A Multi-Agent 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

January 2001

Abstract

In this work we address the problem of arranging meetings for several participants taking into consideration constraints for personal agendas, transportation schedules and accommodation availability. We have implemented a multi-agent 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 and accommodation availability 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 and accommodation availability 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 systems.

In order to build up a multi-agent system for planning meetings, we suppose that every participant has an agenda which is accessible by a Personal Assistant Agent. Another agents are the Flight Travel Agent that has access to the schedules and availability of flights around the world and the Accommodation Hotel Agent that has information about hotels in the world. Such kind of agents are described in [1]. These two agents are implied in the process of collecting information. All the information implied in the system and the problem modelling are formalised using constraint satisfaction formalism, so it is ubiquitous that the agents communicate each other using a FIPA\footnote{Foundation for Intelligent Physical Agents: http://www.fipa.org} compliant language called Constraint Choice Language (CCL\footnote{Constraint Choice Language: http://liwww.epfl.ch/CCL} [2]).
The solving task which is basically to solve a configuration problem is carried out by constraint satisfaction algorithms implemented in the Java Constraint Library (JCL3) [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 Constraint Choice Language (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 Modelling

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 synthesise 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 \( \text{con}(C_i) \). The relation \( R_i \) over the connection of a constraint \( C_i \) is defined by \( R_i \subseteq D_{i1} \times \cdots \times D_{ik} \) and denotes the tuples that satisfy \( C_i \). The arity of a constraint \( C \) is the size of its connection.

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 CSP4:

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.

---

3Java Constraint Library: [http://liaavw.epfl.ch/~terrens/JCL](http://liaavw.epfl.ch/~terrens/JCL)
4In our framework, we refer CSPs as finite and discrete CSPs.
5when we refer to Time variables, we include for each value an exact time meaning an hour, day, a month, a year, etc...
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 and a ReturnFlight. For every participant \( P_i \) exists three variables for each free AgendaSlot in his agenda: StartFreeTime, EndFreeTime, and Place.

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

2.2.2 Domains

For variables OutgoingFlight and ReturnFlight, 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 Start/EndFreeTime and Place, 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:

- **OutgoingFlight\(_j\) - ReturnFlight\(_j\)**: The return flight has to be taken after the outgoing flight. The arrival place of the outgoing flight must be the same place as the departure of return flight.
- **OutgoingFlight\(_j\) - Place\(_j\)**: The departure of the outgoing flight must be the same place as Place\(_j\).
- **OutgoingFlight\(_j\) - MeetingPlace\(_j\)**: The outgoing flight must arrive at the place where the meeting will take place.
- **OutgoingFlight\(_j\) - StartMeetingTime\(_j\)**: All the participants must arrive before the meeting starts.
- **ReturnFlight\(_j\) - EndMeetingTime\(_j\)**: All the participants must leave the meeting place after the meeting has finished.
- **StartFreeTime\(_k\) - StartMeetingTime\(_k\)**: Obviously, for each user, there must exist at least one StartFreeTime\(_k\) which is before the StartMeetingTime.
- **EndFreeTime\(_k\) - EndMeetingTime\(_k\)**: For each user, the EndFreeTime\(_k\) must be after the EndMeetingTime.
- **StartFreeTime\(_k\) - EndFreeTime\(_k\)**: EndFreeTime\(_k\) must be after 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: every Personal Assistant Agent 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 Personal Digital 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 about the availability of rooms in a hotel (Accommodation Hotel Agent) or the agent that requests schedules and availability of flights (Flight Travel Agent).
Finding solutions: once that every Personal Assistant Agent involved in the meeting build the CSP, they can apply constraint satisfaction algorithms from the JCL and find solutions according to the constraints. The Personal Assistant Agent that proposed the meeting receives the solutions of the others Personal Assistant Agents involved in the meeting, and compares them, informing the user about the solution that satisfies all the constraints of the CSPs.

