Playing with agent coordination patterns in MAGE

Visara Urovi and Kostas Stathis

Department of Computer Science,
Royal Holloway, University of London, UK
{visara,kostas}@cs.rhul.ac.uk

Abstract. MAGE (Multi-Agent Game Environment) is a logic-based framework that uses games as a metaphor for representing complex agent activities within an artificial society. More specifically, MAGE seeks to (a) reuse existing computational techniques for norm-based interactions and (b) complement these techniques with a coordination component to support complex interactions. The reuse part of MAGE relates physical actions that happen in an agent environment to count as valid moves of a game representing the social environment of an application. The coordination part of MAGE supports the construction of composite games built from component sub-games and corresponds to coordination patterns that support complex activities built from sub-activities. To illustrate the MAGE approach, we discuss how to use the framework to specify the coordination patterns required to form a virtual organisation in the context of a service-oriented scenario.

1 Introduction

Early work in multi-agent system has focused on the representation of agent interaction construed in terms of communication protocols that agents can use to interact with each other. As these protocols standartise the way in which agents partake in social activities, more recent work has put the emphasis on normative concepts such as obligation, permission, and prohibition, amongst other, to specify the social rules that represent agent protocols (see [3, 16]). However, despite the plethora of frameworks that support agent interactions about social concepts, there is relatively less work on how to represent systematically more complex activities that require agents to coordinate their actions when playing many protocols at the same time. There is, in other words, the need for computational frameworks that compose complex interactions and allow for their coordination.
Our specific motivation results from our participation in ARGUGRID [1], a research project that aims at providing a new model for programming a service Grid at a semantic, knowledge-based level of abstraction through the use of argumentative agent technology. Agents act on behalf of (a) users who specify abstract service requests or (b) providers who offer electronic services on the Grid. User requests result in agents interacting with other agents by forming dynamic Virtual Organisations (VOs) in

order to enable the transformation of abstract user requests to concrete services that the Grid can support. To guarantee that interactions in VOs are of a certain standard, agent-oriented provision of services must conform to service level agreements, while agent interaction more generally must be governed by electronic contracts. One of the requirements of ARGUGRID is that agreements and contracts need to be negotiated on the fly by agents, so there is the need to support protocols and workflows that enable the activities of VO creation, operation, and dissolution. One of the issues then becomes how to represent these complex activities at a knowledge-based level, suitable for argumentation-based agents to use as a framework to coordinate their interactions.

To manage agent coordination for VOs we present a logic-based framework that we call MAGE (Multi-Agent Game Environment). The idea behind MAGE is that the rules of a communication protocol between agents are viewed as the rules of an atomic game played amongst players, the speech acts uttered by agents represent the legal moves in the game, and the roles of agents in the interaction represent the roles of the players in the game. The contribution of MAGE is that given the representation of atomic games it provides a computational framework in which atomic games can be composed into composite ones and provides a systematic framework for their coordination. To illustrate how the resulting framework can be applied to a practical application, we show how to apply it in an ARGUGRID scenario that specifies workflows in terms of agent protocols to support the creation of a VO and its relevant electronic contracts.

The rest of the paper is organized as follows. Section 2 presents the context of the problem that we try to formulate and relates it to two kinds of games: atomic and compound. Atomic games and their specification are discussed in Section 3, while compound games and their specification are discussed in Section 4. Section 5 places our research in the context of existing literature and compares it to related work. We conclude with Section 6 where we also discuss our plans for future work.

2 ARGUGRID Games

We present a scenario that has motivated our work together with negotiation protocol used to negotiate services. We also discuss the link between the envisaged agent interactions and their representation as games. Once we have established this relation, we use it as the base of the MAGE computational framework developed in the next section.

2.1 The Earth Observation Scenario

The ARGUGRID scenario that we want to support considers a government ministry official requiring data about the detection of an offshore *oil spill* [19]. This abstract and high-level goal cannot be immediately satisfied by data within the ministry itself and requires the help of satellite companies that observe parts of the earth at different days. These companies publicise their services on a service Grid that is managed by agents. In this scenario a software agent takes the abstract request of the official and tries to instantiate it in a detailed set of services that can be invoked in sequence to provide the requested information. The scenario further assumes that satellite companies provide different services, each with different capabilities and costs, and one satellite may be more appropriate than another given certain conditions that the ministry sets. The official's software agent provides first a list of possible satellite companies that the official will need to select from, possibly with the advice of the agent, and, once the satellite companies are selected, the agent engages in a contract negotiation process with provider agents to create a VO that will instantiate the lower level services required to meet the official's request.

