A Testbed for Multiagent Systems
Technical Report IIIA-TR-2009-09

Angela Fabregues and Carles Sierra

October 19, 2009

IIIA: Institut d'Investigaci en Intel·ligncia Artificial
CSIC: Spanish Scientific Research Council, UAB
08193 Bellaterra, Catalonia, Spain
`{fabregues,sierra}@iiia.csic.es`

Abstract

There is a chronic lack of shared application domains to test the research models and agent architectures on areas like negotiation, argumentation, trust and reputation. In this paper we introduce such a friendly testbed that we used for all such purposes. The testbed is based on the Diplomacy Game due to its lack of random moves and because of the essential role that negotiation and the relationships between the players play in the game. The testbed may also profit from the existence of a community of bot developers and a large number of human players that would provide data for our experiments. We offer the infrastructure and make it freely available to the MAS community.

***Keywords*: application, testbed, diplomacy game**

1 Introduction

Current research trends in multiagent systems include models of trust [6], reputation [14], and argumentation [5, 4] to improve negotiating strategies [7, 8, 10]. Research progress in the development of these theoretical models has made them very sophisticated (based on cognitive, information or game theoretical grounds) and has enabled software agents to interact with and help humans in a more efficient and believable way. Agents are being endowed with techniques to decide how and when to interact, argue and negotiate, as well as whom to trust and why. Unfortunately, the practical application of the models has been less than expected. This is partly due to the fact that researchers lack shared use cases for comparing the models, the use cases they design tend to be rather artificial and usually biased to a particular model and therefore with little reuse. As a result, most experiments suffer from an insufficiency of both data and computational resources for model validation. Environments are urgently needed that are rich enough to test simulations without having to oversimplify the models on which they are based. In particular, software agents must be able to interact not just with humans but also with other software agents. We argue that the popular strategy board game Diplomacy provides just such a testbed. We are building the necessary infrastructure to use Diplomacy as an environment for a testbed for MAS and in this paper we report our results on that.

We start this paper introducing the Diplomacy game and its player community, sections 2 and 3, to then focus on the agents that already play diplomacy and their valuable use for research in MAS, sections 4 and 5. Then we present the formal languages that we propose to use among the agents in our testbed as a tower of increasingly expressive languages, section 6. Section 7 illustrates the communication protocols. And finally, in section 8 the testbed architecture is described in detail with some results on the validation of the system and its use for research purposes. The paper ends with the Related Work and Discussion sections.

2 Diplomacy

In Diplomacy, players negotiate with the aim of conquering Europe. The rules of the game are unambiguous and the available information about the game is rather rich as it is a quite old game being enjoyed by several generations of players. The game appeared in the 1954s. The game is situated on the Continent at the beginning of the 20^{th} century, before World War I. Each player is in charge of the armed forces, organised in *units*, of a major European power and must decide, in each turn, which movements the various units should execute. The game ends when someone has an army powerful enough to control half of the European 'provinces'. This is achieved by defeating other players' units, conquering their provinces and controlling the supply centres that allow armies to build new units.

One of the most interesting features of Diplomacy is the absence of random movements: there are no cards and no dices. Also, this is not a turn-taking game. That is, all players move their units simultaneously: there is no advantage in a player being the first or last to move. Moreover, all units are equally strong. Consequently, when one attacks another, the winner of the battle is decided by taking into account only the number of units helping one another. This feature is what makes Diplomacy so compelling for our purposes. Accordingly, the most relevant skills for a player are the negotiating ability, the intuition (knowing whom to trust) and the persuasive power.

The game is not difficult. Indeed, the individual plays are quite simple. What is difficult is to resolve the conflicts that crop up owing to the simultaneous public announcement of the movements. Expert players, or masters, usually perform this task. Recently, however, software programs have replaced the masters. From a player's point of view, the most important aspect of the game is the negotiation process: deciding allies, selecting whom to ask for help, arguing with other players to get information about their objectives or to find out what they know, building trust and reputation, maintaining relationships, and so on.

3 Player Community

Diplomacy is often played on the Internet. Interestingly, playing online makes it easier to secretly meet with other players to negotiate and keep conspiracies under wraps.

The larger online community of players meets around *The Diplomatic Pounch*, [2]. People there uses *nJudge*, a software developed in 1987 by Ken Lowe that allows players to play Diplomacy by e-mail and provides an automatic order resolution.[1] This community has an online magazine and a wiki.

A frequent option is to use a web application with a friendly interface for playing Diplomacy online. There are basically two sites: *webDiplomacy* and *playDiplomacy*. Both sites allow playing online and sharing information with other members of the community. The community of *webDiplomacy* play with *phpDiplomacy*, an opensource project that makes internal use of *nJudge*. Although *playDiplomacy* does not even explain how it works, it has a larger social network.

Other software projects can be found on the net that try to help in either resolving the orders (adjudicators) or providing a nice interface for playing (mappers), for example: *jDip* [3].

Besides all the games played on the net using the mentioned software tools, a lot of tournaments of Diplomacy are being organized to play face-to-face.[2] Note that sometimes two players have no common natural language. In that case, they use 'translator sheets' to support communication.[3]

4 Agents playing Diplomacy

The idea of creating a Diplomacy software player (bot) was first suggested by Sarit Kraus 20 years ago [9]. Since then, several other researchers have also tried to build bots but without much impact [13, 15]. Despite this little success, we think that now is the moment to continue this work. We believe this due to the fact that computer and network technologies have evolved so much in the interim that many people now accept entertainment online from their homes as a matter of course. In fact, there is a community around the Diplomacy Artificial Intelligence Development Environment (DAIDE, [1]) that has defined a communication protocol [12] and a message syntax [11] that can be considered a standard for bot communication, specially the level 0 syntax, as the other levels are almost never used. They built an infrastructure to develop bots and compare their performance. Many developers have joined the DAIDE community and develop their own bots to compete.

