# The AORTA Architecture: Integrating Organizational Reasoning in *Jason*

Andreas Schmidt Jensen[1], Virginia Dignum[2], and Jørgen Villadsen[1]

[1] Technical University of Denmark, Kongens Lyngby, Denmark
{ascje,jovi}@dtu.dk
[2] Delft University of Technology, Delft, The Netherlands
m.v.dignum@tudelft.nl

**Abstract.** Open systems are characterized by a diversity of heterogeneous and autonomous agents that act according to private goals, and with a behavior that is hard to predict. They can be regulated through organizations similar to human organizations, which regulate the agents' behavior space and describe the expected behavior of the agents. Agents need to be able to reason about the regulations, so that they can act within the expected boundaries and work towards the objectives of the organization. In this paper, we propose the AORTA[1] architecture for making agents organization-aware. It is designed such that it provides organizational reasoning capabilities to agents implemented in existing agent programming languages without being tied to a specific organizational model. We show how it can be integrated in the *Jason* agent programming language, and discuss how the agents can coordinate their organizational tasks using AORTA.

## 1 Introduction

Open systems rely on organizational structures to guide and regulate agents, because these systems have no control over the internal architecture of the agents. This means that the agents must be able to reason about the organizational structures in order to know what to do in the system and how to do it. Regulations are often specified as organizational models, usually using roles that abstract away from specific agent implementations such that any agent will be able to enact a given role. Roles may restrict enacting agents' behavior space, such that it coincides with the expectations of the system. The system can then be regulated, for example, by blocking certain actions (for example through a *middleware*, such as $\mathcal{S}$-$\mathcal{M}$OISE$^+$ [9]), or by enabling the agents to reason about the expectations of the system.

[1] **A**dding **O**rganizational **R**easoning **t**o **A**gents

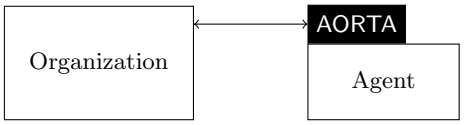


**Fig. 1.** AORTA is part of an agent and provides it with an interface to the organization.

Agents that can reason about organizations are *organization-aware* [14]. This includes understanding the organizational specification, acting using organizational primitives, and cooperating with other agents in the organization to complete personal or organizational objectives. From the agent's perspective, there are two sides of organizational reasoning. First, how can it contribute to the objectives of the organization, and second, how can it take advantage of the organization, once it is a part of it.

AORTA (Adding Organizational Reasoning to Agents) [12] is an organizational reasoning component that can be integrated into the agent's reasoning mechanism, allowing it to reason about (and act upon) regulations specified by an organizational model using simple reasoning rules. AORTA assumes a preexisting organization, is independent from the agent, and focus on reasoning rules that specify how the agent reasons about the specification. The organization is completely separated from the agent, as shown in figure 1, meaning that the architecture of the agent is independent from the organizational model, and the agent is free to decide on how to use AORTA in its reasoning. The separation is possible because AORTA is tailored based on an organizational metamodel, designed to support different organizational models.

In this paper, we propose the AORTA architecture for making agents organization-aware[2]. It is designed such that it can provide organizational reasoning capabilities to agents implemented for existing agent platforms. We present an integration of AORTA in the well-known agent platform *Jason* [1], and show how it lets *Jason*-agents decide how to use their capabilities to achieve their organizational objectives, and furthermore, how they are able to coordinate their tasks.

We consider software architecture as the highest level of abstraction of a software system. The AORTA architecture is designed as a component that can be integrated into existing agent platforms. Existing agents are linked to an AORTA-agent, which features an organizational reasoning cycle that performs organizational reasoning, providing the existing agent with organizational reasoning capabilities. Furthermore, the organizational reasoning is specified in an AORTA-program in which organizational actions and coordination mechanisms for each agent can be defined by the developer.