2.2 The Minimal Concession Protocol

Negotiation of contract terms in ARGUGRID uses a minimal concession protocol, with or without rewards, described in Dung et al [7], see Fig.1.

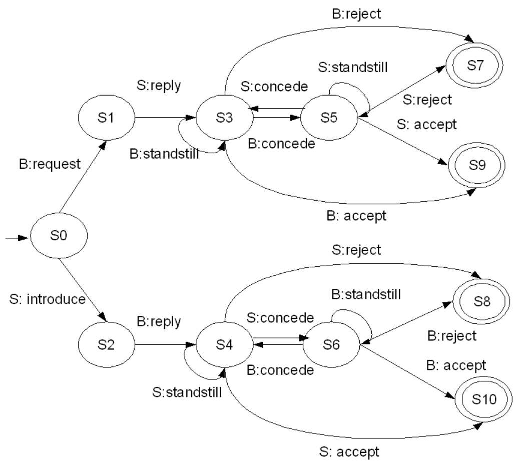


Fig. 1. The *Minimal Concession* protocol with Rewards [7]

The protocol provides the following set of locutions available to agents: *request, introduce, reply, concede, standstill, accept, reject.* The protocol assumes two agent roles, a *buyer* (B) and a *seller* (S). The protocol can start with an *introduce* move made by the seller or with a *request* move made by the buyer. These moves are used to respectively request or introduce an offer e.g. an oil spill detection service with some properties. Afterwards, a *reply* move can be made from the buyer to reply to an *introduce* move, or from a seller to reply to a *request* move. After this

move, *standstill, reject* or *concede* an offer are all moves that can be made by any role. The *accept* move terminates successfully the protocol and the accepted offer is considered the value of the result of the game. Three consecutive *standstill* moves are considered as a *reject* move, which terminates the protocol with no agreement.
An important property of this protocol is that if two agents use the protocol in conjunction with a minimal concession strategy, then every negotiation terminates successfully and the minimal concession strategy is in symmetric Nash equilibrium [7]. A minimal concession strategy is used if the offered service/product does not match what is requested. An agent can concede on a property of the service/product when it is possible to do so. Afterwards the agent will expect the other agent to concede as well allowing the offer to get closer to match the request and vice versa. If the agent, decides to not concede, it can standstill. The other agent will reply to a standstill with a concede locution if standstill is not a consecutive locution, otherwise it will standstill as well.

2.3 The VO Life-Cycle in ARGUGRID

The minimal concession protocol is only a component of the more complex activities that in ARGUGRID allow agents to form and participate in VOs, as shown in Fig. 2.3.

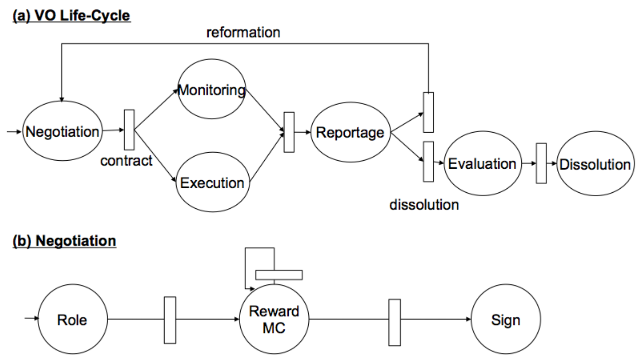


Fig. 2. Negotiation in the *VO Life-cycle* of ARGUGRID

Fig. 2.3(a) shows how by negotiating a successful contract starts the execution and monitoring activities of a VO. Issues raised by the monitoring or execution activities are reported and the VO must in this case be reformed via re-negotiation. If, however, reformation does not apply to these issues or if the execution has fulfilled the goals of the VO creation, the VO is dissolved by having its result being evaluated first. Activities in VOs may require further control-flows for sub-activities as shown in

Fig. 2.3(b); this illustrates how the activity of negotiation is in fact a more complex activity that requires first to determine the roles of the agents in the VO, then negotiate the terms of the VO contract using the minimal concession protocol, and finally the complete contract must be signed by all relevant parties. The details for the remaining activities of monitoring, execution, reportage, evaluation, and dissolution, are beyond the scope of this work. In the remainder of this paper we focus on how to model the control flows of activities as a complex game exemplified by the negotiation activity.