[1] The resolution of orders (i.e. actions performed by units) in Diplomacy is often called the adjudication process.

[2] *The Diplomatic Pounch* maintains the list of Diplomacy face to face tournaments at `http://www.diplom.org/Face/cons`.

[3] Some examples of these sheets can be found in `http://www.ellought.demon.co.uk/dip_translator/`.

Up to now, the bot development efforts have focused mostly on the strategy and tactics of the no-press variant of the game (i.e., without dialectic communication between players). The work that the members of DAIDE have done is relevant, although the next natural step would be to enrich the bots by adding negotiation capabilities. Currently, our work concentrates in creating the testbed itself, making the best use of existing resources insofar as possible and providing tools for analyzing how bots (and humans) behave during a game. The development of our own bots to compete with other bots and human players is ongoing work.

5 Diplomacy for research

Diplomacy is, however, strategically simple for a human to play in comparison to other classic games like Chess and Go. This is because the true complexity of the game lies in the relationships among its players. Those are constantly changing and difficult to analyze but humans negotiate constantly in their every day life. They are used to do it, therefore Diplomacy does not seem really difficult to play. On the other hand, it is difficult for a computer as it cannot take advantage from massive computations because the search space in Diplomacy is huge and because the key for success relies on the information obtained from negotiation and the persuasive capability of the player.

Focusing only on the possible moves that the units can perform on the board, the numbers are very large. There is an average of 30 units on the board during a game. An order has to be assigned to each one of these units per turn. The movements that a unit can perform depend on the number of direct neighbours and neighbour units.[4] Assuming an adjacency factor of 4 and a neighbourhood factor[5] of 2, we obtain that the number of possible movements per unit and phase is $30 \cdot 15$.[6] Overall, a branching factor of 450.

The branching factor of the search is thus so high that even a no-press game cannot be treated by standard search mechanisms. Neither can we reasonably use game theory to solve the problem strategically. Think that the branching factor for chess is around 35. Besides that, the unit movements in Diplomacy depend on the movements done by the other units. The moves that a player performs are not independent from the ones performed by other players. All players perform their moves at the same time, hence a player cannot be sure about the outcome of a move because there can be conflicts between different players' orders. Therefore, it is very difficult to predict the outcome of a movement without information about the opponent's intentions. In fact, the essence of Diplomacy relies on the diplomatic moves that were not taken into account when analyz-

[4]By direct neighbour we refer to units that are in an adjacent region. And neighbor is used for those units that are placed in a region that is at distance two, that is, they are units of a direct neighbour of a direct neighbour.

[5]Number of units that are neighbour.

[6]The average number movements for a unit in every phase are: 1 hold, 4 movements to the 4 adjacent regions, 2 supports to hold to the 2 neighbors and $4 \cdot 2$ supports to move for every two neighbors of our 4 adjacent regions. Therefore $1 + 4 + 2 + 4 \cdot 2 = 15$.

ing the complexity above. Should we take every single negotiation step into consideration, then the number of possible moves would be simply overwhelming as Sarit Kraus already observed in [9].

From the point of view of AI research, Diplomacy is a multiagent system environment where competitive self interested agents need to cooperate to obtain better outcomes. This is done by the signature of agreements where agents involved commit to do a plan of action. Agreements in such environments are reached as the result of successful negotiation processes in which agents dialoge exchanging proposals and information with the aim of convincing the other agent to accept a deal, sometimes arguing. Because of the repetition of negotiation dialogues, negotiations get quicker since most of the previous discussed agreement issues do not need to be discussed again. The agents can guess which are the believes, desires and intentions of other agents just analyzing the past dialogues and the state of the game. And as time goes by, agents can observe how their counterparts honour up the agreements they sign. This information can be used to build a model of agents' behaviour. This model will help in future negotiations, even to decide which agent should we negotiate with. Concepts like trust, honor, sincerity, and others can summarize the perception we can get from an agent as time goes by. Also the reputation of an agent can be taken into account because agents can also talk (gossip) about other agents performance, promises, intentions, ... From the last sentences, it is straightforward to see that several MAS research topics are highly relevant: negotiation, trust, reputation, argumentation. But also other topics like coalition formation, behavior emergence or ways of persuasion can be studied with this testbed.

By means of this testbed, we join together research and entertainment as it was done previously in other projects[7], with the aim of providing an infrastructure to play Diplomacy comfortably —using a web or desktop based graphical application, where and when the player wants to play —we provide bots so the player does not need to wait for other players writing their orders, using each player its own language —a software tool will perform something similar to the translator sheets mentioned earlier, and all for free. At the same time, the testbed will permit the gathering of valuable information that can be useful to support experimental research in MAS and AI.

The 'translator sheets' introduced before in Section 3 is another beneficial aspect of the testbed as it restricts the language that humans use and allows software agents to easily step in by just knowing the language primitives. In other words, there is no need to deal with natural language making the programming of bots much easier while hiding the possibility that humans recognise whether the oponents are human or not.

The testbed is rather new and it is not really popular yet but we expect to have a lot of users in a short time. Those users will provide a lot of data that we will be able to work with. We plan to provide tools for social networking that can give us extra information about the social relationships of the users and how they affect the game. Diplomacy games

[7]In 2005, Luis von Ahn devised the ESP Game, an online game of image labeling that Google is now using to improve its image search results.

