Book Title
Book Editors
IOS Press, 2003

Building Agents that Plan and Argue in a Social Context

Dionysis Kalofonos [a], Nishan Karunatillake [b], Nicholas R. Jennings [b], Timothy J. Norman [a], Chris Reed [c] and Simon Wells [c]
[a] *Department of Computing Science, University of Aberdeen*
[b] *School of Electronics and Computer Science, University of Southampton*
[c] *Division of Applied Computing, University of Dundee*

Abstract. In order for one agent to meet its goals, it will often need to influence another to act on its behalf, particularly in a society in which agents have heterogenous sets of abilities. To effect such influence, it is necessary to consider both the social context and the dialogical context in which influence is exerted, typically through utterance. Both of these facets, the social and the dialogical, are affected by, and in turn affect, the plan that the influencing agent maintains, and the plans that the influenced agents may be constructing. The i-Xchange project seeks to bring together three closely related areas of research: in distributed planning, in agent-based social reasoning, and in inter-agent argumentation, in order to solve some of the problems of exerting influence using socially-aware argument.

Keywords. Multiagent Planning, Argument Protocols, Social Reasoning, Negotiation.

1. Introduction

Negotiation is a key form of interaction in multi-agent systems. It is important because conflict is endemic in such systems and because the individual agents are autonomous problem solving entities that are typically concerned with achieving their own aims and objectives. Given its importance, such negotiations come in many different shapes and forms, ranging from auctions to bilateral negotiations to argumentation. Here we focus on this latter kind of interaction because it offers perhaps the greatest degree of flexibility out of all these many different types. However, this flexibility comes at a price. Specifically, conceptualizing, designing, and building agents that are capable of argumentation-based negotiation is a major challenge. Given this fact, most work in this area is primarily directed at the theory of such agents and those implementations that do exist are somewhat primitive in nature. Moreover, much of the theoretical work in this area tends to concentrate on a specific aspect of the negotiation and fails to provide a coherent overarching framework. Against this background, we describe our work in the Information Exchange Project (*i*-Xchange) that seeks to rectify these shortcomings.

In more detail, this work seeks to integrate and pull together the following key components of an agent's activity as it relates to argumentation-based negotiation:

- The ability of an agent to devise a plan of action that takes account of the fact that the agent is situated within a multi-agent community. Thus such an agent can devise a plan that involves steps that will be performed by agents other than itself.

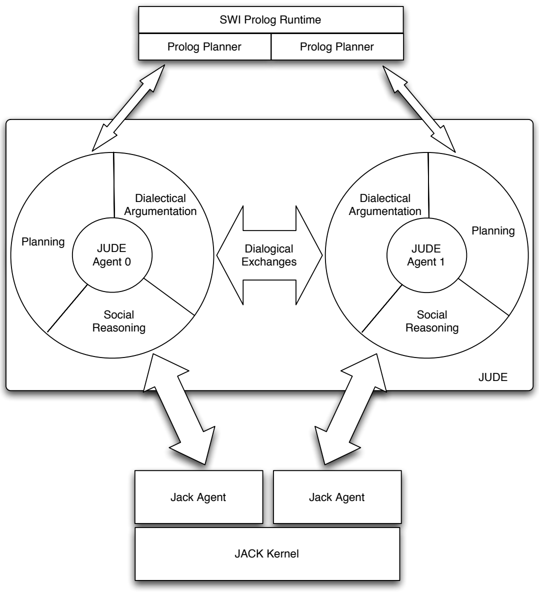

Figure 1. The structural components of the *i*-Xchange architecture

- For those actions that need to be performed by other agents, social reasoning is required to determine what agents should be chosen to perform what actions. This involves determining which agents are most suited for the task, which are likely to be available, and which are likely to be the most reliable.
- Once the appropriate agents have been identified, the agent needs to determine the most effective way of persuading these agents to perform the desired action. This dialogical goal can be achieved through a variety of means such as by offering rewards, making threats, or making use of social relationships that exist between the agents.

By bringing together these key building blocks, we are able to produce a coherent framework and software architecture for an agent that can perform a complete planning-acting cycle in which argumentation is used as the basis for all social inter-changes. As well as detailing the various components and their interfaces, we also demonstrate their operation in an e-Science scenario that has motivated much of this work.