2.4 VO Activities as Complex Games

The games metaphor was originally proposed to model human-computer interaction by Stathis and Sergot in [18] and was subsequently applied to formulate agent interaction protocols in [17]. We outline next how agent activities about VOs in ARGUGRID can be formulated as games.
The basic unit of the games metaphor is the notion of an atomic game, which describes a set of rules about an initial state, a set of player roles, a set of game moves, the effects the moves have on the state, a specification of when a move is legal, a set of terminating states, and a set of results [17]. The minimal concession protocol described earlier seen as a game implies that the initial state of the protocol is the initial state of the game, the roles of the participating agents are the roles of the players, the protocol locutions are the game moves, the effects of locutions on the protocol state are the effects of moves on the state of the game, the preconditions of locutions are the valid moves, the final protocol states are the terminating states of the games, the set of protocol outcomes are the possible game results. The result of a game does not necessarily need to be zero-sum [13], by requiring a winner and a loser, but it can also give rise to a win/win or loose/loose situations.
To obtain complex interactions we combine atomic games to build more complex, composite games. An example of a complex interaction is the control flow of the Fig. 2.3(b), where we need to combine three different games: first the agents can play a *role negotiation* game to determine their roles, after establishing their roles, they play a *minimal concession with reward* game to agree on the terms of the contract, they can reiterate this game for as long they find an agreement and finally the *sign* game becomes active for the agents to sign the contract.
In a composite game we want to be able to parallelise, choose or synchronize atomic games. In order to capture these control-flow aspects we use workflow concepts. In general, the term *workflow* refers to the specification of a work procedure or a business process in a set of *atomic activities* and relations between them in order to coordinate the *participants* and the activities they need to perform [2]. In our case, the *participants* are the agents and the *atomic activities* are the atomic games. We consider atomic games as atomic activities and then use basic patterns to describe complex workflows as a combination of atomic games into more complex structures. A composite game then is a collection of flow patterns capturing the coordination mechanisms for atomic games. We will see later how our example of the negotiation composite game (illustrated

in Fig. 2.3(b)) will be defined as an aggregation of three patterns: a sequence, a conditional, and an iteration pattern.

3 Atomic Games in MAGE

Following earlier work on the games metaphor [18], we view communicative interactions within an agent society abstractly as game interactions [17]. As the rules of a game represent all valid evolutions of the game's state, we use the following logic program to describe the rules of a game:

```
game(State, Result)←
      terminating(State, Result).
game(State, Result)←
      not terminating(State, Result),
      valid(State, Move),
      effects(State, Move, NewState),
      game(NewState, Result).
```

To formulate a particular game we need to decide how to represent a game state, its initiating and terminating states, how players make valid moves, and how the effects of these moves change the current state to the next one until the terminating state is reached.

3.1 The State of Atomic Games

To represent the State of a game we use a term of the form Id@T, where Id is a unique identifier of a complex term describing the attributes of the state's configuration, and T is the system's time that uniquely identifies the actual evolutions of the complex term as a result of the interaction. The rationale behind this kind of representation is that in MAGE we acknowledge the fact that the interaction within a multi-agent system application can become quite complex. To cater for the complexities of practical applications we assume that complex terms have an underlying object-based data-model. To represent complex terms we use the syntax of C-Logic [5]. A term of the form:

```
min_concession:mc1 [
    parties⇒ {agent:a1 [role ⇒ seller], agent:a2 [role⇒buyer]},
    buyer_position ⇒ offer:o1 [price ⇒80, resolution⇒20, delivery ⇒2],
    seller_position ⇒ offer:o2 [price ⇒100, resolution⇒20, delivery ⇒2],
    standstill_count ⇒ 1,
    result ⇒ nil.
]
```

is identified by mc1 denoting an instance of an object whose class is that of a minimal concession protocol with two participating agents a1 and a2, complex terms whose role attribute is seller and buyer respectively, where the buyer in the previous round has made an offer o1 (a complex term), while the seller has made another offer o2 (another complex term), there is one standstill move that has been encountered, and the result of the interaction is still incomplete as the value is still nil. Such a complex term has a first-order logic translation, see [5] for details.