The rest of the paper is organized as follows. We begin, in section 2, with a description of the organizational metamodel, and briefly discuss a simple scenario, which we later implement in AORTA and *Jason*. In section 3, we present

[2] The implementation of the AORTA architecture is available as open source at `http://www2.compute.dtu.dk/~ascje/AORTA/`.

the AORTA architecture. Section 4 describes the integration with *Jason*. We discuss related work in section 5 and conclude the paper in section 6.

## 2 Organizational model

Organizational models are used in multi-agent systems to give agents an explicit representation of an organization. Different models are proposed in the literature (e.g. $\mathcal{M}$OISE$^+$[9], OperA [5], ISLANDER [6]). A common trait is the use of *roles*, abstracting implementation details away from expectations, and *objectives*, defining the desired outcome of the organization.

Reasoning in AORTA is based on an organizational metamodel, which supports different organizational models. The metamodel is based on roles and objectives.

A role, $\mathsf{role}(r, O)$ has a *name*, $r$, and is responsible for a set of *objectives*, $O$. An objective is denoted $\mathsf{objective}(o)$. Roles can form a *dependency* relation over an objective, $\mathsf{dependency}(r_1, r_2, o)$, such that $r_1$ depends on $r_2$ for the completion of an objective $o$. Objectives may be partially *ordered*, $\mathsf{order}(o_1, o_2)$, indicating that certain objective $o_1$ must be completed before objective $o_2$. Role enactment is denoted $\mathsf{rea}(a, r)$, which means that agent $a$ enacts role $r$. Furthermore, $\mathsf{active}(o)$ denotes objective $o$ is active (an objective is active if it has not yet been completed and all objectives it depends on have been completed).

Existing organizational models can be mapped to this metamodel. For example, if $\mathcal{M}$OISE$^+$ is being used, an objective is a goal, which is part of a mission, and a role would be responsible for missions it is permitted or obligated to pursue. Note that since the metamodel is currently based on roles and objectives, and has no notion of norms, it is not yet possible to reason about norms that are enforced.

### 2.1 First responders

We consider a scenario of first responders at a fight between groups of people, some of them being injured and requiring medical attention.

> *After a match between Manchester United and Manchester City, the fans are fighting and some of them are badly hurt. The authorities have been contacted, and a group number of medics and police officers (the first-responders) have arrived. The medics are supposed to help the injured, while the police officers are supposed to break up the fight. The fans may try to prevent medics from rescuing injured fans from the other team.*

The organizational specification is shown in figure 2. For this paper, we assume that the agents entering the organization are *cooperative*, that is, they will pursue organizational objectives and cooperate with the other agents in the organization. It is, however, simple enough to consider self-interested agents as well; they will just be more likely to pursue their personal objectives rather than those of the organization.

role(*medic*,{*injuredFound*,*injuredSaved*,*removeBlocker*}).
role(*officer*,{*fightFound*,*fightStopped*}).
dependency(*medic*,*officer*,*removeBlocker*).
order(*injuredFound*,*injuredSaved*).
order(*fightFound*,*fightStopped*).
objective(*injuredFound*).
objective(*injuredSaved*).
objective(*removeBlocker*).
objective(*fightFound*).
objective(*fightStopped*).

**Fig. 2.** Organizational specification of the crisis response scenario.

Agents in the scenario will have to reason about which role(s) to enact, how to achieve and coordinate their objectives, and how to complete objectives that the agents are not capable of achieving themselves (i.e., by delegating to another, more capable agent).

## 3 The AORTA architecture

Classical BDI agents are represented by sets of beliefs, desires and intentions, where desires are possible states of affairs that the agent might want to realize, and intentions are those states of affairs that the agent has committed to (attempt to) realize. A similar representation can be made for organizational reasoning: the agent holds beliefs about the organization (its specification and instantiation) and can use that for reasoning about organizational objectives that are possible (or required) to be achieved, roles that can be enacted, norms that are enforced, and so on. An integration of the organization within the agent, makes the agent more likely to take both the organization and its own beliefs into account in its reasoning. Furthermore, by representing the organization as beliefs, the organizational structure can be changed, if necessary. For example, if the organization changes (reorganization), or if the agent finds out that it has wrong beliefs about the organization.