2. The Information Exchange

The *i*-Xchange uses two multiagent system frameworks; JACK and JUDE (the Jackdaw University Development Environment. JUDE is a lightweight, flexible, industrial-strength agent platform that uses a modular approach to agent development. This enables domain specific functionality to be encapsulated into a module which can be dynamically loaded into an agent at runtime. Individual agents within the *i*-Xchange are represented by JUDE agents composed of a number of modules. A proxy module incorporating a communications bridge allows a 1:1 relationship with JACK agents. The reason for the use and integration of multiple disparate frameworks is twofold, firstly it allows existing domain specific software to be used without reimplementation, and secondly it demonstrates that agents can be developed under different extant frameworks and integrated into a single heterogeneous MAS.

An *i*-Xchange agent is composed of three modules offering domain specific functionality. These are the planning, social-reasoning, and dialogical argumentation modules which are discussed through the remainder of this section. Figure 2 gives an overview of the components and modules that comprise the infrastructure for an *i*-Xchange MAS. Modules communicate with each other to provide aggregate behaviours. Inter-module communication is achieved by passing data objects between modules. Two such objects are the *service request* and the *proposal*. A service request is created by the Planning module to encapsulate a partial plan consisting of a set of actions and the name of an agent committed to perform the actions (initially set to $\perp$). A proposal is created by the dialogical argumentation module during a dialogue to encapsulate a service request received from another agent and any associated social issues. Figure 2 shows a complete circuit of communication for a simple enquiry dialogue between the six modules incorporated in a pair of *i*-Xchange agents.

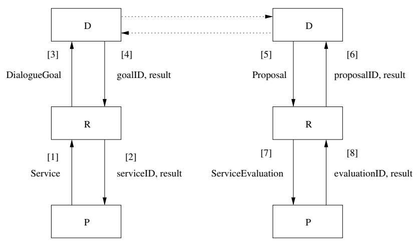

Figure 2. Basic inter-module and inter-agent communication pathways. The modules P, R, and D on the left constitute the initiating agent, iXchangeagent$_0$, and the modules P, R, and D on the right constitute the recipient agent, iXchangeagent$_1$.

2.1. Planning

The planning module makes use of the Graphplan algorithm first introduced in [1] which we have implemented in Prolog. Let us discuss the algorithm in detail and present our extensions for the extraction of services from the constructed plans and for service evaluation through the merging of services into the constructed plans.

A Planning graph is a layered graph with each layer consisting of a set of propositions and a set of actions (see figure 3). Each layer (named time-step) represents a point in time, hence a set of propositions at a time-step n represents a snapshot of the state of the world at the time-step n, while a set of actions appearing at a time-step n contains all the actions that are executable in the state of the world at time-step n.

The graph consists of two kinds of nodes namely, the proposition nodes which form the proposition sets at each time-step and the action nodes which are the instantiations

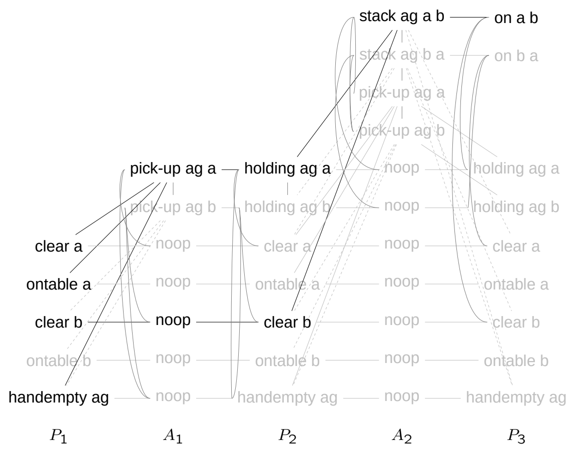

Figure 3. A planning graph generated for a very simple blocks world problem (some nodes and edges are omitted for clarity).

of the operators on the propositions of the time-step. The nodes within the graph are connected with three kinds of edges:

Precondition edges: The precondition edges connect the actions nodes of a time-step n with their preconditions appearing in time-step n.

Delete edges: The delete edges connect the action nodes of a time-step n with their negative effects appearing in time-step $n + 1$.