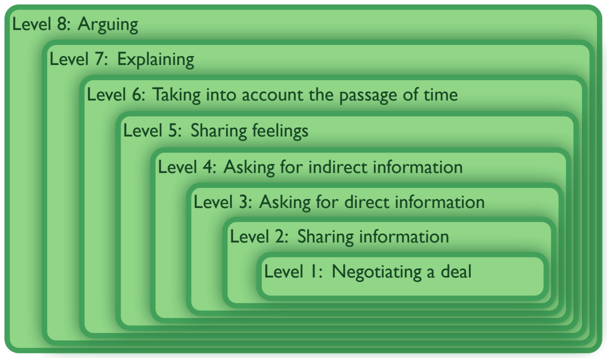

Level 8: Arguing
Level 7: Explaining
Level 6: Taking into account the passage of time
Level 5: Sharing feelings
Level 4: Asking for indirect information
Level 3: Asking for direct information
Level 2: Sharing information
Level 1: Negotiating a deal

Figure 1: Language Levels

are usually played without revealing the player identity. But we can also allow games where players meet beforehand to increase/modify the social relationships.

Independently on the type of game, with hidden names or not, a *software mediator* could be able to help human players to play Diplomacy using information of past games. A mediator can assist a player in the decision making process just showing his/her past deals and their outcomes. Or even arguing to try and convince the human player to do what the software mediator ‘thinks’ is better. In this way, human players can be more receptive to the idea of having software agents helping them and the human players can also indicate the mediators what is preferable for them and why.

The proposal is thus to use Diplomacy as a testbed that combines humans and agents.

6 Language

In MAS as well as in Diplomacy, one of the most important features is the communication language between agents. This is done in our testbed using a shared language L described in this section. Because of the complexity of building an agent capable of understanding a complex language, we define it as an 8 level language; starting from L_1 and increasing the expressiveness as the language level increases. Figure 6 represents graphically the language level hierarchy.

L is a generic language that could be used for many other applications. It defines the illocutions that the agents can use to communicate and the basic concepts like *Agree*, *Desire*, *Feel*, etc. The language is parametric on the vocabulary for a specific application domain, described as an ontology.

Bots might be programmed to use languages with different expressive power. We think that the expressive ingredients that are most relevant for MAS research are: cognitions, information exchange, agreements, emotions, time, and argumentation. For instance, we could have argumentative bots that would not use time, or cognitive bots that have no emotions. These combinations account for a large number of languages that we can imagine as nodes in a graph where a directed arc related a less expressive language to a more expressive language. In this section, for simplicity we will select a representative subset of those possible languages and will present them as a particular path in such graph. Language L_i in this section is more expressive than L_j if and only if $i > j$.

In Figure 2 we present a compressed version of the language level definition expressed in BNF.[8] Each language extends the languages in lower levels, that is, if there is no re-writing rule for a term in L_i then it can be found in lower levels L_j, with $j < i$. The undefined non terminal symbols that appear in L should be specifically defined for each application domain. These symbols are: *time*, *agent*, *action* and *predicate*. The higher the language level that we use, the more complex the actions and the predicates.

In the rest of this section, we describe and illustrate the expressivity of each language level. The examples use the Diplomacy ontology introduced in Figure 3. To that end, we define the symbols:

$$\begin{aligned}
&\mathit{agent} ::= \mathit{power}\\
&\mathit{action} ::= \mathit{order}\\
&\mathit{predicate} ::= \mathit{offer}\\
&\mathit{time} ::= \langle \mathit{phase}, \mathit{year}\rangle
\end{aligned}$$

For instance, `Unit`(`rus`, `Region`(`stp`, `scs`)) is a term meaning that 'There is a unit from Russia in the south coast of Saint Petersburg', `pce`({`ita`, `rus`}) is a predicate meaning 'Peace between Italy and Russia', and `sup`(`Unit`(`rus`, `Region`(`spa`, `ecs`)), `mto`(`Unit`(`ita`, `Region`(`mar`, `amy`)), `Region`(`par`, `amy`))) is an example of action where 'The unit of Russia in the east coast of Spain supports the movement of the italian army in Marseilles to Paris'.

L_1**: Negotiating a deal.** This is the first language level. It allows agents to negotiate deals following the protocol described in Section 7. The deals can be either two sets of commitments, one for every agent involved in the deal, or a global agreement in which a set of agents agree on something, usually the truth of a predicate. Here you have two examples of sentences in L_1:
'Italy proposes to Russia a deal by which Italy commits to do a movement from its army in Marseilles to Paris and Russia commits to support the Marseilles italian army's movement with the unit in the east coast of Spain':

```
propose(ita, rus,
    {Commit(ita,rus,
        Do(mto(Unit(ita, Region(mar, amy)),
              Region(par, amy)))),
```

[8]Note that: *expression*$^+$ ::= [*expression expression ... expression*], non terminal symbols are written in italic, and undefined symbols (referring to terms in the ontology) appear in underlined italics.