3.2 State Evolution

The moves of the game are represented by complex terms too. The complex term below

```
speech_act:m1[actor ⇒ a1, act⇒introduce, offer ⇒ o1, role⇒ seller],
```

describes that the seller agent a1 utters introduce about an offer o1. Such moves are used as the contents of events that happen at a specific time. An assertion of the form happens(m1, 12), states that move m1 has happened at time 12. Such an event changes the state of a game.

```
holds_at(Id, Class, Attr, Val, T)←
  happens(E, Ti), Ti ≤ T,
  initiates(E, Id, Class, Attr, Val),
  not broken(Id, Class, Attr, Val, Ti, T).

broken(Id, Class, Attr, Val, Ti, Tn)←
  happens(E, Tj), Ti < Tj ≤Tn,
  terminates(E, Id, Class, Attr, Val).

holds_at(Id, Class, Attr, Val, T)←
  method(Class, Id, Attr, Val, Body),
  solve_at(Body, T).

attribute_of(Class, X, Type)←
  attribute(Class, X, Type).
attribute_of(Sub, X, Type)←
  is_a(Sub, Class),
  attribute_of(Class, X, Type).

instance_of(Id, Class, T)←
  happens(E, Ti), Ti ≤ T,
  assigns(E, Id, Class),
  not removed(Id, Class, Ti, T).
removed(Id, Class, Ti, Tn)←
  happens(E, Tj), Ti < Tj ≤ Tn,
  destroys(E, Id).

assigns(E, Id, Class)←
  is_a(Sub, Class),
  assigns(E, Id, Sub).

terminates(E, Id, Class, Attr, _)←
  attribute_of(Class, Attr, single),
  initiates(E, Id, Class, Attr, _).

terminates(E, Id, _, Attr, _)←
  destroys(E, Id).
terminates(E, Id, _, Attr, IdVal)←
  destroys(E, IdVal).
```

Fig. 3. A subset of the *Object-based Event Calculus* from [9]

We use the object-based event calculus (OEC) of Kesim and Sergot [9] to capture state changes of complex terms. A subset of the OEC is given in Fig. 3. The first two clauses derive the value of an attribute for a complex term holds at a specific time. The third clause describes how to represent derived attributes of object as method calls computed by means of a solve_at/2 meta-interpreter as specified in [10]. The fourth and fifth clauses support a monotonic inheritance of attributes for a class limited to the subset relation. The sixth and seventh clauses determine how to derive the instance of a class at a specific time. The effects of an event on a class is given by assignment assertions; the eighth clause states how any new instance of a class becomes a new instance of the super-classes. Finally, the ninth clause deletes single valued attributes that have been updated, while the tenth and eleventh clauses delete objects and dangling references.

3.3 Valid Moves and their Effects

Before the event of a move being made in the state of the game, we must have a way to check that the move is valid. One simple definition is to make valid moves equivalent to the legal moves of the game:

```
valid(State, Move) ↔ legal(State, Move).
```

To specify valid moves, we specify when moves are legal. For example, to specify when a request move is legal in the minimal concession protocol we write:

```
legal_at(Id@T, Move) ←
   instance_of(Id, min_concession, T),
   speech_act:Move[actor ⇒ A, act⇒request, offer ⇒ Product, role⇒ buyer],
   holds_at(S, agent_of, A, T),
   holds_at(A, role, buyer, T).
```

Other definitions of valid moves are possible, for instance, Artikis et al [3] provide a more detailed account of valid moves in terms of social concepts such as obligations, permission and power. The important point here is that our framework can accommodate these for an application by providing a different definition of valid/2.
Once a move has been determined as valid, a new state of the game must be brought about due to the effects of the move. As by making moves players cause events to happen, if we assume that the happening of such moves take only one unit of time, we can specify their effects as:

```
effects(Id@T, Move, Id@NewT) ←
       add(happens(Move T)),
       NewT is T + 1.
```