AORTA provides organizational reasoning capabilities to agents, and extends classical BDI reasoning, allowing the agents to reason about organizational matters. Organizational reasoning is divided into *organizational option generation*, *organizational action deliberation* and *organizational coordination*. An organizational option is something that the agent should consider, such as an active objective, or a role that can be enacted or deacted [11]. For instance, initially in the scenario, the medics will only search for injured people. When all areas have been searched, this objective has been completed and a new objective, rescuing the injured, will be possible. An organizational action is the execution of an organizational option: actually enacting a role or committing to an organizational objective. This creates the expectation (for the organization) that the agent should somehow believe it is able to (help) achieving it, either by itself, by cooperating with other agents, or by delegating it to one or more agents in

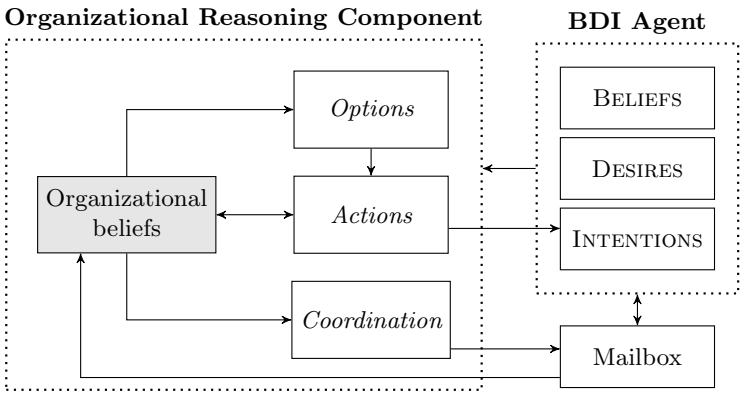


**Fig. 3.** The Organizational Reasoning Component of AORTA.

the dependency relation of its role. Note that self-interested or deceitful agents might know that they cannot achieve an organizational objective, but will commit to it anyway to disturb the organization. Organizational coordination is organization-level coordination, which is based on the agent's mental state.

The *organizational reasoning component* of AORTA is depicted in figure 3. The agent (assumed to be a BDI agent) has a mental state, which is coupled to AORTA. Based on the mental state, AORTA can determine which organizational options to choose, and the organizational actions might change the mental state. For instance, in order to consider the available organizational options, AORTA uses the agent's capabilities and intentions. Furthermore, intentions may influence the reasoning, e.g., when the intention to coordinate a task requires use of the organizational model. Finally, AORTA lets agents commit to objectives: an organizational action leads to change in the agent's intentions, corresponding to the fact that the agent commits to the objective. The coordination component sends messages using the mailbox, and incoming messages can change the organizational structure.

### 3.1 Mental state

BDI agents usually have knowledge bases containing their beliefs and intentions. AORTA-agents are agents that contain an AORTA-component, which means that they not only have belief and intention bases, they also have knowledge bases for the organizational aspect. Each knowledge base will hold different kinds of formulas depending on their purpose.

**Definition 1 (Knowledge bases).** *The AORTA knowledge bases are based on a predicate language, $\mathcal{L}$, with typical formula $\phi$ and operators $\wedge, \neg, \forall$. The agent's belief base and intention base are denoted $\Sigma_a$ and $\Gamma_a$, respectively. The language of the organization is denoted $\mathcal{L}^{\text{org}}$, and $\mathcal{L}^{\text{org}} \subseteq \mathcal{L}$. The organizational specification and options are denoted $\Sigma_o$ and $\Gamma_o$, respectively. We then have the following*

*knowledge bases:*

$$\Sigma_o, \Gamma_o \subseteq \mathcal{L}^{\text{org}} \qquad \Sigma_a, \Gamma_a \subseteq \mathcal{L}$$

We define different kinds of formulas for each knowledge base, which allows us to target specific knowledge bases in different situations.