Level 1: Negotiating a deal

L_1 ::= propose$(\alpha, \beta, deal_1)$ | accept$(\alpha, \beta, deal_1)$ |
reject$(\alpha, \beta, deal_1)$ | withdraw(α, β)
$deal_1$ ::= Commit$(\alpha, \beta, \varphi)^+$ | Agree(β, φ)
φ ::= $\underline{predicate}$ | Do($\underline{action}$) | $\varphi \wedge \varphi$ | $\neg\varphi$
β ::= α^+
α ::= $\underline{agent}$

Level 2: Sharing information

L_2 ::= L_1 | inform$(\alpha, \beta, info_2)$
$info_2$ ::= $deal_1$ | Obs(α, β, φ) | Belief(α, φ) | Desire(α, φ) |
$info_2 \wedge info_2$ | $\neg info_2$

Level 3: Asking for direct information

L_3 ::= L_2 | inform$(\alpha, \beta, info_3)$ | query$(\alpha, \beta, info_3)$ |
answer$(\alpha, \beta, info_3)$
$info_3$ ::= $info_2$ | Unknown$(\alpha, info_3)$ | $info_3 \wedge info_3$ | $\neg info_3$

Level 4: Asking for indirect information

L_4 ::= L_3 | inform$(\alpha, \beta, info_4)$ | query$(\alpha, \beta, info_4)$ |
answer$(\alpha, \beta, info_4)$ | inform(α, β, L_4) | query(α, β, L_4) |
answer(α, β, L_4)
$info_4$ ::= $info_3$ | Unknown$(\alpha, info_4)$ | Unknown(α, L_4) |
$info_4 \wedge info_4$ | $\neg info_4$

Level 5: Sharing feelings

L_5 ::= L_4 | inform$(\alpha, \beta, info_5)$ | query$(\alpha, \beta, info_5)$ |
answer$(\alpha, \beta, info_5)$ | inform(α, β, L_5) | query(α, β, L_5) |
answer(α, β, L_5)
$info_5$ ::= $info_4$ | Unknown$(\alpha, info_5)$ | Unknown(α, L_5) |
Feel$(\alpha, feeling)$ | $info_5 \wedge info_5$ | $\neg info_5$
$feeling$ ::= $VeryHappy$ | $Happy$ | Sad | $Angry$

Level 6: Taking into account the passage of time

L_6 ::= L_5 | propose$(\alpha, \beta, deal_6, t)$ | accept$(\alpha, \beta, deal_6, t)$ |
reject$(\alpha, \beta, deal_6, t)$ | withdraw(α, β, t) | inform$(\alpha, \beta, info_6, t)$ |
query$(\alpha, \beta, info_6, t)$ | answer$(\alpha, \beta, info_6, t)$ | inform(α, β, L_6, t) |
query(α, β, L_6, t) | answer(α, β, L_6, t)
$info_6$ ::= $info_5$ | $deal_6$ | Obs$(\alpha, \beta, \varphi_6, t)$ | Belief(α, φ_6, t) |
Desire(α, φ_6, t) | Unknown$(\alpha, info_6, t)$ | Unknown(α, L_6, t) |
Feel$(\alpha, feeling, t)$ | $info_6 \wedge info_6$ | $\neg info_6$
$deal_6$ ::= $deal_5$ | Commit$(\alpha, \beta, \varphi_6, t)^+$ | Agree(β, φ_6, t)
φ_6 ::= $\underline{predicate}$ | Do($\underline{action}$, t) | $\varphi_6 \wedge \varphi_6$ | $\neg\varphi_6$ | $\varphi_6; \varphi_6$
t ::= $\underline{time}$

Level 7: Explaining

L_7 ::= L_6 | inform$(\alpha, \beta, info_7, t)$ | query$(\alpha, \beta, info_7, t)$ |
answer$(\alpha, \beta, info_7, t)$ | inform(α, β, L_7, t) | query(α, β, L_7, t) |
answer(α, β, L_7, t)
$info_7$ ::= $info_6$ | Unknown$(\alpha, info_7, t)$ | Unknown(α, L_7, t) |
Explain$(info_7, t)$ | Explain(L_7, t) | $info_7 \wedge info_7$ | $\neg info_7$

Level 8: Arguing

L_8 ::= L_7 | inform$(\alpha, \beta, info_8, t)$ | query$(\alpha, \beta, info_8, t)$ |
answer$(\alpha, \beta, info_8, t)$ | inform(α, β, L_8, t) | query(α, β, L_8, t) |
answer(α, β, L_8, t)
$info_8$ ::= $info_7$ | Unknown$(\alpha, info_8, t)$ | Unknown(α, L_8, t) |
Explain$(info_8, t)$ | Explain(L_8, t) | Attack$(info_7, info_7)$ |
Support$(info_7, info_7)$ | $info_8 \wedge info_8$ | $\neg info_8$

Figure 2: Language Levels

year ::= integer

phase ::= spr | sum | fal | aut | win

power ::= fra | eng | tur | rus | ita | aus | ger

coast ::= ncs | scs | ecs | wcs

regionType ::= amy | sea | coast

supplyCenter ::= spa | mar | par | stp | ...

province ::= *supplyCenter* | gas | bur | sil | tus | ...

region ::= Region(*province*, *regionType*)

unit ::= Unit(*power*, *region*)

order ::= hld(*unit*) | mto(*unit*, *region*) | sup(*unit*, hld(*unit*)) | sup(*unit*, mto(*unit*, *region*)) | rto(*unit*, *region*) | dsb(*unit*) | bld(*unit*) | rem(*unit*) | wve(*power*)

offer ::= pce($power^+$) | aly($power^+$, $power^+$)

Figure 3: Summary of the Diplomacy ontology. See the complete Diplomacy ontology at:
`http://www.iiia.csic.es/dip/language`

```
Commit(rus, ita,
      Do(sup(Unit(rus, Region(spa, ecs)),
             mto(Unit(ita, Region(mar, amy)),
                 Region(par, amy))))})
```

'Italy accepts to Agree with Russia that they are allied against England':

accept(`ita`, `rus`, Agree({`ita`, `rus`}, `aly`({`ita`, `rus`}, `eng`)))

L_2**: Sharing information.** This language level adds the ability of sharing information with other agents. It can be information about previous commitments, observed actions, beliefs, desires or deals.
E.g. 'Italy informs England that Italy keeps an agreement of peace with Russia':

inform(`ita`, `eng`, Agree({`ita`, `rus`}, `pce`({`ita`, `rus`})))

