A Multi-Agent Recommender System for Planning Meetings

Santiago Macho, Marc Torrens and Boi Faltings

Artificial Intelligence Laboratory (LIA)
Swiss Federal Institute of Technology (EPFL)
IN-Ecublens, CH-1015, Lausanne
Switzerland
akira@lia.di.epfl.ch, torrens@lia.di.epfl.ch, faltings@lia.di.epfl.ch

Abstract

In this work we address the problem of arranging meetings for several participants taking into consideration constraints for personal agendas and transportation schedules. We have implemented a multi-agent recommender system that solves the problem.

Building such applications implies to consider two main issues: collecting information from different sources on the Internet, and solving the problem itself. We show that multi-agent systems that use constraint satisfaction for modelling and solving problems can be very suitable for this kind of systems.

1 Introduction

In this paper we address the problem of arranging meetings among several people. The problem involves combining personal agendas with transportation schedules in order to find appropriate meeting places, dates and times.

The traditional way of arranging meetings for several participants implies a negotiation by hand of dates and sometimes also places. Every participant has an agenda with some available dates for the meeting. The task of taking a decision for a meeting is not an easy problem, specially in the case that: the participants are quite busy and/or the meeting takes several days and/or they live in different places, etc. The problem becomes more complex if we consider transportation schedules and people have several meetings with different people in different places. The problem could be even more difficult to solve if we take into consideration user's preferences, where every participant has different criteria. In such situations it is mostly impossible to plan meetings in an optimal way by hand.

Basically, our problem is naturally a choice problem. Participants to meetings have to choose among several options. These choices cannot be taken freely because many elements are interconnected, i.e. there are dependencies and incompatibilities between the different choices to take. Considering these factors, the problem of arranging meetings according to transport constraints and personal preferences can be easily formulated as a Constraint Satisfaction Problem (CSP).

With the information about transportation schedules by neutral travel providers and the possibility of interaction among agents on the Internet, the task can be mostly done automatically and thus it could become a useful tool for people. On the other hand, the application demonstrates the utility of using constraint satisfaction for solving complex problems in multi-agent recommender systems.

In order to build up a recommender multi-agent system for planning meetings, we suppose that every participant has an agenda which is accessible by a `Agenda Agent` through Internet. Another agent called `Flight Scheduler Agent` has access to the schedules and availability of flights around the world. Such kind of agent is described in [1]. These two agents are implied in the process of collecting information. Other agents are needed for accomplishing the task: the `Planner Agent` and the `Solver Agent`. All the information implied in the recommender system and the problem modelling are formalised using constraint satisfaction formalism, so it is ubiquitous that the agents communicate each other using a FIPA[1] compliant language called **C**onstraint **C**hoice **L**anguage (CCL[2]) [2]. The solving task which is basically

[1]Foundation for Intelligent Physical Agents: `http://www.fipa.org`

[2]Constraint Choice Language: `http://liawww.epfl.ch/CCL`

to solve a configuration problem is carried out by constraint satisfaction algorithms implemented in the **J**ava **C**onstraint **L**ibrary (JCL[3]) [3].

In the next section we describe how to formalise our problem using Constraint Satisfaction Problems. Then, we show how to solve the problem using information gathering on the Web and constraint satisfaction techniques under the multi-agent framework. Next section is intended for describing the multi-agent architecture for our system, more concretely: the **C**onstraint **C**hoice **L**anguage (CCL), the different agents and the interaction between them. Then, we point out further work and we finish the paper giving some conclusions.

2 Problem Modeling

The problem of arranging meetings is formulated in our framework as a CSP. In the following subsection, we briefly describe CSPs and then we present a concrete way to model our problem by identifying the main components of such formulation.

2.1 Constraint Satisfaction Problems (CSPs)

Constraint Satisfaction Problems (CSPs) (see [4]) are ubiquitous in applications like configuration [5, 6], planning [7], resource allocation [8, 9], scheduling [10] 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 to a given problem. The main advantages of constraint-based programming are the following:

- It offers a general framework for stating many real world problems in a succinct, elegant and compact 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.

Formally, a finite, discrete Constraint Satisfaction Problem (CSP) is defined by a tuple $P = (X, D, C)$ 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.

