

D-agent is a self-contained process, consisting of a single agent program. It's functional specification involves fragment decomposition, constraint satisfaction procedures, choice and ordering heuristics for constraint propagation.

The network of constraints in a product model can be viewed as a bipartite graph in which each component (part, equipment, geometric element, etc.) contains links to the constraints in which it is involved. The constraints, in turn, contain links to the components they relate. A product model may include several types of constraints. Geometric constraints, for example, allow the designer to specify how the geometry should be built declaratively, rather than constructing it interactively as is done in many CAD systems. Similarly, algebraic constraints maintain relationships between product variables. Constraints may also involve features, which we define as regions of interest within a particular context. Fragments such as FMS on the shop floor level, FMS cell, etc. are supplied with procedures for satisfying constraints on the lower-level fragments they contain. Local constraint networks within fragments define relationships among their subsidiary components.

A component catalog is usually associated with a fragment containing all its possible instances. The catalogue knowledge is the main knowledge base. It contains knowledge about fragments, the types of components available for configurations, and component attribute specifications. An example of requirements specification for the 'payload capacity' unary constrained attribute is defined by the following structure:

Name (a pointer to the requirement):	*Machine_payload_ capacity*
Attribute (project feature):	*payload capacity*
An object type:	*Machine*
Operator (comparison):	*= (is equal to)*
Goal (required interval value):	*[10.000 14.000]*
Precision (acceptable deviation):	*5 (%)*

The task in a CSP is to assign labels to nodes such that all the constraints on the nodes are satisfied simultaneously. The typical FMS design consists of a number of machine tools of different types, robots for parts supply, a clamping/resetting station, and some auxiliary equipment. Figure 4 shows an excerpt from the example DCN network for FMS cell configuration problem, defining a set of multiple unary and binary constraints.

The exception knowledge in the form of binary constraints is attributed to the component types or classes it applies to, with different possible levels of expressiveness. Knowledge about simple incompatibility between components may be attached to a component as (i) a list of incompatible components checked against when a new component is selected or (ii) predicate logic expressions about unacceptable partial configurations, which form a checking procedure.

D-agent supports indefinite template description, i.e. one in a conceptual way when only component types are given, without defining values of attributes. D-agent selects a definite pattern decision proceeded from the template context.

A-AGENT: TEMPLATE DESIGN ASSISTANT

Project assistant is an agent that supports expert knowledge acquisition for the template design. A D-agent tries to find a template in the KB, corresponding to its level of abstraction. If this decision does not exist, an A-agent is generated. It has to be associated with an expert to perform the functional fragment representation mapping into the agent's structural model, i.e. new template generation. To perform the expert assignation task, the expert's KB is revised and the appropriate expert is selected.

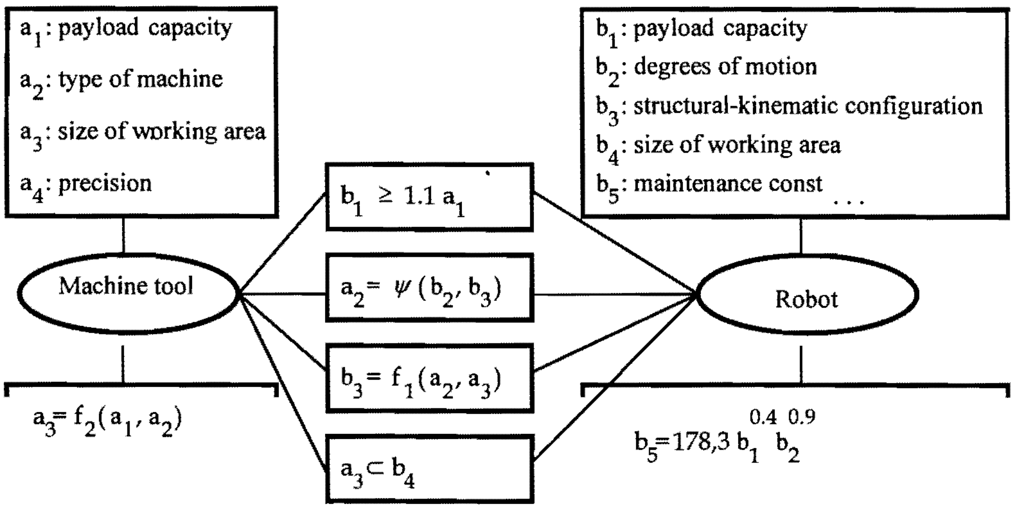
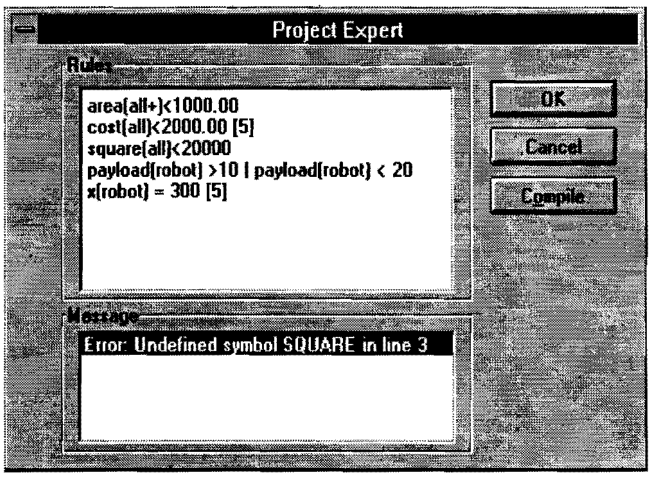