L_3**: Asking for direct information.** At level three, agents can request other agents for information. Answers to queries are similar to informs.
E.g. 'England asks Italy if Italy and Russia have an agreement of peace':

query(`eng`, `ita`, Agree({`ita`, `rus`}, `pce`({`ita`, `rus`})))

'Italy answers England that Italy and Russia do have an agreement of peace':

answer(`ita`, `eng`, Agree({`ita`, `rus`}, `pce`({`ita`, `rus`})))

L_4**: Asking for indirect information.** Level four allows to inform about dialogical moves between agents.
E.g. 'Russia asks Italy whether Italy answered to England that Italy and Russia had an agreement of peace':

```
query(rus, ita,
      answer(ita, eng,
             Agree({ita, rus}, pce({ita, rus}))))
```

L_5**: Sharing feelings.** This level is the emotional one. Feelings can be exchanged between agents.
E.g. 'Italy asks Russia whether Italy answering England that Italy and Russia had an agreement of peace made Russia feel sad':

```
query(ita, rus,
      answer(ita, eng,
             Agree({ita, rus}, pce({ita, rus}))) →
      Feel(rus, Sad))
```

L_6**: Taking into account the passage of time.** L_6 aggregates time to L_5. With L_6, we can talk about the past and make promises for the future. The time is added as an extra argument to predicates and illocutions. But it is not necessary to specify it, as the expressions of L_5 are also expressions of L_6.
E.g. 'Russia informs Italy that if Italy informs in the future to any power that Italy and Russia have and agreement of peace, then Russia will feel Angry':

```
inform(rus, ita,
       (inform(ita, power,
               Agree({ita, rus},
                     pce({ita, rus})), $t_1$) ∧ $t_1 > t_0$) →
      (Feel(rus, Angry, $t_2$) ∧ $t_2 > t_1$),
      $t_0$)
```

L_7**: Explaining.** Dialogues often include explanations and explanation requests. This level adds that possibility to allow agents to explain why things are like they are.
E.g. 'Italy asks Russia for an explanation of why the fact that someone (power in the expression) beliefs that there is an agreement of peace between Russia and Italy makes Russia feel Angry':

```
query(ita, rus,
      Explain(
          Belief(power,
                 Agree({ita, rus}, pce({ita, rus}))) →
          Feel(rus, Angry)))
```

L_8**: Arguing.** And finally, level 8 allows agents to express rebuts and supports between arguments.
E.g. 'Russia informs England that its alliance with Italy against England and Italy's desire of Paris support the imminent Italian attack from Marseilles to Paris':

```
inform(rus, eng, Support(
   Agree({ita, rus},aly({ita, rus},eng)) ∧ Desire(ita, par),
   Do(mto(Unit(ita, Region(mar, amy)), Region(par, amy)))
   ))
```

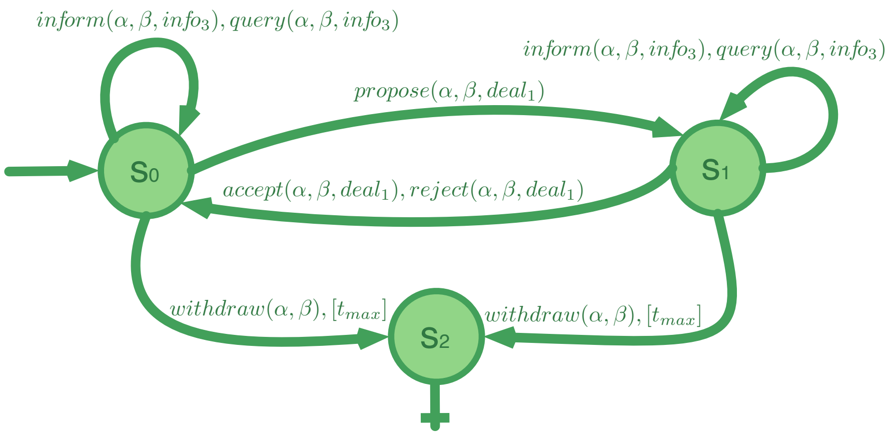
7 Protocol

For a correct understanding of the dialogues that can be generated with the different languages it is necessary also to share a communication protocol. A protocol regulates the interaction between agents letting the agents make proposals, accept or reject them. This protocol will be often used to negotiate between only two agents. But, as you can see in the example of Figure 4, it works with multi-partner negotiation also. We propose to use a different protocol for each language level satisfying definition 1, but only show one here. For details on other protocols go to `URLblinded`. A protocol is understood as a finite state machine where the nodes represent negotiation states and arcs represent transitions between states. The labels of the arcs correspond to expressions of a particular language or timeouts. When the negotiation is in a given state and an agent utters an illocution that matches the label of an outgoing arc the transition is performed. When we reach a state and stay there for as much time as the timeout value labelling an outgoing arc, the transition is made.

Definition 1 *A communication protocol for language level i is a finite state machine $p = \langle S, s_0, S_f, E, \lambda \rangle$, where:*

- S is the set of negotiation states.
- $s_0 \in S$ is the initial state.
- $S_f \subseteq S$ is the set of final states.
- $E \subseteq S \times S$ represents transitions between states.
- $\lambda : E \mapsto L_i \cup \mathbb{N}$ is a labeling function that assigns to every transition a term of the language L_i or a natural number representing a timeout.

An example of protocol for L_3 is represented graphically in Figure 4.