Add edges: The add edges connect the action nodes of a time-step n with their positive effects appearing in time-step $n + 1$.

For each action node placed into the graph a number of edges are generated that connect the action nodes with their preconditions. Those edges are the precondition edges (see figure 3:P_1 – A_1). Having placed the action nodes into the graph, Graphplan collects their effects which are placed as propositions into the subsequent time-step (see Figure 3:P_2). Subsequently, Graphplan connects the action nodes with their positive effects through a set of add edges (see Figure 3:A_1 – P_2), and with their negative effects through a set of delete edges.

Since the actions of the same time-step are applied on the same state with a finite set of resources there is a potential that the actions will interfere during the consumption of non sharable resources. In order to capture the conflicts that may arise through the parallel execution of actions, Graphplan propagates binary constraints among the actions that appear in the same time-step. The propagated constraints are named mutual exclusion relations (mutex for short) and their meaning is that only one of the two action nodes marked as mutually exclusive can be performed at that time-step. The notion of exclusivity extends to the propositions meaning that two propositions marked as mutually exclusive cannot coexist at the same time-step.

At the completion of the generation of a time-step Graphplan tries to identify the goals and ensure that the goals are mutex free at the newly generated time-step. If the goals are present and mutex free, Graphplan performs a search within the generated graph in order to extract a plan. The search starts from the goals going backwards (i.e. towards the initial state). Moreover, the search is layered meaning that a new layer of the graph is considered only when the current goal set is fully satisfied with the actions that appear in the current layer. To make this more concrete let us have a look at our example of planning graph shown in figure 3. Graphplan starts from 3:P_3 where it tries to identify the goals. Having found the goals Graphplan tries to support all the goals with a mutex free set of actions from the previous time-step 3:P_2. If that is not possible Graphplan backtracks. If a set is found Graphplan generates a new goal set containing the preconditions of the selected actions and repeats the process with the new goal set. The search succeeds when Graphplan reaches the initial state. Upon the successful completion of the search the extracted plan is returned (the extracted plan of our example is highlighted in figure 3).

Having constructed a plan, the planning module fragments the plan into a set of services. A service consists of a unique identifier of the service, and the name of the agent that has committed to the execution of the service (initially set to $\perp$). More importantly, a service contains a set of actions which are causally related on the actor who needs to perform them. For instance the highlighted plan of figure 3 consists of a service containing the actions ‘pick-up ag a’ – ‘stack ag a b’ as the same actor who is going to pick up the block needs also to stack it. An action consists of a literal representation of its instance (in the Prolog formalism), and the point in time that it needs to be executed, hence the precedence constraints among the actions of a service are explicitly captured within the service.

Having received a service the planning module extracts the actions of the service and tries to incorporate the actions into the plan in the time-steps defined in the service. If that is achievable then the service is integrated successfully with the plan and the planning component returns a positive reply (i.e. $\top$), negative otherwise (i.e. $\perp$).

2.2. *Social Reasoning*

An agent planning and acting in a solitary manner has capabilities that are limited to its own. Thus, the goals it can achieve by functioning as an individual entity are rather constrained. The real potential of agents arises when these solitary entities begin to act as communities. In such a context, opportunities exist for individual agents to compensate for each other’s deficiencies by acting collectively, thereby achieving higher overall performance as a system. A key mechanism for coordinating social interaction between agents is *negotiation* [2].

In abstract, negotiation is commonly viewed as a dialectic process that allows two or more parties to interact and resolve *conflicts of interest* that they have among each other with respect some issues of mutual interest [3,4]. For example, in a situation where a buyer agent attempts to purchase a car from a seller agent, there is a clear conflict of interest between the two parties with respect to the price of the car. The buyer is interested in paying the lowest price possible, whereas the seller is interested in gaining the highest price possible (thus, the conflict of interest). Negotiation provides a means for the two agents to resolve their conflict of interest by allowing them to come to a mutually acceptable agreement. Thus, it can be observed that the ultimate goal of the negotiation is to arrive at a mutual agreement and, thereby, resolve the conflict of interest present among the different parties.