Figure 4. An excerpt from the constraint model of the FMS problem domain

An agent appears at the expert's desktop, reflecting the real hardware system architecture. In order to supply an expert with the facilities of knowledge representation, master tools are implemented. A project expert tool is a compiler that processes the rules defining integrated constraints (Figure 5). All constraints in a column are processed as a conjunction. Disjunctive function is also supported for the constraints, written in a row (*payload*). Each constraint can be associated with the precision value (see values in square brackets).



```
area[all+]<1000.00
cost[all]<2000.00 [5]
square[all]<20000
payload[robot] >10 | payload[robot] < 20
x[robot] = 300 [5]
```

```
Error: Undefined symbol SQUARE in line 3
```

Figure 5. Project Expert window

The following types of integrated constraints are supported:

- global constraint, applied to all the components (see *cost* for example),
- global additive constraint, processed as an addition of all attribute values belonging to the selected type (*area*),
- group constraint, applied only to the components of the selected type (*payload*).

Figure 6 depicts the Master Function tool. As shown in this figure, the taxonomic structure of the component and resource type catalogues is exploited with the familiar GUI tools allowing the definition of rules of compatibility of components. Functional constraints are organized as a DLL library for functions, defined for object classes. The Master Function mode is used to define specific functional constraints for selected objects by associating object attributes with function arguments, as shown in Figure 6.

Inheritance is supported, i.e. a new template can be generated by inheriting the parent's properties, redefining them in a late-binding manner. Also a new template can be added to the catalogue without reference to parent template descriptions. Any description can be removed together with all the knowledge pertaining to it without consequence for the rest of the template knowledge base.