$inform(\alpha, \beta, info_3), query(\alpha, \beta, info_3)$
$inform(\alpha, \beta, info_3), query(\alpha, \beta, info_3)$
$propose(\alpha, \beta, deal_1)$
S0
S1
$accept(\alpha, \beta, deal_1), reject(\alpha, \beta, deal_1)$
$withdraw(\alpha, \beta), [t_{max}]$
S2
$withdraw(\alpha, \beta), [t_{max}]$

Figure 4: Communication Protocol for L_3

During a negotiation process, a lot of deals can be signed; that is, a lot of proposals can be finally accepted. If it is desired, as in the protocol in Figure 4, the protocol can specify a t_{max} used to automatically terminate the negotiation process when no message is sent during such specific amount of time. In this particular case, the negotiation terminates when one of the agents decides to withdraw. Note also, that an agent can also reject a non accepted deal previously proposed by himself in order to retract that proposal.

8 Testbed Architecture

The testbed is designed as a set of software components that allow to check and analyze the correct behaviour of bots playing Diplomacy. At the same time, the testbed includes a framework that makes it easy to develop new bots, as the potential actions the bot may choose are calculated by the framework. It also includes a library that allows the bot to talk with other players. With this testbed, the researcher can build a bot and get relieved from the painful programming communication between agents and concentrate on the interesting decisions from MAS research perspective: what to do and whom to say what, that is, *reasoning*. Next, we describe the three components of the testbed and how it can be used.

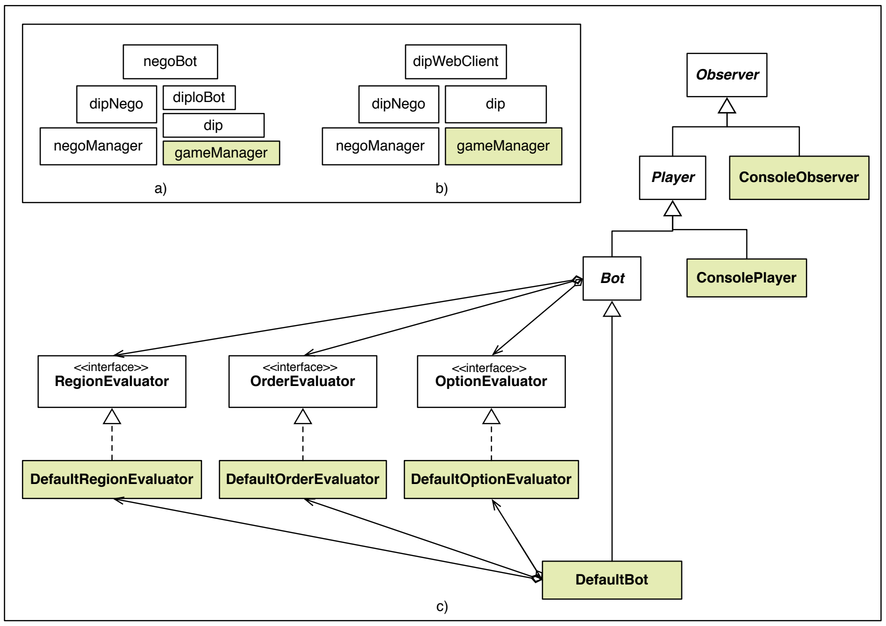


Figure 5: a) *negoBot* structure, b) *dipWebClient* structure and c) UML 2 class diagram of the bot development framework

8.1 dip

dip is a framework for bot development written in java that incorporates a simple bot architecture for playing Diplomacy. A bot developer has to just build the mental state for its agent and program the decision making process that makes this bot to be able to play Diplomacy with or without negotiation.

The bot developer can choose one out of three different levels of abstraction that *dip* provides. As it can be seen in Figure 5 c), *dip* provides a hierarchy of abstract classes: *Observer*, *Player* and *Bot*. The *Observer* class only observes a game. It gets information about the current state of the game and about the orders that the players write in each phase. Contrarily, *Player* is not only an observer. As its name indicates, *Player* plays Diplomacy getting from the *Observer* all the necessary information and written orders.

As examples of clients and players, *dip* provides the classes *ConsoleObserver* and *ConsolePlayer* that allow a human player to observe a game reading the messages that will appear in an alphanumeric console[9] (*ConsoleObserver*) and, then, to play the game by writing the orders that he/she chooses (*ConsolePlayer*).

In addition, *dip* provides *Bot* that is the more simple way to implement a bot. *Bot* deals with the search of the best option, that is the best order combination, using the implementations of *RegionEvaluator*, *OrderEvaluator* and *OptionEvaluator* that the bot developper has to specify. In this way, we reduce the complexity of bot development to just a simple specification of the preferences of the programmer among different regions, orders and options at any time.[10]

In summary, a bot developper can choose between using *Bot* or implement from scratch the orders selection mechanism. In both cases, *dip* will deal with the management of the game state and the orders, as well as with the communication with the game manager.

8.2 negoBot

negoBot is a bot capable of playing Diplomacy using negotiation[11] developed over *defaultBot*, a bot that uses *Bot* and provides an implementation of the three interfaces that *Bot* needs to work properly. Therefore, *negoBot* does not even have to worry about the game strategy because *defaultBot* already does it. Then, the functionality that *negoBot* provides is a model of the agents (players) and their relationships (enemies, allieds, friends) that drives the decisions that *negoBot* has to take: which deal should it propose? who should it propose the deal to? when? should it accept the deal that someone is currently proposing to it? should it respect the deal? *negoBot* has always the last word when choosing the set of orders to write. *defaultBot* provides a ranking of them and *negoBot* can choose, for instance, the best one respecting the deals that it has signed.