Negotiation is so central and fundamental because it provides the agents with the means of influencing the behaviour of their *autonomous* counterparts. By definition, an autonomous entity cannot be forced to adopt a certain pattern of behaviour. Thus, negotiation provides agents with the means to convince their autonomous counterparts by forwarding proposals, making concession, trading options, and, by so doing, (hopefully) arriving at a mutually acceptable agreement [5]. Apart from being used as a means to achieve agreements, negotiation also underpins agents' efforts to coordinate their activities, achieve cooperation, and resolve conflicts in both cooperative [6] and self-interested [7] domains.

Increasingly it is argued that incorporating the ability to exchange arguments within such a negotiation interaction mechanism not only increases the agent's ability to reach agreements, but also enhances the mutual acceptability of the agreement [8,9,10,11]. In more detail, when agents interact within a multi-agent society, in most cases they do so with imperfect knowledge about their own capabilities, preferences, and constraints and those of their counterparts. When agents interact within such knowledge imperfections they may lead to another form of conflict between the agents, termed *conflicts of opinion*, which may hinder the the agents ability to reach agreements or lead them to sub-optimal agreements with their counterparts. “Argumentation-Based Negotiation" allows the agents to exchange additional meta-information such as justifications, critics, and other forms of persuasive locutions within their interactions. These, in turn, allow agents to gain a wider understanding of each others capabilities, preferences, and constraints, thereby making it easier to resolve such conflicts that may arise due to incomplete knowledge. In the current implementation the social reasoning component considers three important decisions; namely (i) what agent to argue with, (ii) what issues to argue on, and (iii) within which ranges. The following considers these in more detail:

- **What agent** In considering which counterpart to interact with the social reasoning component considers two important aspects; first the structure of the society and secondly the experience that it has gained in its past encounters. In most instances, an agent society usually embodies a structure. Certain agents may act in certain roles within the agent society which may lead to relationships with other agents acting certain other roles. As a result of these roles and relationships agents may obtain specific obligations to others and may gain rights to influence certain others. These obligations and rights constitute social influences which can be constructively exploited in a society. In such a context, when considering the counterpart to interact with agent could constructively exploit these social influences. For instance, when negotiating for a certain service an agent may choose another which already is obliged to provide this capability through the social structure. Instead of randomly picking any agent in the society, using such a heuristic in selecting its counterpart may make the interacting more efficient. Apart from the social influences of the structure of the society, agents may also take into account the experience gained in its past encounters. In more detail, if the agent has interacted with that agent in the past to obtain a certain service, it may select the same agent when it requires the same type of service again. This may enhance the interaction being more effective since the agent already knows that it has the capability to perform the required service, which may not always be the case in selecting a random agent from the society.

- **What Issues** Once an agent has chosen its counterpart, the next main decision that it needs to make are the set of issues to negotiate with. In the negotiation literature, this set of issues is generally referred to as the negotiation object [5]. For example, when two agents are negotiating the sale of a car they will address a number of parameters such as price, warranty period, and after sale service. Each of these will be a certain negotiation issue, whereas all of these issues taken together will form the negotiation object. One of the advantages of using an argumentation-based negotiation approach is that new issues can be introduced or the existing once be retracted from the negotiation object during the argumentative encounter.
- **What ranges** Finally, the agent would need to decide the ranges (i.e., the upper and the lower limits) for each particular negotiation issue that it should adhere to during its negotiation encounter. The agent may have a certain objective for each particular issue. For instance, the buyer agent may desire to minimise the price paid while trying to maximise the quality or after-sales service parameters. The seller on the other hand may wish to maximise the price while attempting to minimise the after-sales service parameter. Thus, the upper and the lower limits as well as the direction (maximise or minimise) would depend the agents individual desire in the negotiation interaction. A rational agent wouldn't make an offer that costs the agent more than the expected benefit it aims to receive in return. Thus, the upper limit of the all the negotiation issues would have a cost less than the exacted benefit of buying that service. On the other hand, an agent would not make an offer with a negative reward since it will be irrational for another agent to accept such an offer. Between these upper and the lower bounds an agent can derive a series of offers with a combination of issue value tuples that are viable (cost is less than the benefit). This will give the set of proposals that the dialogue module can use in its encounter.