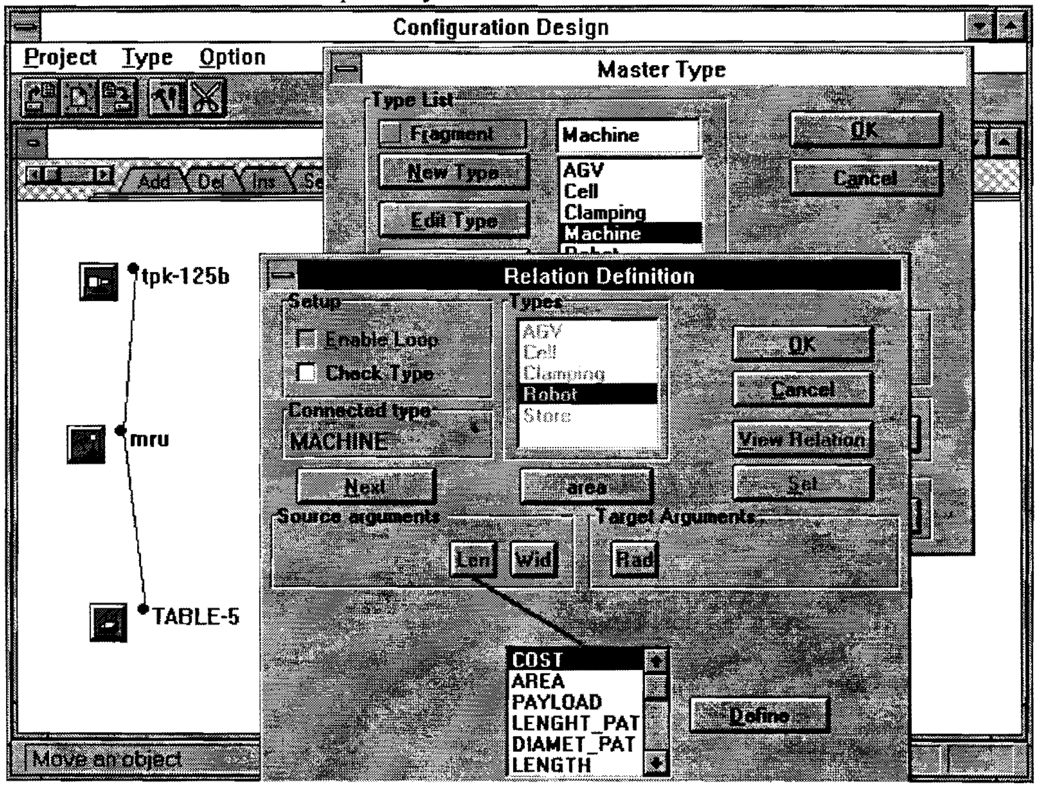


Figure 6. Expert Assistant interface

COMMUNICATION OF AGENTS

The CCDA architecture is depicted in Figure 7. It provides complex behavior by D-agent servers in response to a request from D-agent clients. D- and A-agents also generate SQL queries for the P-agent, responsible for component DB maintenance, in order to be supplied with the relevant information about components.

D-agent's coordination, synchronization and collaboration in conflicts of proposed solutions of a distributed design are supported by means of retracting and constraint relaxation. Each D-agent client tries to find template solutions based on its own criteria and send it to the D-agent server in the form of a proposal. The server evaluates these concurrent proposals. If these proposals satisfy integrated fragment constraints then it sends the confirmation message and writes these proposals in the design solution KB. If agreement can not be reached, the server sends messages to relax some constrains to the

D-agents, participated in each conflict. These steps are repeated until a satisfactory decision is achieved or it is recognized that conflict can not be solved due to the over-constraintness.

Communication among agents is performed through a message exchange infrastructure according to the pre-established set of protocols [12]. It is represented as a set of message schemata with the following structure. Vocabulary is a domain-specific part where each word has a formal annotation written in Knowledge Interchange Format (KIF) [6]. A KIF is a prefix version of the first order predicate calculus with various extensions to enhance its expressiveness. KIF defines a set of objects, functions and relations whose meaning is fixed, but the users are free to define the meaning of any other symbols that are not predefined. A message is a Knowledge Query and Manipulation Language (KQML) expression in which the arguments are terms or sentences of the vocabulary in KIF [10].