[3]Java Constraint Library: `http://liawww.epfl.ch/~torrens/JCL`

2.2 The Problem of Arranging Meetings as a CSP

The problem of arranging meetings can be formulated as a choice problem, more specifically as a Constraint Satisfaction Problem (CSP). For simplicity, we consider that we have to plan only one meeting among several participants that lives in different places. Three phases are needed in order to model a problem as a CSP[4]:

1. *Variables:* identify the variables involved in the problem,
2. *Domains:* associate to all variables the appropriate finite domain of discrete values, and
3. *Constraints:* link the constrained variables by means of allowed/disallowed combinations of values.

In our framework, there is a set of n participants ($P = \{P_0, \ldots, P_{n-1}\}$). The meeting has to take place when all the participants P_i are available. In addition, the meeting will be in a set of m possible predefined cities ($C = \{C_0, \ldots, C_{m-1}\}$). Normally, this set of places corresponds to the places where the involved people live.

For each participant P_i we define his/her agenda as a set of k `AgendaSlot` ($AS = \{AS_0, \ldots, AS_{k-1}\}$). An `AgendaSlot` is defined as a `StartSlotTime`[5], an `EndSlotTime`, and a `SlotPlace`.

Next, we identify the variables for our model, what are the associated domains and what kind of constraints the system has to take into consideration. The model has been simplified for a better comprehension of the formalism.

2.2.1 Variables

The variables of the CSP depend on the solution we want to find out. In our case, for each participant (P_i) to the meeting we are interested in: an `OutgoingFlight`$_i$ and a `ReturnFlight`$_i$. For every participant P_i exists three variables for each free `AgendaSlot` in his agenda: `StartFreeTime`$_j$, `EndFreeTime`$_j$ and `Place`$_j$.

Other variables concern the meeting itself: the `StartMeetingTime`, the `EndMeetingTime` and the `MeetingPlace`.

[4]in our framework, we refer CSPs as finite and discrete CSPs.

[5]when we refer to *Time* variables, we include for each value an exact time meaning an hour, day, a month, a year, etc...

2.2.2 Domains

For variables $\texttt{OutgoingFlight}_i$ and $\texttt{ReturnFlight}_i$ the domains are possible flights the participant P_i can take to attend the meeting. At the beginning of the solving process, the system does not know explicitly what are the possible flights for such variables, these domains are only known once the system starts solving the problem and after querying the flight database.

Concerning the variables of the type $\texttt{Start/EndFreeTime}_j$ and $\texttt{Place}_j$, the values are retrieved from the corresponding agendas for every participant and for each free time slot.

The values of the variables concerning the meeting itself are known a priori. In some sense, these values define the problem to solve. For example, if we want to plan a meeting, normally the problem can be stated as:

> "We want to meet next month, from 15^{th} to 23^{rd} in some of the places we live. The meeting will take place during 3 days. We can meet on Saturdays but not on Sundays".

From such a formulation, the system deduces the domains of the variables related to the meeting. In other words, these variables define the problem the system has to solve.

2.2.3 Constraints

Constraints are used for defining the search space and thus the solving algorithms will find well defined solutions. Basically, the constraints involved in our problem are:

- $\texttt{OutgoingFlight}_j$-$\texttt{ReturnFlight}_j$: The return flight has to be taken after the outgoing flight. The arrival place of the outgoing flight must be the same than the departure of return flight.
- $\texttt{OutgoingFlight}_j$-$\texttt{Place}_j$: The departure of the outgoing flight must be the same place as $\texttt{Place}_j$.
- $\texttt{OutgoingFlight}_j$-`MeetingPlace`: The outgoing flight must arrive at the place where the meeting will take place.
- $\texttt{OutgoingFlight}_j$-`StartMeetingTime`: All the participants must arrive before the meeting starts.
- $\texttt{ReturnFlight}_j$-`EndMeetingDate/Time`: All the participants must leave the meeting place after the meeting has finished.
- $\texttt{StartFreeTime}_k$-`StartMeetingTime`: For each user, there must exist at least one $\texttt{StartFreeTime}_k$ which is before the `StartMeetingTime`.
- $\texttt{EndFreeTime}_k$-`EndMeetingTime`: For each user, the $\texttt{EndFreeTime}_k$ must be after the `EndMeetingTime`.
- $\texttt{StartFreeTime}_k$-$\texttt{EndFreeTime}_k$: $\texttt{EndFreeTime}_k$ must be after $\texttt{StartFreeTime}_k$. With this constraint we guarantee that the free time slots are well defined.