The Figure 5 a) represents the module dependency of *negoBot*. All the modules represented in that figure have been already developed by us except the game manager that is currently under development. But this

[9]Console, Shell, Bash terminal

[10]The evaluators can be parametrized to evaluatre differently according to the current game state.

[11]Almost all bots currently developed play the no-press version of Diplomacy. That is, the game without negotiation. *negoBot* allows to negotiate using level 1 of our language definition.

is not a shortcoming as *dip* implements the standard Diplomacy protocol and can communicate with different existing servers such as: *AiServer* or *Parlance*. In the figure there are two modules not yet mentioned: *dipNego* and *negoManager*. *dipNego* implements the language described in Section 6 and *negoManager* allows the players to communicate using the language level decided by them at the begining of the game.

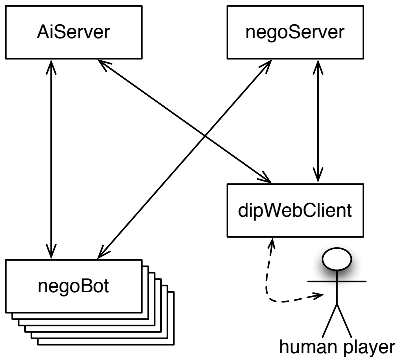
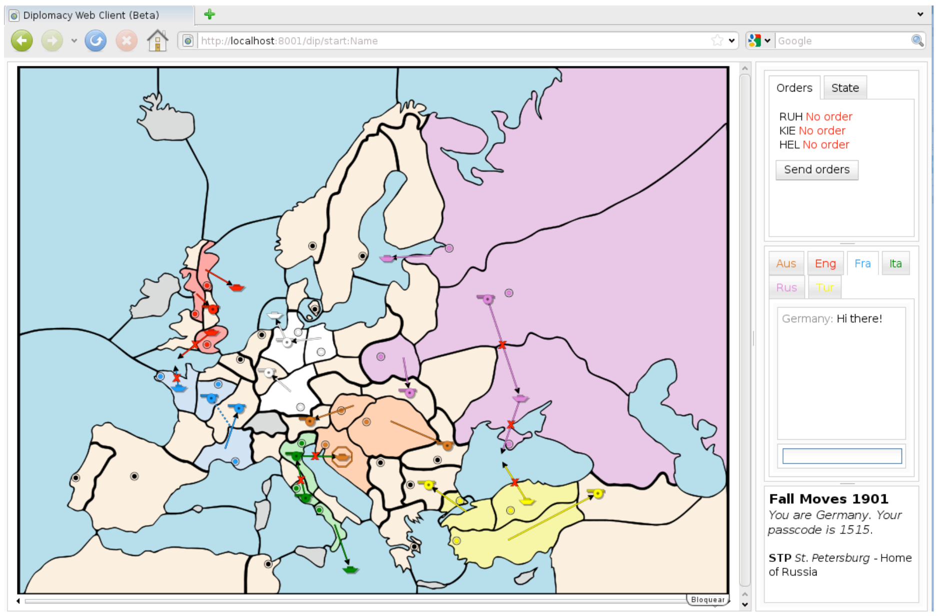


Figure 6: Execution example of a human playing Diplomacy with the *dipWebClient* against six *negoBot*s

8.3 dipWebClient

With *dip* and *negobot* we can run a Diplomacy game between seven *negoBot*s using *negoManager* and, for example, *AiServer* managing the game. To be able to observe how our bots play we can connect to either an observer provided by *dip*, *consoleBot*, or to a graphical friendly web interface called *dipWebClient*. In Figure 6 we can see how the different components would be connected to allow a human to observe a game. *dipWebClient* can be set as observer, using directly the *Observer* class of *dip*, or as player, using the *Player* class. In both cases, it allows a human agent to follow a game just observing how the units move from one region to another in the map. Besides, if we set *dipWebClient* as a player, we will be able to point out which orders do we want to write using the mouse on the map. We can also talk with other players, humans or bots capable of negotiating, by means of a chat interface that is visible just next the map of the game. In Figure 7 a screenshot of a Diplomacy game is visualised using *dipWebClient*. Currently, there are some other graphical interfaces to play Diplomacy but no one as clear and useful as this one. *dipWebClient* provides a sliding bar that allows to visualise on the map all the history of movements of a game. You can check past moves just moving the slider right and left.


Diplomacy Web Client (Beta)
http://localhost:8001/dip/start:Name
Google
Orders
State
Send orders
Aus
Eng
Fra
Ita
Rus
Tur
Bloquear

RUH No order
KIE No order
HEL No order

Germany: Hi there!

Fall Moves 1901
You are Germany. Your passcode is 1515.

STP *St. Petersburg* - Home of Russia

Figure 7: Screenshot of *dipWebClient*

8.4 Results

All the components of the testbed have been tested and used for playing games with and without negotiation. The reuse of code —remember that all components depend on dip, makes it easier to test everything and reduces the number of errors that can appear in the code. Currently, all the components of the testbed presented in this paper work properly.

One of the goals of *dip* has been the creation of a repository of small code modules useful for bot development. In fact, one of our goals with the implementation of *defaultBot* and *negoBot* was to check the facility of people not directly involved in the *dip* framework development to focus on what it is really interesting for them: the strategy of the game and the negotiation. The software structure of *dip* makes the reusability of code easier in the development of bots. Thus, the building of a new bot becomes just the reuse of a set of modules plus the development of a few new ones. For instance, in the case of *negoBot*, the strategic modules that allow region, order and option evaluation provided by *defaultBot* have been combined with the *agent* and the *agent relationship* modules and with a decision taking module developed from scratch. Other components are currently under development and will be briefly discussed in Section 10.