In our representation of state, once an event has happened, its effects are added to the state implicitly, via inititiates/4 definitions that initiate new values for attributes of a state term, terminates/4 clauses that remove attribute values from a state term, and assigns/3 definitions for assigning to ids new instances of terms. An example, of how new values are initiated for attributes for the minimal concession protocol is given below:

```
initiates(Ev, Id, seller_position, Offer)←
    happens(Ev, T),
    instance_of(Id, min_concession, T),
    Ev[act ⇒ Act, actor ⇒ Aid, role ⇒ seller, offer ⇒ Offer],
    changes_seller_position(Act).

changes_seller_position(introduce).
changes_seller_position(concede).
changes_seller_position(reply).
```

The above definition initiates the current position made by a seller to be stored in the state of the game as a result of a request, reply or concede move. The old offer is terminated and substituted by a new request because of the way the object event calculus is specified (see the ninth clause in Fig. 3).
It is important to note that other specifications of effects/3 are possible depending on what assumptions we make about the duration of moves captured in events. In addition, the state could be represented explicitly as a set of assertions as in [17] rather that implicitly, with rules that define what holds in it, as in MAGE. Both of these issues, however, are beyond the scope of this paper. It suffices to say here that once a choice of state representation has been made, the framework can accommodate them by suitably adjusting the effects/3 definition.

3.4 Initial and final states of a game

For the state of an atomic game to be created, the framework discussed so far requires the assertion of an event that will first create the term via an assigns/3 assertion. The assertion:

```
assigns(Ev, Id, min_concession)←
    Ev[act ⇒ construct, protocol ⇒ min_concession, id ⇒ Id].
```

will allow the creation of an instance for the minimal concession protocol, which can then be queried using the sixth clause of Fig. 3. To complete the instantiation process we also need to specify the initial values for the attributes of the complex term representing the minimal concession protocol. For this we need to define separately the initiates/4 rules as the one below:

```
initiates(Ev, Id, party_of, Val)←
      Ev[act ⇒ construct, protocol ⇒ min_concession, parties ⇒ agent: Val].
```

Additional initiates/4 clauses are needed to define the whole of the initial state, one for each attribute value.
The initial state of the game will evolve as a result of moves been made in the state of a game. This state will eventually reach the final state from which we can extract the game's result. We specify this via terminating/2 predicates. For example, the definition:

```
terminating(Id@T, Result)←
      instance_of(Id, min_concession, T),
      holds_at(Id, result, R, T),
      not R==nil.
```

specifies the conditions under which the minimal concession protocol will terminate and at the same time returns the result.

4 Compound Games in MAGE

Compound games are complex games composed from simpler, possibly atomic, sub-games. Based on our previous work in applying compound games to develop multi-agent systems [17], in this section we show how to develop compound games in the MAGE framework, with aim to support the coordination of complex agent activities such as ARGUGRID workflows.

4.1 A Compound Game

To give an example of how sub-games will appear in the main game, consider as an example the state of the VO negotiation in ARGUGRID, as specified in Fig. 2.3.

```
vo_negotiation: Id [
 parties ⇒ {agent:a1, agent:a2, agent:a3},
 process ⇒ Workflow
]
```

The sub-games of VO negotiation are specified in the Workflow value of the process attribute, instantiated to terms of the form:

```
seq((new(roles, r1),
      roles:r1,
      if(r1[result⇒success, repeat(seq(new(m1,mcwr), mcwr:r1), m1[result⇒exit])),
      if(m1[agreement⇒achieved, seq((new(s1,sign), sign:s1)))
)
```

The above term states that the process of the negotiation is a sequence (seq) of sub-games involving first a sub-game that allows the construction of a new game r1 for defining the VO roles. Then the newly created game roles:r1 must be played, and if the result of the roles game is success, the roles agents must enact in the VO have been agreed (if), and the workflow must continue with repeatedly creating a minimal concession protocol mcwr with identifier m1 and playing it until the result of this game is exit (repeat). This means that either an agreement has been achieved during the negotiation or the game has been played more than a certain maximum and no agreement was achieved. Only if the agreement attribute of m1 is set to achieved, the construction of new sign game with identifier is started and played to complete the negotiation process.