4 The Multi-Agent Architecture

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

- **Personal Assistant Agent**: is the interface agent between the users and the multi-agent system.
- **Flight Travel Agent**: is connected to a database of flights over the world.
- **Accommodation Hotel Agent**: is the agent responsible to find an accommodation on the cities involved in the meeting.

Every agent have different data. A Personal Assistant Agent can organise a meeting or participate giving a partial solution based on the availability of its user. The other agents will complete the data needed to organise the meeting. They cannot interact directly with the users.

![Figure 1: The multi-agent architecture.](image)

In our system, a user that propose 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).

This user uses the Personal Assistant Agent to input the data into the multi-agent system. When the Personal Assistant Agent has all the data about the meeting we want to plan, it builds the associated CSP using CCL⁶. This agent is the responsible to add data to the CSP using the personal agenda of the user. Variables concerning the hotels availability and variables concerning the flight schedules are not yet present in the CSP. Then, the Personal Assistant Agent sends the CSP to the Accommodation Hotel Agent to get constraints about hotel availability in the city of the meeting. A similar process is carried out for getting the information about flight schedules between the cities of the user and the city of the meeting. At this point, the CSP is ready to be solved. The Personal Assistant Agent 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. Solutions only deal with the constraints of the user that proposed

---

⁶CCL = Constraint Choice Language. See next section
the meeting without deal with the constraints of the other users involved in the meeting. The other Personal Assistant Agents involved in the meeting do the same process: query the agenda of their users, send the CSP to complete it with constraints about hotels and flights and finally solve it. When the Personal Assistant Agents have the solutions, they send them to the Personal Assistant Agent that proposed the meeting. This agent compares the solutions from the others agents with its solution, and show to the user solutions that satisfies the constraints of all the users. In the case that the meeting proposition is accepted, the agenda is updated according to the new meeting. It sends a message to the other agents involved in the meeting with the solution chosen by the agent that proposed the 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:

- 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 Constraint Choice Language 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 (\(=\), \(\neq\), \(<\), \(>\), \(\leq\), \(\geq\), and *goodlist*), 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:

```xml
<CSP-variable Name="MeetingCity" Type="string">
  <Domain Type="String">
    <CSP-value-list>
      <List>
        <li>Zurich </li>
        <li>Geneva </li>
        <li>Amsterdam </li>
      </List>
    </CSP-value-list>
  </Domain>
</CSP-variable>
```
3. *Intensional relations*: instead of working only with extensional relations between two variables (good-list or no-good-list) we also work with intensional relations (=, ≠, <, >, <=, >=). 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:

```
<Action Name="CSP-solve">
  <CSP-solve>
    <CSP CSP-ref="id939978811975 ">
      <!-- ZONE OF VARIABLES 
      <CSP-variable Name=".
      ...
      </CSP-variable>
      ...
      </CSP-variable>
      ...
      </CSP-variable>
      <!-- ZONE OF CONSTRAINTS 
      <CSP-relation Variables=".
      ...
      </CSP-relation>
      ...
      </CSP-relation>
      ...
      </CSP>
    </CSP-solve>
  </Action>
</Expression>
```

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

### 4.2 Personal Assistant Agent

The *Personal Assistant Agent* is the agent that interfaces between the user and the multi-agent 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 extracts from the agenda of the user. Then, it asks for variables and constraints to the *information agents*: the *Accommodation Hotel Agent* (variables and constraints about accommodation availability) and the *Flight Agent* (variables and constraints about flights). When the *Personal Assistant Agent* has all the necessary variables, it uses the constraint satisfaction algorithms of JCL for solving the CSP. It also updates the agenda when a new meeting is planned. The user can modify the meeting using the *Personal Assistant Agent*. If he/she modifies the data of the meeting planned, then the system must replan the meeting starting the process with the agents involved in it.

### 4.3 Flight Travel 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 world. It is proactive, if a flight is cancelled due the weather or another problems, it sends a message to all the *Personal Assistant Agents* to replan meetings if necessary.