3 Problem Solving

In our framework, problem solving is mainly composed of two phases, gathering information and finding solutions:

- Gathering information: the recommender system collects information from different agents in order to model the corresponding CSP. Some domains of the variables are filled in by means of queries to the involved agents. The `Planner Agent` is responsible for requesting to *information agents* the needed information in the appropriate order to build the whole CSP. *Information agents* are, for example, the agent that collects information of the free time slots of different user's agendas (`Agenda Agent`) or the agent that requests schedules and availability of flights (`Flight Scheduler Agent`).
- Finding solutions: once the CSP is built, the solver agent can apply constraint satisfaction algorithms from the JCL and find solutions according to the constraints. When the solutions are found, the system informs to each `Personal Assistant Agent` about the solutions.

4 The Multi-Agent Architecture

The multi-agent recommender system is composed by the following agents (see Fig. 1):

- `Personal Assistant Agent`: is the interface agent between the user and the multi-agent system.
- `Planner Agent`: is the main agent, who is responsible for getting the involved variables, the associated domains and the needed constraints in order to plan the meeting. Basically, this agent composes the whole problem as a CSP.
- `Agenda Agent`: every user has a personal agenda with all the information about meetings and personal constraints. These agendas are accessible by the `Agenda Agent`.
- `Flight Agent`: is connected to a database of flights over the world.

- `Solver Agent`: is the agent responsible to solve the CSP using the JCL algorithms.

In our system, a user that coordinates the meeting plan is charged of inputing the main parameters of the meeting, such as:

- the users willing to attend the meeting,
- how long the meeting will take,
- in what range of dates the meeting must be planned, and
- where the meeting can take place (it is possible to give several optional places).

When the `Personal Assistant Agent` has all the data about the meeting we want to plan, it builds the associated CSP and sends it to the `Planner Agent`. The `Planner Agent` receives a CSP without all the needed variables to recommend the meeting. Variables concerning the user's agendas and variables concerning the flight schedules are not yet present in the CSP. Then, the `Planner Agent` sends the CSP to every `Agenda Agent` of participants involved in the meeting to get constraints about their availability. A similar process is carried out for getting the information about flight schedules between the cities of the users and the city of the meeting. After this process of collecting information, the `Planner Agent` contains a CSP with all variables and constraints about the meeting, the users and the possible flights. At this point, the CSP is ready to be solved. Thus, the `Planner Agent` sends the CSP to the `Solver Agent` in order to solve it. Once the `Solver Agent` has received the CSP, it has to perform two phases: firstly it translates the CSP written in the CCL to the data structures of the JCL, and secondly it uses the JCL algorithms to find solutions to the problem. Once the CSP is solved, the `Solver Agent` has to perform the inverse work. It translates the CSP from the JCL structures to the CCL and sends it to the `Planner Agent`. Then, the `Planner Agent` passes the CSP to the `Personal Assistant Agent` who will show the solutions to each user. In the case that the meeting proposition is accepted, the agenda is updated according to the new meeting.

Our multi-agent system uses ACL message for interacting. As a content language of the ACL messages, we use the Constraint Choice Language (CCL).

4.1 Constraint Choice Language (CCL) [2]

CCL is a FIPA compliant content language based on Constraint Satisfaction techniques. The CCL specification includes semantic foundations, abstract syntax and language ontology. CCL:

Figure 1. The multi-agent architecture.

- is based on constraint satisfaction formalism,
- is suitable for choice problems or CSPs,
- supports communication about CSPs from modelling right through to problem solving, and
- has been incorporated in the FIPA 1999 standard as content language FIPA-CCL.