These three components (the agent, the issues, and their respective ranges) compose the dialogue object which is passed to the dialogical argumentation module. In the next section we discuss how an agent argues with the chosen counterpart to gain the service required.

2.3. Dialectical Argumentation

Agents in the *i*-Xchange system possess a number of capabilities, for example, they are able to construct plans for achieving goals (detailed in section 2.1), and they are able to reason in a social context about how to get other agents in the MAS to perform the actions required by the plan (as detailed in section 2.2). Once a plan has been constructed and a strategy for achieving the plan is devised it is necessary to interact with the other agents to engage them in performing the tasks required to satisfy the plan. In the *i*-Xchange this is achieved through the use of argumentative dialogue which is controlled by the *Dialectical Argumentation* (DA) module.

Dialogue games have been proposed as a means to model the interactions between participants during argumentative dialogues. One branch of dialogue game research is into formal dialectical games [12]. These are two-player, turn-taking games in which the moves available to the players represent the locutional acts or utterances made by the participants of a dialogue. In other words a formal dialectical system expresses a protocol for communication between conversing agents by regulating what an agent can say and when they can say it.

Many dialectical games have been proposed based on the characterisations of a range of dialogical situations, for example, Hamblin's "simple dialectical system" [12] and Mackenzie's DC [13] are targeted towards fallacy research whilst McBurney and Parsons specify some games for use in communication between agents in MAS [14]. The formal dialectic systems used in the *i*-Xchange are represented using the unified specification format introduced in [15]. This representation is part of a unified framework for representing, rapidly implementing and deploying formal dialectic systems called the Architecture for Argumentation (A4A). To facilitate this, the framework incorporates a range of general machinery for representing dialogues and dialectical games. Each dialectical game is itself designed to model the interactions between participants in a particular dialogical situation.

An example of the most basic dialogical interaction between *i*-Xchange agents is illustrated in the sample system output in section 3 where a partial plan has been constructed by the planning module and the social-reasoning module has selected an agent to carry out the partial plan. The DA module initiates a dialogue with the nominated agent to determine whether the other agent is capable of performing the actions required of the partial plan. The dialogue uses a small range of moves to achieve this which are detailed as follows using the A4A schema:

Game

Name iXchange$_0$
Turns ⟨ Liberal, Single ⟩
Participants = {init, resp}
Stores: ⟨CStore, Init, Mixed, Set, Public⟩
⟨CStore, Resp, Mixed, Set, Public⟩

Structure

Initiation
Requirements:
$T_{current}$ = 0
Effects:
$T^{init}_{next\_move}$ = ⟨Initiate, (–)⟩

Termination
Requirements:
T_{last_move} = ⟨Affirm, (–)⟩ ∨
T_{last_move} = ⟨Deny, (–)⟩
Effects:
Dialogue$_{status}$ = complete

Moves

⟨Initiate, (S)⟩
Requirements:
$T_{Current}$ = 1
Effects:
$T^{listener}_{next\_move}$ = ⟨ Acknowledge, (–)⟩

⟨Acknowledge, (S)⟩
Requirements:
$T^{listener}_{last\_move}$ = ⟨ Initiate, (-)⟩
Effects:
$T^{listener}_{next\_move}$ = ⟨ Enquire, (PP)⟩

⟨Enquire, (S)⟩
Requirements:
$T^{listener}_{last\_move}$ = ⟨ Acknowledge, (–) ⟩
Effects:
CStore$^{speaker}_{current}$ + PP ∧ CStore$^{speaker}_{current}$ + PP ∧ (T_{next_move} = ⟨Affirm, (PP)⟩ ∨

T_{next_move} = $\langle$Deny, (PP)$\rangle$)

$\langle$**Affirm, (S)**$\rangle$
Requirements:
$T^{listener}_{last\_move}$ = $\langle$ Enquire, (PP) $\rangle$ **Effects:**
$CStore^{speaker}_{current}$ + PP

$\langle$**Deny, (S)**$\rangle$
Requirements:
$T^{listener}_{last\_move}$ = $\langle$ Enquire, (PP) $\rangle$
Effects:
$CStore^{speaker}_{current}$ + $\neg$PP $\wedge$
$CStore^{speaker}_{current}$ – PP

The iXchange$_0$ protocol is split into three parts; game, structure and moves. The game part specifies the turn structure, participants, and commitment stores. The structural part specifies the required state for legal initiation of a dialogue and the states under which the dialogue will terminate. The moves part specifies the moves which players can make during a dialogue. Moves are presented in terms of their legality requirements and resultant effects if the move is legally played. Legality requirements are formulated in terms of earlier moves during the dialogue. Effects are formulated in terms of legal responses and commitment store updates.