4.2 Coordination of active sub-games

The main issue to be considered in compound games is the coordination of moves in active sub-games. We define coordination specifying the predicate active_at/3. Using active sub-games, we can define valid moves in a complex game to include all the valid moves in the active sub-games:

```
valid(Id@T, Move) ←active_at(Id, SubId, T), valid(SubId@T, Move).
```

For VO negotiation we define active subgames as follows:

```
active_at(Id, SubId, T)←
        instance_of(Id, neg, T),
        Id [process⇒Workflow],
        pattern(Workflow),
        runs(Id, Workflow, SubId, T).
```

Patterns in our framework are interpreted by a runs/4 predicate that parses the coordination structure and checks which sub-games are running. For the VO negotiation process three patterns are required: a sequence, an if-conditional, and a repeat loop, as specified below.

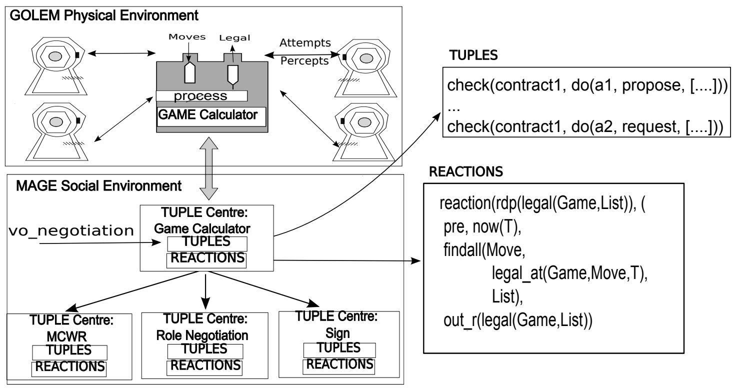
```
runs(G, seq((A,B)), A, T)←
      not pattern(A),
      not terminating(A@T,_).
runs(G, seq((A,B)), C, T)←
      not pattern(A),
      terminating(A@T,_),
      runs(G, seq(B), C, T).
runs(G, seq((A,B)), C, T)←
      if-conditional(A),
      runs(G, A, C, T).
runs(G, seq(A,B), C, T)←
      repeat-loop(A),
      runs(G, A, C, T).
```

```
runs(G, if(Id[Prop⇒Val], P), A, T)←
      holds_at(Id, Prop, Val, T),
      runs(G, P, A, T).
runs(G, repeat(P, Id[Prop⇒Val]), A, T)←
      not holds_at(Id, Prop, Val, T),
      runs(G, P, A, T).

pattern(P)← sequence(P).
pattern(P)← if_conditional(P).
pattern(P)← repeat_loop(P).

sequence(seq((_,_))).
if_conditional(if(_,_)).
repeat_loop(repeat(_,_)).
```

Note that the top-leve game G is required as a parameter in the definition of runs/4 as a reference to the global variables of the interaction. Note also that the definition of the above patterns can be combined to form arbitrary complex structures. In particular, more workflow primitives are specified in a similar manner [21], but they are not discussed here due to lack of space.

4.3 Implementation status

We have built a prototype of MAGE that allows the deployment of a set of distributed objects in the GOLEM platform [4]. We call these objects *Game Calculators* for GOLEM agents to interact with them to coordinate their interactions as shown in Fig. 4. More specifically, GOLEM agents can call methods of a calculator object by means of actions performed in the environment. The content of such actions represents a move in the compound game.



```
check(contract1, do(a1, propose, [....]))
...
check(contract1, do(a2, request, [....]))
```

```
reaction(rdp(legal(Game,List)), (
 pre, now(T),
 findall(Move,
         legal_at(Game,Move,T),
         List),
 out_r(legal(Game,List))
```

Fig. 4. Implementing MAGE using TuCSoN and GOLEM

To implement games, we link the internal part of the Game Calculator object with a TuCSoN tuple centre [14], a Linda-like extension of the concept of tuple space as a reactive logic based blackboard.