A traditional way to formulate constraints in discrete CSPs is to define the tuples by an explicit list of allowed or excluded values between the implied variables. The **C**onstraint **C**hoice **L**anguage deals with a slight different notation which simplifies the implementation of constraint engines. In particular, it allows us to express two types of constraints:

- *Exclusion constraints*, which act on a single variable and are specified as a *no-good list*.
- *Relations*, which act on two variables and are restricted to a closed set of seven general types ($=$, $!=$, $<$, $>$, $=<$, $>=$, and not), but can be formulated on tuples.

The use of tuple-valued variables allows the language to handle n-ary constraints by introducing variables whose values represent the tuples allowed by the constraint. The advantage of this formulation is that solving or consistency engines can be restricted to unary and binary constraints.

4.1.1 An example about how to express CSPs using XML

CCL allows agents to express a CSP as we defined it in section 2.1. However, three restrictions on the CSP representation have been made to make the model minimal and more suitable for a communication language:

1. *Binary constraints*: all constraints must be binary, i.e. constraints that involves two variables. This restriction is often made in the CSP community, since most powerful solving techniques only apply to binary CSPs. However, this is only a slight restriction because we can transform n-ary constraints to binary constraints.

2. *Discrete variable domains*: most of the real-world problems like configuration, scheduling or planning can be formulated using CSPS that have variables with domains that contain discrete values. For example, suppose that we want to write in CCL the fact that the variable `MeetingPlace` of the type `String` has as domain the values: {`Zurich, Geneva, Amsterdam`}. This variable would be expressed in CCL as follows:

```
<CSP-variable Name="MeetingCity" Type="string">
    <Domain>
        <CSP-value-list Npart="1">
            <List-values Values=
              "\{Zurich,Geneva,Amsterdam\}"/>
        </CSP-value-list>
    </Domain>
</CSP-variable>
```

3. *Intensional relations*: instead of working with extensional relations between two variables (`good-list` or `no-good-list`) we work with intensional relations ($=$, $!=$, $<$, $>$, $<=$, $>=$ and not). In this way, we facilitate the merge of CSP when collecting information from several sources.

A CSP expressed in CCL is composed by a zone of variables and a zone of constraints (relations), for example:

```
(request
:sender FrontBot@iiop://liasun24.epfl.ch:7999/acc
:receiver planner@iiop://liasun24.epfl.ch:7999/acc
:content  #3318<?xml version="1.0"?>
<!DOCTYPE Expression SYSTEM "CCL.dtd">
<Expression>
    <Action Name="CSP-solve">
        <CSP-solve>
            <CSP CSP-ref="id939978811875 ">
                <!-- ZONE OF VARIABLES >
                <CSP-variable Name=...
                       ...
                </CSP-variable>
                <CSP-variable>
                       ...
                </CSP-variable>
                       ...
                <!-- ZONE OF CONSTRAINTS >
                <CSP-relation Variables=...
                       ...
                </CSP-relation>
                <CSP-relation Variables=...
                       ...
                </CSP-relation>
                       ...
            </CSP>
        </CSP-solve>
  </Action>
</Expression>
:language FIPA-CSP
:protocol fipa-request
:conversation-id liasun24.epfl.ch/128.155.6312188 )
```

In the following sections, we briefly describe how each agent of the system works.

4.2 Personal Assistant Agent

The `Personal Assistant Agent` is the agent that interfaces between the user and the recommender system for planning meetings and travels. With this agent, the user expresses his/her needs and preferences for the meeting. The `Personal Assistant Agent` builds first a CSP with only a few variables (`StartMeetingTime`, `EndMeetingTime`, `MeetingPlace`, etc) and some constraints. Finally it sends the CSP to the Planner agent.

4.3 Planner Agent

The `Planner Agent` is the main agent of the system. It is the responsible for building the whole CSP. When the `Planner Agent` receives the request to solve the CSP, it asks for variables and constraints to the *information agents*: the `Agenda Agents` (variables and constraints of every user) and the `Flight Agent` (variables and constraints about flights). When the `Planner Agent` has all the necessary variables to solve the CSP, it sends the CSP to the `Solver Agent` who will solve the problem.