When all three *i*-Xchange agent modules are integrated in a single agent, the game, iXchange$_0$, is sufficient to enable an agent to engage in a simple dialogue with another and determine whether the other agent can execute a partial plan thus enabling a communication round trip between two agents composed of the basic *i*-Xchange agent modules.

3. Example

This section illustrates the use of the *i*-Xchange MAS when applied to an *e*-Science scenario. The *e*-Science domain consists of a network of host machines, a set of datasets that need to be processed, a set of data transportation mediums that can transfer the datasets between hosts and a set of data processing systems that are needed to be available in the host machine so that the dataset can be processed. The following operators are supported;

move dataset: Moves a data transportation medium loaded with a data set between hosts.

move data processing system: Moves a data transportation medium loaded with a data processing system between hosts.

move data transportation medium: Moves a data transportation between hosts.

load dataset: Loads a dataset to a data transportation medium.

load data processing system: Loads a data processing system to a data transportation medium.

unload dataset: Unloads a data set from a data transportation medium.

unload data processing system: Unloads a data processing system from a data transportation medium.

execute: Executes a data processing system at a given host.

terminate: Terminates the execution of a data processing system.

process: Processes a data set.

The *e*-Science domain has the following properties; Three classes of agents appear in the domain, namely the data processing systems, the data transportation mediums, and the host machines. The *e*-Science domain is a mixture of the transportation class of planning domains (logistics for instance) and the puzzle class of planning domains (for example blocks-world). The first property allows us to model complex societies structured as a hierarchical network of social influences. Hence the agents are provided with a rich social model to exploit the capabilities of their social reasoning component. The second property allows the agents to fully exploit their planning capabilities as the *e*-Science domain allows for the specification of very complex planning problems. The combination of the two properties provides a scenario where complex agent interactions can emerge for conflict resolution and task delegation, hence the agents can exploit their capabilities on dialectical argumentation. The following fragment illustrates system output as the planner generates a partial plan, the social reasoning module nominates an agent to execute the plan and the dialectical argumentation module engages that agent in dialogue.

```
Planner: {Agent=p1, Action=load_dps(t2, p1, m3), Time=1}
Planner: {Agent=p1, Action=move_dps(t2, p1, m3, m2), Time=2}
Planner: {Agent=p1, Action=unload_dps(t2, p1, m2), Time=3}
Planner: {Agent=p1, Action=execute(p1, m2), Time=4}
Planner: {Agent=p1, Action=process(d1, p1, m2), Time=6}
Planner: {Agent=p2, Action=execute(p2, m2), Time=1}
Planner: {Agent=p2, Action=process(d2, p2, m2), Time=2}
Planner: {Agent=p2, Action=terminate(p2, m2), Time=3}
Planner: {Agent=t1, Action=move_dtm(t1, m3, m2), Time=1}
Planner: {Agent=t1, Action=move_dtm(t1, m2, m1), Time=2}
Planner: {Agent=t1, Action=load_ds(t1, d1, m1), Time=3}
Planner: {Agent=t1, Action=move_ds(t1, d1, m1, m2), Time=4}
Planner: {Agent=t1, Action=unload_ds(t1, d1, m2), Time=5}
Planner: {Agent=t2, Action=load_dps(t2, p1, m3), Time=1}
Planner: {Agent=t2, Action=move_dps(t2, p1, m3, m2), Time=2}
Planner: {Agent=t2, Action=unload_dps(t2, p1, m2), Time=3}
ServiceImpl: ID: 77158a:10abd317fb0:-7fcf