We have already described how a human player can use *dipWebClient* and how a developer can observe his/her bots using this interface. But *dipWebClient* can also be used to record what a human expert player does in a specific situation. This data logging can be used to implement data mining or machine learning technics.

This server allows also to write free text when so is agreed by the players. This communication server allows a researcher to monitor his/her bots capability of keeping a communication dialogue. A simulator used to test bots is also provided: it allows the researcher to talk to his/her bot and observe its reactions.

9 Related Work

Almost all the bots that are currently implemented follow a standard on communication defined by DAIDE. This is the use of a client-server architecture where the server is the game manager and clients can be either observers or players. Clients cannot communicate directly, the communication is always between client and server. The server may decide to forward a message to another client. The communication is done via TCP/IP following the protocol defined in [12]. The syntax of the messages that client and server can exchange follow the level 0 language syntax. These two parts, the communication protocol and the syntax of language level 0, are shared by almost everyone who is developing bots. But the rest of language levels is not. In fact, we claim that a new language definition for communication between players is needed and we have proposed a new one that separates the domain dependent terms from the language itself and divide it in several levels of complexity. This new language is defined from the point of view of multiagent system researchers and allows to test the work in a lot of research topics like, for example: negotiation, trust, reputation and argumentation. We are in the process of outreach the community and try of persuade developers to adopt the language tower.

10 Discussion

Diplomacy is an ideal environment for testing MAS because players must constantly confer, sign agreements and decide whether to honour them, decide who to cooperate with, ... There is also a large group of human players ready to go head to head with software agents and accustomed to both playing online and dealing in a restricted language. There are also a lot of developers wanting to provide bots of Diplomacy.[12] Finally, the game has no random elements that could decrease the relevance of the experimental results. Therefore, the results obtained by the use of this testbed would be significative.

The testbed is already available online at `URLblinded` Next work will be focussed mainly on developing good negotiator agents capable of playing against humans in the different language levels of L. In fact, this testbed was needed as a step before being able to test our own work on negotiation. However, we also wanted to share it with everyone interested in negotiation or argumentation.

In parallel with the negotiating bot development, we are working to provide the following new tools to the testbed:

[12]DAIDE community has over 200 members.

- a desktop based graphical interface similar to the one we provide for playing online.
- a tool for translating from a restricted natural language to the testbed language. It will assist human players when negotiating with agents.
- a web application that manages the creation of new games and provides a rating of the players and some tools for social networking.
- a repository of the source code of modules that provides several capabilities needed for a full Diplomacy negotiator bot, such as the functionality that *defaultBot* and *negoBot* already provide.
- a new functionality for the web application that will allow other researchers to launch a bot connected to our game manager automatically.
- a game manager with its adjudicator. They are almost finished and implemented in java using *dip*. Remember that we are currently using other game managers that follow the standard communication protocol and language.

By means of this testbed we provide to the MAS community an environment where very expressive illocutions can be exchanged between a small set of agents, usually 7. Agents compete with the aim of getting more power. But power is obtained by cooperation. To be able to increase your power, you have to convince others to help you. Thus, agents repeatedly negotiate to convince others to accept a plan of action where both cooperate and split the benefit. Being friendly, persuasive, loyal and trustworthy are some of the skills that can be tested with this testbed.

References

[1] Diplomacy ai development environment site. `http://www.daide.org.uk/`.

[2] The diplomatic pounch. `http://www.diplom.org`.

[3] jdip, a diplomacy mapper and adjudicator, 2004. `http://jdip.sourceforge.net/`.

[4] T. Alsinet, C. Chesñevar, L. Godo, and G. Simari. A logic programming framework for possibilistic argumentation: Formalization and logical properties. *Fuzzy Sets and Systems*, 159(10):1208–1228, 2008.

[5] L. Amgoud and H. Prade. Using arguments for making and explaining decisions. *Artif. Intell.*, 173(3-4):413–436, 2009.

[6] J. Debenham and C. Sierra. Trust and honour in information-based agency. pages 1225–1232, 2006.

[7] S. Kraus. Designing and building a negotiating automated agent. *Computational Intelligence*, 11:132–171, 1995.

[8] S. Kraus. Negotiation and cooperation in multi-agent environments. *Artificial Intelligence*, 94:79–97, 1997.

[9] S. Kraus, D. Lehmann, and E. Ephrati. An automated diplomacy player. In D. Levy and D. Beal, editors, *Heuristic Programming in Artificial Intelligence: The 1st Computer Olympia*, pages 134–153. Ellis Horwood Limited, 1989.

[10] R. Lin, S. Kraus, J. Wilkenfeld, and J. Barry. Negotiating with bounded rational agents in environments with incomplete information using an automated agent. *Artificial Intelligence*, 172(6-7):823 – 851, 2008.

[11] D. Norman. Diplomacy ai development environment message syntax, 2006. `http://www.ellought.demon.co.uk/dipai/dpp_syntax.rtf`.

[12] A. Rose. The diplomacy centralisation project client-server protocol, 2003. `http://www.daide.org.uk/external/comproto.html`.

[13] J. Shaheed. Creating a diplomat. Master's thesis, Department of Computing, Imperial College Of Science, Technology and Medicine, 180 Queen's Gate, London, SW7 2BZ, UK, June 2004.

[14] C. Sierra and J. Debenham. Information-based reputation. In *First International Conference on Reputation: Theory and Technology*, 2009.

[15] A. Webb, J. Chin, T. Wilkins, J. Payce, and V. Dedoyard. Automated negotiation in the game of diplomacy. Master's thesis, January 2008.