**Definition 2 (Formulas).** *AORTA uses reasoning formulas, $\mathcal{L}_R$, with typical element $\rho$, which are based on organizational formulas, option formulas, belief formulas and goal formulas.*

$$\rho ::= \top \mid \mathit{org}(\phi) \mid \mathit{opt}(\phi) \mid \mathit{bel}(\phi) \mid \mathit{goal}(\phi) \mid \neg\rho \mid \rho_1 \wedge \rho_2$$

Organizational formulas, $\mathsf{org}(\phi)$, queries the organizational specification, option formulas, $\mathsf{opt}(\phi)$, queries the options base, belief formulas, $\mathsf{bel}(\phi)$, queries the belief base and goal formulas, $\mathsf{goal}(\phi)$, queries the intention (or goal) base. We can use the formulas to specify things such as:

$$\mathsf{org}(\mathsf{objective}(\mathit{injuredFound})) \wedge \neg\mathsf{bel}(\mathit{injuredFound}),$$

where the first part of the conjunction queries the organizational specification, $\Sigma_o$, and the second part queries the agent's belief base, $\Sigma_a$. The formula queries whether there is an organizational objective (to find victims), which the agent currently does not believe it has achieved.

**Definition 3 (Mental state).** *The AORTA mental state, MS, is a tuple of knowledge bases:*

$$MS = \langle \Sigma_a, \Gamma_a, \Sigma_o, \Gamma_o \rangle.$$

The implementation of the mental state is based on tuProlog [4], which is a Java-based lightweight implementation of ISO-Prolog. We chose tuProlog because of its efficiency and straightforward interface in Java, allowing us to query a Prolog database without requiring any external system-dependent libraries. Each AORTA-agent has its own instance of tuProlog, comprising its entire mental state. That is, all knowledge bases of an agent are implemented in a single Prolog instance by wrapping each rule in a predicate depending on its nature. For example, the reasoning formula $\mathsf{bel}(a \wedge b) \wedge \neg\mathsf{org}(c \wedge d)$ is converted to the following Prolog query: `bel(a), bel(b), \+ (org(c), org(d))`. This translation makes querying straightforward, while still keeping the distinction between the different knowledge bases.

Note that we let AORTA-agents have their own mental state, rather than integrating AORTA into the knowledge bases of an agent in an existing platform. This means that the belief base and goal base of AORTA must be synchronized with the agent, which could lead to pitfalls in an integration process (especially if the knowledge bases are not properly synchronized). However, our aim is to enable AORTA to be integrated with most of the existing agent platforms, and since it requires only that formulas must be converted between the language of AORTA and the agent platform in question, we find that it makes the implementation of AORTA simpler to understand.

### 3.2 Acting and coordinating

At the center of AORTA-agents are the *organization-specific actions*. While an agent will have access to a number of domain-specific actions (such as a medic performing a life-saving action), an AORTA-agent will furthermore be able to consider certain organizational options (what happens by enacting a certain role, pursuing an objective), or performing organizational actions (enacting a role, committing to an objective).

**Definition 4 (Organization-specific actions).** *The set of options with typical element $a_O$ is denoted Opt and the set of actions with typical element $a_A$ is denoted Act.*

$$a_O ::= \mathit{consider}(\phi) \mid \mathit{disregard}(\phi)$$
$$a_A ::= \mathit{enact}(\rho) \mid \mathit{deact}(\rho) \mid \mathit{commit}(\phi) \mid \mathit{drop}(\phi)$$

Actions are executed using a transition function, $\mathcal{T}_O$ and $\mathcal{T}_A$, respectively. Each action is only applicable in certain states. For example, $\mathsf{consider}(\phi)$ can only be applied if $\Sigma_o \models \phi$ in the current state, and the effect is that $\phi$ is added to $\Gamma_o$. Role enactment, $\mathsf{enact}(\rho)$, is applicable only when $\rho$ is the name of a role, the agent does not currently enact that role. Committing to an objective, $\mathsf{commit}(\phi)$, is possible only if $\phi$ is an organizational objective, and $\phi$ is not already a belief or a goal[3]. $\mathsf{disregard}(\phi)$, $\mathsf{deact}(\rho)$ and $\mathsf{drop}(\phi)$ simply remove the respective formula from the appropriate knowledge base.