|-> name: load_dps(t2, p1, m3) – time: 1
|-> name: move_dps(t2, p1, m3, m2) – time: 2
|-> name: unload_dps(t2, p1, m2) – time: 3
|-> name: execute(p1, m2) – time: 4
|-> name: process(d1, p1, m2) – time: 6
ServiceImpl: ID: 77158a:10abd317fb0:-7fce
|-> name: execute(p2, m2) – time: 1
|-> name: process(d2, p2, m2) – time: 2
|-> name: terminate(p2, m2) – time: 3
ServiceImpl: ID: 77158a:10abd317fb0:-7fcd
|-> name: move_dtm(t1, m3, m2) – time: 1
|-> name: move_dtm(t1, m2, m1) – time: 2
|-> name: load_ds(t1, d1, m1) – time: 3
|-> name: move_ds(t1, d1, m1, m2) – time: 4
|-> name: unload_ds(t1, d1, m2) – time: 5
ServiceImpl: ID: 77158a:10abd317fb0:-7fcc
|-> name: load_dps(t2, p1, m3) – time: 1
|-> name: move_dps(t2, p1, m3, m2) – time: 2
|-> name: unload_dps(t2, p1, m2) – time: 3
Reasoning: addServiceRequest
Reasoning: received ServiceRequest with ID:77158a:10abd317fb0:-
7fcf and actions:
|-> name: load_dps(t2, p1, m3) – time: 1
|-> name: move_dps(t2, p1, m3, m2) – time: 2
|-> name: unload_dps(t2, p1, m2) – time: 3
|-> name: execute(p1, m2) – time: 4
|-> name: process(d1, p1, m2) – time: 6
iXchangeAgent0::Reasoning: invokeAddDialogueGoal
iXchangeAgent0::DialogueModule Dialogue Goal Added
iXchangeAgent0::DialogueModule sending message to: iX-
changeAgent1 with message content:INIT DIALOGUE
iXchangeAgent1::DialogueModule handling message from: iX-
changeAgent0 with message content:INIT DIALOGUE
iXchangeAgent1::DialogueModule sending message to: iX-
changeAgent0 with message content:OK
iXchangeAgent1::DialogueModule calling invokeAddProposal in
Social Reasoning Module
iXchangeAgent1::ReasoningModule: addProposal()
iXchangeAgent1::ReasoningModule: invokeAddServiceEvaluation
Loading scheduler.pl... ok – PlanningModule: iXchangeAgent1:
Schedule for move_dtm(t1, m3, m2) at 1 – result: yes
iXchangeAgent1::PlanningModule::Schedule for move_dtm(t1,
m2, m1) at 2 – result: yes
iXchangeAgent1::PlanningModule::Schedule for load_ds(t1, d1,
m1) at 3 – result: yes
iXchangeAgent1::PlanningModule::Schedule for move_ds(t1, d1,
m1, m2) at 4 – result: yes
iXchangeAgent1::PlanningModule::Schedule for unload_ds(t1, d1,
m2) at 5 – result: yes
iXchangeAgent1::Reasoning: invokeAddProposalResponse
iXchangeAgent1::DialogueModule Response to proposal:
77158a:10abd317fb0:-7fcd iXchangeAgent1
iXchangeAgent1::DialogueModule sending message to: iX-
changeAgent0 with message content:ACCEPT: iXchangeAgent1
iXchangeAgent0::DialogueModule handling message from: iX-
changeAgent1 with message content:ACCEPT: iXchangeAgent1
iXchangeAgent0::DialogueModule calling invokeAddDialogueRe-
sult in Social Reasoning Module iXchangeAgent0::Reasoning: ad-
dDialogueResult()
iXchangeAgent0::Reasoning: invokeAddServiceResponse
iXchangeAgent0::Planning Module: Setting the agent name for
service with ID: 77158a:10abd317fb0:-7fcd to: iXchangeAgent1
ixchange.shared.ServiceImpl@97eded
```

The output demonstrates the iXchange system, starting with the planning module constructing a number of partial-plans. A service is then constructed from each partial plan requiring an action to be performed by an agent at a certain timepoint. The services are then passed to the social-reasoning module which determines which agents can execute each partial plan encapsulated in each service. The social-reasoning module then instantiates a dialogue goal which requires the dialogue module to communicate with the agent selected to perform the service and determine whether that agent will perform the required actions. If the recipient agent can perform the action then the dialogue is successful and the planner is informed via the social-reasoning module that an agent has been found which has accepted to perform the required action at the determined timepoint.