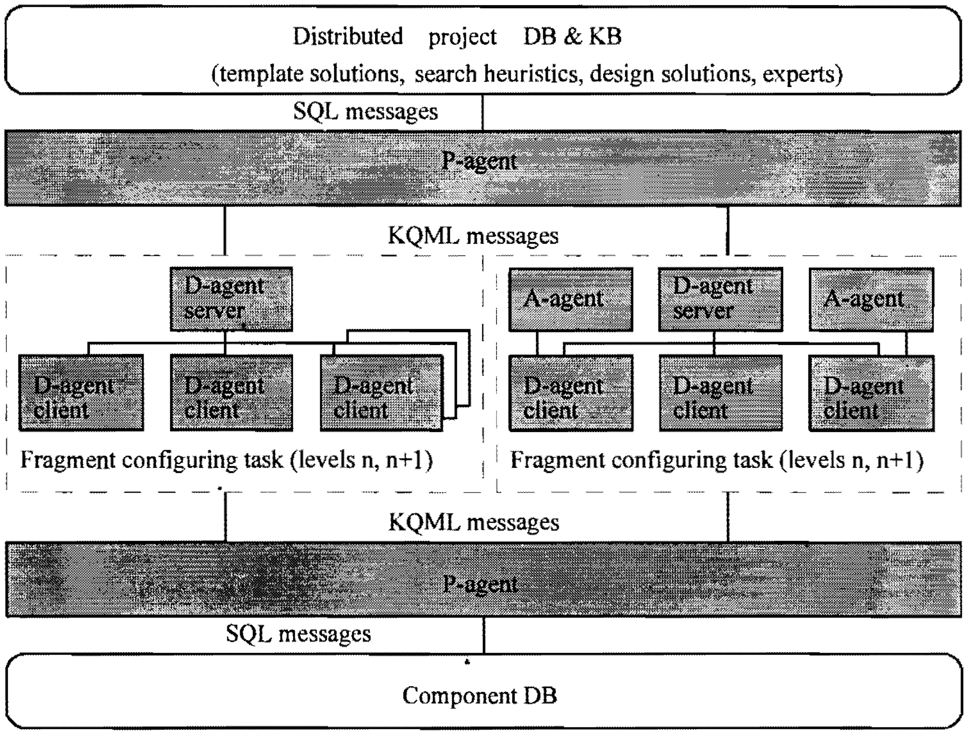


Figure 7. The CCDA system architecture

The D-agent actions are the KQML messages that include constraints, property of object, etc. and other user defined primitives. A message is a KQML expression in which the arguments are terms or KIF-sentences. KQML message is a list of components enclosed in matching parentheses. The syntax of the message is that of a performative followed by an unordered list of keyword-value pairs. A performative indicates the type of communication. There are two types of KQML messages: requests and announcements. For example, a message representing a query about the working area size of the particular machine tool might be encoded as:

(ask-one :receiver ***MachineDBManager***
:sender ***la_155f3***
:content ***working_area(Machine, Min, Max)***
:language ***prolog***
:reply-with ***tpk-125b)***

and may elicit the response:

(reply :receiver ***la_155f3***
:sender ***MachineDBManager***
:content ***working_area(Machine, 5.25, 9.86)***
:language ***prolog***
:in-reply-to ***tpk-125b)***

In this message, the KQML performative is ask-one, the content is ***working_area(Machine, X)*** and the assumed ontology is identified by the token ***MachineDBManager***.

The prototypes of agents are implemented using Borland C++ programming language with constraint satisfaction inference capabilities and runs on a PC. The use of a restricted version of the KQML library and the message representation API to support communications hides all details associated with network communication from the user.

Conclusion and Future Work

We have considered the DCSP model and the algorithm of distributed search for the configuration design problem. They are implemented in the multi-agent based computer environment - CCDA. Design decomposition into fragments takes advantage of distribution of design process among agents, each of them working with a single fragment, and supports cooperative centered mode of interaction among agents, which can reduce the time and increase the quality of configuration design process.

The first experiments were provided with the environment for the DEsign of Structured Objects (DESO). At that stage, constraint-based internal knowledge representation model and different communication models, based on reduced set of KQML and TCP/IP protocol stack, were investigated. The system was composed basically of A-agents, which had limited inferencing capabilities of D-agents (consistency control and indefinite template search procedures). On the basis of the experience obtained through DESO development, it can be posted that the universality of the described knowledge representation scheme for all kinds of structured systems makes it feasible to provide a powerful interactive tool for knowledge base maintenance. The constraint-based approach has given us also an opportunity to realize "test-generate" programming methodology to prune a problem search space. Agents communicating on a knowledge level can encapsulate their internal knowledge and then be invoked remotely as network services when needed. The results of these experiments were presented at a number of conferences [14-17].

The current prototype version attempts to use a DCOM object-based model of agent interactions [4]. A P-agent implementation and performance analysis with different data models is under investigation. In our future work more complicated agent negotiation algorithms are to be investigated, including those based on cognitive maps and negative-positive-neutral logic. Heterogeneous hardware and software environment is also to be explored.

ACKNOWLEDGMENTS

The Ministry of Science and Technology of Russia initially supported the work described under the grant N236/132. The experiments never would have succeeded without the contribution of the group of researchers and programmers of the laboratory of computer integrated systems of St. Petersburg Institute of Informatics and Automation of the Russian Academy of Science.

REFERENCES

[1] *Artificial Intelligence (special issue).* 94(1), 1997.

[2] R. Chopra, R. Srihari, and A. Ralston. Expensive Constraints and HyperArc-Consistency. In *Workshop on Constraint-Based Reasoning (CONSTRAINT-96),* Key West, Florida, 1996.

[3] H.R. Frost and M.R. Cutkosky. Design for Manufacturability via Agent Interaction. In *1996 ASME Design for Manufacturing Conf.*, pp. 18-22, Irvine, CA, Aug., 1996.

[4] Distributed Component Object Model protocol DCOM/1.0. Network Working Group, Internet-Draft, 1996.

[5] Y. Deville and P. Van Hentenryck. An efficient Arc Consistency Algorithm for a Class of CSP Problems. In *12th Int. Conf. On AI (IJCAI-91),* pp. 325-330, Sydney, Australia, Morgan Kaufmann Publishers, CA, 1991.