4.4 Agenda Agent

Every user has an `Agenda Agent` that is connected to his/her agenda database. The `Agenda Agent` is responsible to maintain the personal agenda of every user. This agent queries the agenda in order to know the free time slots of the user. It also updates the agenda when a new meeting is planned.

4.5 Flight Agent

The `Flight Agent` is an agent that is connected to the database of flights (schedules and availability) provided by a neutral travel provider such as Galileo. This agent offers information services about all the flights over the wold.

4.6 Solver Agent

The Solver Agent is the agent responsible for the translation of the CSP written in CCL to the **J**ava **C**onstraint **L**ibrary (JCL) structures. The Solver Agent uses the constraint satisfaction algorithms of JCL for solving the CSPs. When the JCL has solved the CSP, JCL sends back the solutions to the Solver Agent who is responsible for translating the solutions from the internal JCL data structures to the CCL format and send it back to the Planner Agent.

4.7 Interaction between agents

The agents of the multi-agent recommender system for planning meetings are FIPA ACL compliant. In the next subsections we focus on the interaction between the agents. Firstly we show an overview of ACL messages and finally we show the following interactions:

- Personal Assistant Agent - Planner Agent interaction,
- Planner Agent - Agenda Agent interaction,
- Planner Agent - Flight Agent interaction, and
- Planner Agent - Solver Agent interaction

4.7.1 Overview of ACL messages

The FIPA Agent Communication Language (ACL) is based on speech act theory: messages are actions, as they are intended to perform some action by virtue of being sent. The specification consists of a set of message types and the description of their pragmatics, that is the effects on the mental attitudes of the sender and receiver agents. Every communicative act is described with both a narrative form and a formal semantics based on modal logic [11].

In the FIPA ACL specification there is the description of some high-level protocols like request, contract net, several kinds of auctions, etc. Our multi-agent recommender system uses the request protocol shown in Fig. 2. With this protocol, an agent requests another agent to perform an action, and the receiver agent is able to perform it or replay that it can not do it.

The content field of an ACL message contains the expression (action, proposition or object) and the object (CSP, solution-list, etc) which is referred by the expression, all codified in CCL.

Figure 2. The FIPA ACL request protocol.

4.7.2 Personal Assistant Agent - Planner Agent interaction

The Personal Assistant Agent sends a request message with the action CSP-solve to the Planner Agent. This message starts a new conversation between the Personal Assistant Agent and the Planner Agent. We use the FIPA-request protocol, so the possible answers from the Planner Agent to the Personal Assistant Agent are the FIPA ACL messages: not-understood, refuse, or agree.

In Fig. 3 we show how the Personal Assistant Agent and the Planner Agent interact.

The Personal Assistant Agent sends a request message to the Planner Agent with the action CSP-Solve. When the Planner Agent has the solution(s), it sends back the object CSP-Solution if there is only one solution or the object CSP-SolutionList if there are more than one solution.

4.7.3 Planner Agent - Agenda Agent interaction

Once the Agenda Agent (of every user) receives the request from the Planner Agent, it has to query the agenda database to set the constraints. The Agenda Agent of every user add to the original CSP new variables and new constraints. Once the Agenda Agent adds the constraints to the CSP, it sends a inform message (this message is inside the request protocol with the Planner Agent) to the Planner Agent with the proposition CSP-constraints. This message ends

Figure 3. The interaction between the `Personal Assistant Agent` and the `Planner Agent`.

Figure 4. The interaction between the `Planner Agent` and the `Agenda agent`.

the conversation between the `Agenda Agent` and the `Planner Agent`. The `Planner Agent` sends the action `CSP-give-constraints` and receives the object `CSP` that is the original CSP but with the new constraints added by the `Agenda Agent`. In Fig. 4 we illustrate the interaction between the `Planner Agent` and the `Agenda agent`.

4.7.4 Planner Agent - Flight Agent interaction

The CSP needs to add the values and constraints about flights. When the `Flight Agent` receives the message with the CSP, it searches to the flight database the available flights for every user to go to the city(s) of the meeting.

Fig. 5 shows the interaction between the `Planner Agent` and the `Flight Agent`.