4. Conclusions

There are two facets to the results presented here. The first is that the i-Xchange is the first system to successfully integrate planning, social reasoning and argumentation. Though plans have been maintained in the context of agents communicating, this is the first time that individual agents have been equipped with modern planning techniques that are fully integrated with the communication subsystems, and where communicative “failures" (in the sense of refusals) have been taken into account by the planner on the fly. Similarly, though social structures have long formed a part of agent reasoning, and have contributed to the environment in which planners operate, this is the first time that plan refinement has explicitly involved reasoning about the social context. Finally, though argumentation has often been thought of in a social context [11], this is the first time that social reasoning has been integrated to the execution of specific argument protocols for inter-agent communication.

There is also a practical facet: the i-Xchange clearly demonstrates that the use of heterogeneous engineering techniques, different agent platforms and architectures, and a wide variety of languages and tools can provide a rich but solid foundation for a single, focused, coherent agent system. As the sophistication of individual systems continues to increase, and the scope of functionality becomes ever wider, such cross-platform heterogeneous development is going to become ever more the norm.

References

[1] A. L. Blum and M. L. Furst Fast planning through planning graph analysis. In *Artificial Intelligence*, 90: 281–300, 1997.

[2] M. N. Huhns and L. M. Stephens. Multiagent systems and societies of agents. In Gerhard Weiss, editor, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, chapter 2, pages 79–120. MIT Press, Cambridge, MA, USA, 1999.

[3] N. R. Jennings, S. Parsons, C. Sierra, and P. Faratin. Automated negotiation. In J. Bradshaw and G. Arnold, editors, *Proceedings of the 5th International Conference on the Practical Application of Intelligent Agents and Multi-Agent Systems (PAAM-2000)*, pages 23–30, Manchester, UK, 2000.

[4] A. Lomuscio, M. Wooldridge, and N. R. Jennings. A classification scheme for negotiation in electronic commerce. In F. Dignum and C. Sierra, editors, *Agent-Mediated Electronic Commerce: A European AgentLink Perspective*, volume 1991, pages 19–33. Springer Verlag, March 2001.

[5] N. R. Jennings, P. Faratin, A. R. Lomuscio, S. Parsons, C. Sierra, and M. Wooldridge. Automated negotiation: Prospects, methods and challenges. *Int. J. of Group Decision and Negotiation*, 10(2):199–215, 2001.

[6] R. Mailler, V. Lesser, and B. Horling. Cooperative negotiation for soft real-time distributed resource allocation. In *Proceedings of the Second AAMAS*, pages 576–583, 2003.

[7] J. Rosenschein and G. Zlotkin. *Rules of encounter : designing conventions for automated negotiation among computers*. MIT Press, Cambridge, MA, USA, 1994.

[8] N. R. Jennings, S. Parsons, P. Noriega, and C. Sierra. On argumentation-based negotiation. In *Proceedings of International Workshop on Multi-Agent Systems (IWMAS'98)*, Boston, USA, 1998.

[9] S. Parsons, C. Sierra, and N. R. Jennings. Agents that reason and negotiate by arguing. *Journal of Logic and Computation*, 8(3):261–292, June 1998.

[10] I. Rahwan, S. D. Ramchurn, N. R. Jennings, P. McBurney, S. Parsons, and L. Sonenberg. Argumentation-based negotiation. *The Knowledge Engineering Review*, 18(4):343–375, 2003.

[11] K. Sycara. Persuasive argumentation in negotiation. *Theory and Decision*, 28(3):203–242, May 1990.

[12] C. L. Hamblin. *Fallacies*. Methuen and Co. Ltd, 1970.

[13] J. D. Mackenzie. Question begging in non-cumulative systems. *Journal Of Philosophical Logic*, 8:117–133, 1979.

[14] P. McBurney and S. Parsons. Dialogue games in multi-agent systems. *Informal Logic*, 22(3):257–274, 2002.

[15] S. Wells and C. Reed. Formal dialectic specification. In I. Rahwan, P. Moraitis, and C. Reed, editors, *First International Workshop on Argumentation in Multi-Agent Systems*, 2004.