[6] M.R. Genesereth, et al., Knowledge Interchange Format, Version 3.0 Reference Manual, Technical Report Logic-92-1, Computer Science Department, Stanford University, 1992.

[7] A.P. Gupta, W.P. Birmingham, and D.P. Siewiorek. Automating the Design of Computer Systems. *IEEE Trans. On Computer-Aided Design of Integrated Circuits and Systems*, 12(4): 473-487, 1993.

[8] M. Klein. Supporting conflict resolution in cooperative design systems. *IEEE Transactions on System, Man, Cybernetics,* 21(5):1379-1390, 1991.

[9] A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8: 99-118, 1977.

[10] J. Mayfield, Y. Labrou, and T. Finin. Evaluation of KQML as an Agent Communication Language. Intelligent Agents. *Lecture Notes in Artificial Intelligence,* Volume 1037, Springer-Verlag, 1996.

[11] D. Serrano. Constraint-Based Concurrent Design. *Systems Automation: Research&Applications,* 3(1) :217-230, 1991.

[12] L.B. Sheremetov. Estructuras de comunicación para la resolución de problemas de manera distribuida en la ingeniería concurrente. *Temas en ciencia y tecnologia.* 1(1):3-29, 1997.

[13] M.P. Singh. Multiagent Systems, A Theoretical Framework for Intentions, Know-How, and Communications. *Lecture Notes in Computer Science, Volume 799, (subseries: Lecture Notes in Artificial Intelligence),* Springer-Verlag, 1994.

[14] A.V. Smirnov, L.B. Sheremetov, G.V. Romanov, and P.A. Turbin. Multi-Paradigm Approach to Cooperative Decision Making. In *Proc. of the II International Conference on Concurrent Engineering: Research and Applications.* pp. 215 - 222, Washington, DC, August 23-25, Concurrent Technology Corporation, 1995.

[15] A.V. Smirnov, L.B. Sheremetov, and P.A. Turbin. Constraint-Based Expert System for the Design of Structured Objects. In *Proc. of the International AMSE Conference on System Analysis, Control & Design, Methodologies & Examples SYS'95.* V.2, pp. 64-71, Brno, Czech Republic, July 3-5, 1995.

[16] A.V. Smirnov, A.S. Kulinitch, L.B. Sheremetov, G.V. Romanov, and P.A. Turbin. DESO: A Constraint-Based Environment Prototype for Cooperative Design of FMS. In *Proc. of the III IASTED International Conference.* pp. 384-387, Cancun, Mexico, June 14-16. IASTED/ACTA Press. Anaheim-Calgary-Zurich, 1995.

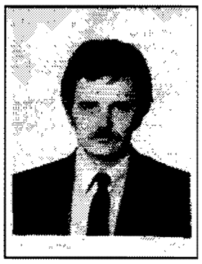
[17] A.V. Smirnov, L.B. Sheremetov, and P.A. Turbin. Information Support of FMS Configuration Design. In *2nd IEEE/ECLA/IFIP International Conference on Architectures and Design Methods for Balanced Automation Systems - BASYS'96.* Costa da Caparica, Portugal, 17-19 June, 1996.

[18] E. Tsang. Foundations of Constraint Satisfaction. Academic Press, 1993.

[19] M. Wooldridge and N.R. Jennings (Eds.). Intelligent Agents - Theories, Architectures, and Languages. *Volume 890 of Lecture Notes in Artificial Intelligence,* Springer-Verlag, January, 1995.

[20] M. Wooldridge, J.P. Mueller, and M. Tambe (Eds.). Intelligent Agents II. *Volume 1037 of Lecture Notes in Artificial Intelligence,* Springer-Verlag, January, 1996.

__Leonid Borisovich Sheremetov__, received his Ph.D from St. Petesburg Institute for Informatics and Automation of the Russian Academy of Sciences (SPIIRAS). His research interests include Multi-Agent Systems, Decision Support Systems, Espert Systems and Distance Learning. Dr. Leonid Sheremetov is a professor of the Agent Laboratory inn CIC-IPN, México.



__Alexander Victorovitch Smirnov__, received is Ph.D from St. Petesburg University of Electric Engineering and D.Sc. from SPIIRAS. His research interest include knowledge Engineering, System Analysis, Multi-Agent Systems, Decision Support Systems, Design Theory, concurrent Engineering and Virtual Enterprises. Prof. Smirnov is a Deputy-Director and head of Computer Integrated Systems Laboratory of SPIIRAS, Russia.