Using TuCSoN, compound and atomic games are conceived as a combination of the ReSPeCT language [14] and the OEC. A Game Calculator is represented by a tuple centre that can be configured from an agent (either a coordinator agent or the agent who is interested to start the negotiation) to work as a specific compound game (such as VO negotiation). The Game Calculator can communicate with multiple tuple centres representing the atomic games. Further details of the implementation are beyond the scope of this work.

5 Related Work

The Electronic Institution (EI) approach [8] uses organisational concepts to model the interaction of agents. Our framework is similar to EIs in the sense that their *scenes* are our atomic games and their *norms* as the rules that capture the validl moves for the agent as the game progresses. EIs also support a *performative structure* that enables a developer to define dependencies such as choice points, synchronization and parallelism mechanisms between scenes based on role flow policies among scenes specifying which paths can be followed by which agent's role. In our framework the EI performative structures are defined as compound games that structure atomic games, which can be coordinated by activity patterns.
Artikis et al [3] propose a model for norm-governed multi-agent systems as executable specification of open agent societies. This work represents social constraints by making a clear distinction between physical capabilities, institutional power, permissions and sanctions to enforce policies. Social constrains are a sophisticated version for defining our valid moves of a game that captures the social state of the interaction. We too distinguish between possible actions happening in the environment supported by GOLEM, from social actions happening in MAGE, and we link them via physical objects that support agent coordination. As our focus is on coordination and as their emphasis is on normative concepts, the two approaches can be seen as complementary to each other, especially as they both use the Event Calculus as the underlying computational mechanism, even if we assume an object-based data-model. However, in our model we do not prove properties of interactions, which can be an extension of our work.
McBurney and Parson [11] present an abstract framework to represent complex dialogues as sequences of moves in a combination of dialogue games. Agents agree the game the need to play at a control layer, in our terms a compound game, and then play the protocol at an execution layer, in a our case a sub-game. The framework admits combinations of different dialogue types that in our framework corresponds to the coordination of compound games. However, McBurney and Paron's dialogical games abstract away from the game state and they do not define the valid moves as a way of analysing the different kinds of pre-conditions and post-conditions on the state of the interactions. Instead their formalism is based on agents selecting and agreeing to play these dialogues. On the contrary our framework seeks to provide a computational mechanism for coordination in complex interactions that are construed as compound games.
Kesim et al [6] propose a framework to specify and execute workflows based on Event Calculus. In this work, EC is used to describe the specification and execution of activities in a workflow. The activities are assigned to agents using a coordinator agent that knows which agents can perform which activities. Similarly to Kesim's work we use EC to define workflows but we use workflows to dynamically define compositions of games. Another difference with Kesim et al is that activities are tasks to be executed from one agent. In our case we have atomic games that

become active to enable many agents to socially interact and coordinate using games.
Omicini et al [15] propose a model to distribute a workflow among different tuple centres (conceived as the entities that coordinate agent's activities) by linking tuple centres with linkability operators. In our approach we use the linkability of tuple centres as the coordination mechanism that the Game Calculator uses to start and terminate new games. We also provide a representational framework that can be used systematically to represent patterns of interactions, like workflows.

6 Conclusions and Future Works

We have presented MAGE, a logic-based framework that uses games as a metaphor for representing complex agent activities within an artificial society. We have illustrated how MAGE can reuse existing computational techniques for norm-based interactions and support their coordination. Using examples from the ARGUGRID projects, we have illustrated how the reuse part of MAGE relates physical actions that happen in an agent environment to count as valid moves of a game representing the social environment of an application. Coordination in MAGE supports the construction of complex games built from component sub-games and corresponds to coordination patterns that support complex activities built from sub-activities. We have discussed how to use the framework to specify the coordination patterns required to form a virtual organisation in ARGUGRID.
Future work involves to formulating the whole of the VO lifecycle of ARGUGRID in MAGE to build a library of reusable coordination patterns for similar applications.