### 4.4 Accommodation Hotel Agent

The *Accommodation Hotel Agent* is an agent that is connected to the database of hotels in order to book a hotel in the city of the meeting. Also as the *Flight Travel Agent* the database could be provided by a neutral provider.
4.5 Interaction between agents

The agents of the multi-agent 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 - Flight Travel Agent interaction,
- Personal Assistant Agent - Accommodation Hotel Agent interaction and
- Personal Assistant Agent\_A - Personal Assistant Agent\_B

4.5.1 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 [1].

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 system uses the inform and the request protocol shown in Fig. 2. With the inform protocol an agent informs another agent about a situation. With the request protocol, an agent requests another agent to perform an action, and the receiver agent is able to perform it or replay that it cannot do it.

![Figure 2: The FIPA ACL request protocol.](image)

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.

4.5.2 Personal Assistant Agent - Flight Travel Agent interaction

The Personal Assistant Agent needs to add to the CSP the values and constraints about flights. When the Flight Travel Agent receives the message CSP\_give\_constraints with the CSP, it searches to the flight database the available flights for the user to go to the city of the meeting.

We use the FIPA-request protocol, so the possible answers from the Flight Travel Agent to the Personal Assistant Agent are the FIPA ACL messages: not-understood, refuse, or agree.

Fig. 3 shows the interaction between the Personal Assistant Agent and the Flight Travel Agent.

![Figure 3: The interaction between the Personal Assistant Agent and the Flight Travel Agent.](image)
4.5.3 Personal Assistant Agent - Accommodation Hotel Agent interaction

The interaction between the Personal Assistant Agent and the Accommodation Hotel Agent is similar to the interaction described above. The Personal Assistant Agent needs to add to the CSP values and constraint about hotels. The Accommodation Hotel Agent searches in the hotels database the availability of rooms in the city of the meeting.

Fig. 4 shows the interaction between the Personal Assistant Agent and the Accommodation Hotel Agent.

![Figure 4: The interaction between the Personal Assistant Agent and the Accommodation Hotel Agent.](image)

4.5.4 Personal Assistant Agent\(_A\) - Personal Assistant Agent\(_B\) interaction

The Personal Assistant Agent\(_A\) sends a request message with the action CSP-solve to the Personal Assistant Agent\(_B\)\(^7\). This message starts a new conversation between the Personal Assistant Agent\(_A\) and the Personal Assistant Agent\(_B\).

![Figure 5: The interaction between the Personal Assistant Agent\(_A\) and the Personal Assistant Agent\(_B\).](image)

The Personal Assistant Agent\(_A\) sends a request message to the Personal Assistant Agent\(_B\) with the action CSP-Solve in order to start the solving process in the Personal Assistant Agent\(_B\). When it 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 in order to compare with the solutions found by the Personal Assistant Agent\(_A\).

Next subsection describes the Java Constraint Library.

4.5.5 Java Constraint Library

We implemented the Java Constraint Library (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\(^8\) 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.

\(^7\)To simplify the example we suppose that there are only 2 users involved in the meeting. If there are more users, the Personal Assistant Agent\(_A\) sends a request message to every participant

\(^8\)An applet is an application designed to be transmitted over the Internet and executed by a Java-compatible Web browser.
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 multi-agent 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.

Another interesting issue is how to learn from previous experiences. In the Personal Assistant Agent 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.

A possible application of our multi-agent system is as a portable planning. Mobile phones of the next generation could be programmed and we could integrate the Personal Assistant Agent inside a mobile phone, having a powerful planning between different users (see Fig. 6).

Figure 6: An application of our multi-agent system.

6 Conclusions

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

Concretely, we have implemented a multi-agent system which is able to plan meetings using agenda’s information, availability of hotels and transportation schedules. Agents communicate each other using the Constraint Choice Language (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 Java Constraint Library (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