4.7.5 Planner Agent - Solver Agent interaction

At the moment that the `Planner Agent` has the CSP with the variables and the constraints that were set by the *information* agents (`Agenda Agent`s of every user and the `Flight Agent`) the `Planner Agent` contains all the necessary information to solve the CSP. In order to solve the CSP, the `Planner Agent` sends a `request` message to the `Solver Agent` with the action `CSP-Solve`. Fig. 6 illustrates this interaction.

Figure 5. The interaction between the `Planner Agent` and the `Flight Agent`.

JCL solves the CSP and returns the solution(s). Remember that the CSP is represented in the content language of the message and written in CCL. The `Solver Agent` is

Figure 6. The interaction between the `Planner Agent` and the `Solver Agent`.

responsible for translating the CSP from the CCL to the JCL data structures. When the `Solver Agent` has found the solution(s) of the CSP, it sends back a message to the `Planner Agent` with the solution(s).

Next subsection describes the **J**ava **C**onstraint **L**ibrary.

4.7.6 Java Constraint Library

We implemented the **J**ava **C**onstraint **L**ibrary (JCL), which allows us to package constraint satisfaction problems and their solvers in compact autonomous agents suitable for transmission on the Internet. It provides services for:

- creating and managing discrete CSPs
- applying preprocessing and search algorithms to CSPs

JCL can be used either in an applet[6] or in a stand-alone Java application. 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 Web and a constraint shell built on the top of this library, allowing CSPs to be edited and solved.

5 Further Work

Many extensions to this work are planned. An interesting future research topic will be how to combine the *gathering information* phase with the *solving problem* phase dynamically. The recommender system could start solving the problem without having all the information in the CSP. Then, the system would collect information when being completely necessary. This idea implies to solve the problem dynamically when searching information. The advantage would be that we will not collect unnecessary information, and the user could get some first solutions very quickly.

[6]An *applet* is an application designed to be transmited over the Internet and executed by a Java-compatible Web browser.

Another interesting issue is how to learn from previous experiences. In the agendas we could have a user profile with a set of predefined preferences (constraints) that will be taken into consideration for next meeting plans.

In the future, we also want to deal with several different kinds of transportation agents (not only flights). In this way, we will be able to plan travels and meetings using different transport means. In this direction, a project with train schedules is already on the track.

Once, the system finds several possible solutions to the problem, users have to choose one of them. This process can be very tedious and difficult since people tend to prefer different options. For avoiding to perform this process manually, we will study some negotiation issues related to multi-agent systems in order to apply such techniques to our framework.

6 Conclusions

In this paper, we have shown that constraint techniques can be very useful for solving complex problems addressed by recommender systems such as the problem of arranging meetings and scheduling travels.

Concretely, we have implemented a multi-agent recommender system which is able to plan meetings using agenda's information and transportation schedules. Agents communicate each other using the **C**onstraint **C**hoice **L**anguage (CCL), a FIPA compliant content language for modelling problems using constraint satisfaction formalism. With this system, we also have shown the utility of using the **J**ava **C**onstraint **L**ibrary (JCL) for solving complex problems in multi-agent systems.

7 Acknowledgements

We thank Steve Willmott and Monique Calisti for the useful comments on modelling the problem of arranging meetings and travels as a CSP, and for building up the Constraint Choice Language in the FIPA framework.

References

[1] Marc Torrens and Boi Faltings. Smart clients: Constraint satisfaction as a paradigm for scaleable in-

telligent information systems. In *Working Notes of the Workshop on Artificial Intelligence on Electronic Commerce, AAAI-99*, Orlando, Florida, USA, 1999.

[2] Steve Willmott, Monique Calisti, Boi Faltings, Santiago Macho Gonzalez, Omar Belakdhar, and Marc Torrens. CCL: Expressions of Choice in Agent Communication. In *The Fourth International Conference on MultiAgent Systems (ICMAS-2000)*, Boston, USA, 2000.

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

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

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

[6] 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.

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

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

[9] 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.

[10] Mark Fox. *Constraint-Directed Search: A Case Study of Job-Shop Scheduling*. Morgan Kaufmann Publishers, Inc., Pitman, London, 1987.

[11] Foundation for Intelligent Physical Agents (FIPA). FIPA Agent Specification 1997. In *Technical Report*, 1997.