References

1. ARGUmentantion as a foundation for the semantic GRID (ARGUGRID). http://www.argugrid.eu/, 2009.
2. Workflow Management Coalition. http://www.wfmc.org/, 2009.
3. Alexander Artikis, Marek J. Sergot, and Jeremy V. Pitt. Specifying Norm-Governed Computational Societies. *ACM Trans. Comput. Log.*, 10(1), 2009.
4. Stefano Bromuri and Kostas Stathis. Situating Cognitive Agents in GOLEM. In *Engineering Environment-Mediated Multi-Agent Systems, EEMMAS 2007*, volume 5049/2008 of *Lecture Notes in Computer Science*, pages 115–134. Springer, 2007.
5. W. Chen and D. S. Warren. C-logic of Complex Objects. In *PODS '89: Proceedings of the eighth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 369–378, New York, NY, USA, 1989. ACM Press.
6. Nihan Kesim Cicekli and Yakup Yildirim. Formalizing Workflows Using the Event Calculus. In *DEXA'00: Proceedings of the 11th International Conference on Database and Expert Systems Applications*, pages 222–231, London, UK, 2000. Springer-Verlag.

7. Phan M. Dung, Phan M. Thang, and Francesca Toni. Argument-based Decision Making and Negotiation in E-business: Contracting a Land Lease for a Computer Assembly Plant. In *9th International Workshop on Computational Logic in Multi-Agent Systems (CLIMA)*, Dresden, 2008.
8. Marc Esteva, Bruno Rosell, Juan A. Rodriguez-Aguilar, and Josep Ll. Arcos. Ameli: An agent-based middleware for electronic institutions. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 236–243, Washington, DC, USA, 2004. IEEE Computer Society.
9. F. Nihan Kesim and Marek Sergot. A Logic Programming Framework for Modeling Temporal Objects. *IEEE Transactions on Knowledge and Data Engineering*, 8(5):724–741, 1996.
10. Nihan Kesim. *Temporal Objects in Deductive Databases.* PhD thesis, Imperial College, 1993.
11. Peter McBurney and Simon Parsons. Games that agents play: A formal framework for dialogues between autonomous agents. *Journal of Logic, Language and Information*, 11(3):315–334, 2002.
12. Jarred McGinnis, Stefano Bromuri, Visara Urovi, and Kostas Stathis. Automated workflows using dialectical argumentation. In *GES07*. German e-Science Conference, 2007.
13. Roger B. Myerson. *Game Theory: Analysis of Conflict.* Harvard University Press, September 1997.
14. Andrea Omicini and Enrico Denti. From Tuple Spaces to Tuple Centres. *Science of Computer Programming*, 41(3):277–294, nov 2001.
15. Andrea Omicini, Alessandro Ricci, and Nicola Zaghini. Distributed workflow upon linkable coordination artifacts. In Paolo Ciancarini and Herbert Wiklicky, editors, *Coordination Models and Languages*, volume 4038 of *Lecture Notes in Computer Science*, pages 228–246. Springer, 2006.
16. Adrian Paschke and Martin Bichler. SLA Representation, Management and Enforcement. In *EEE '05: Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'05) on e-Technology, e-Commerce and e-Service*, pages 158–163, Washington, DC, USA, 2005. IEEE Computer Society.
17. Kostas Stathis. A Game-based Architecture for Developing Interactive Components in Computational Logic. *Journal of Functional and Logic Programming*, 2000(5), 2000.
18. Kostas Stathis and Marek J. Sergot. Games as a Metaphor for Interactive Systems. In *In HCI'96, People and Computers XI.*, pages 19–33. Springer-Verlag, 1996.
19. Francesca Toni. E-business in ArguGRID. In Jörn Altmann and Daniel Veit, editors, *Grid Economics and Business Models, 4th International Workshop, GECON 2007*, volume 4685 of *Lecture Notes in Computer Science*, pages 164–169. Springer, 2007.
20. Visara Urovi, Stefano Bromuri, Jarred McGinnis, Kostas Stathis, and Andrea Omicini. Experiences in automated workflows using dialectical argumentation. In *IADIS International Conference "Intelligent Systems and Agents" (ISA 2007)*, Computer Science and

Information Systems, pages 3–8, MCCSIS 2007, Lisbon, Portugal, 3–8 July 2007. IADIS Press.

21. Wil M. P. van der Aalst, Arthur ter Hofstede, Bartosz Kiepuszewski, and Ana Barros. Workflow patterns home page. http://www.workflowpatterns.com/, 2009.