Notice the correspondence between elements in Opt and Act: if the agent considers enacting a role, the enact action allows it to enact that role. However, once the role is enacted, the option is no longer an option. Since the agent now enacts the role, it seems appropriate to remove the option from $\Gamma_o$. This is done using an *option removal function*, $O$, which removes options, when they are no longer applicable (that is, when their respective organizational action would be undefined).

We are now in a position to introduce *organizational reasoning rules*: option and action rules. These rules enable the agent to decide which organization-specific actions to perform.

**Definition 5 (Reasoning rules).** *The sets of option rules $\mathcal{R}_O$ and action rules $\mathcal{R}_A$ are defined as follows.*

$$\mathcal{R}_O = \{\rho \implies a_O \mid \rho \in \mathcal{L}_R, a_O \in \mathit{Opt}\}$$
$$\mathcal{R}_A = \{\rho \implies a_A \mid \rho \in \mathcal{L}_R, a_A \in \mathit{Act}\}$$

Finally, since each agent has its own organizational state, they need to be able to coordinate and synchronize their organizational knowledge. While such

[3] The correspondence between goals and beliefs is based on achievement goals in the GOAL agent programming language [7], which are defined such that $\phi$ is an achievement goal iff $\phi$ is a goal and $\phi$ is not currently believed.

coordination can happen in different ways, we choose to use *organizational messages*. In order to determine whether a message is intended for AORTA, organizational messages are wrapped in an organizational wrapper, om, which is an unary predicate with the message as a single term.

**Definition 6 (Organizational Messages).** *An organizational message is defined as*

$$\text{msg}(\alpha, \mathsf{om}(M),$$

*where* om *is the organizational wrapper, and $M$ is the message. In* outgoing *messages, $\alpha$ corresponds to the set of recipient agents, and in* incoming *messages, $\alpha$ is the sender.*

Each agent can then specify how to coordinate using a set of *coordination rules*, which specifies certain criteria for *when* and with *whom* to coordinate.

**Definition 7 (Coordination rules).** *A coordination rule is a triple,*

$$(c, \phi, m),$$

*where $c$ is the trigger for coordination and is a set of positive or negative reasoning formulas, $\phi$ defines the set of agents to coordinate with, and $m$ is the message.*

The coordination trigger $c$ can, e.g., be the set $\{\mathsf{bel}(\textit{injuredFound})\}$, which will trigger at a point where $\Sigma_a \models \textit{injuredFound}$ is true and $\Sigma_a \models \neg\textit{injuredFound}$ was true in the previous state.

### 3.3 AORTA reasoning cycle

The configuration of an AORTA-agent consists of the agent's knowledge bases, a number of option, action and coordination rules, and a message box for incoming (inbox) and outgoing (outbox) organizational messages. The initial state consists of a set of initial beliefs and goals, and the organizational specification.

The agent has a number of *state transition rules* available, which can be used to change its state. A reasoning cycle in AORTA is executed using a *strategy* that decides which transition rules to execute.

The agent has transition rules for execution of option and action rules, called Opt and Act, a transition rule for external updates, Ext, and two rules for coordination, Coord and Chk.

Opt can be applied to an option rule in a given state, $\rho \implies a_O$, if $\rho$ is entailed and the option transition function, $\mathcal{T}_O$, is defined for $a_O$.

Act works similarly for action rules, using the action transition function, $\mathcal{T}_A$, and the option removal function, $O$.

Ext changes the agent's mental state to accommodate updates from outside AORTA. For example, if the agent perceives something, Ext adds the percept to the belief base.

```
options {
  [org(role(R,Os)), bel(me(Me), member(O,Os), cap(O))] => consider(role(Role,Os))
  [bel(me(Me)), org(role(R,Os), rea(Me,R), member(O,Os), objective(O), active(O))]
                                                        => consider(objective(O))
}
actions {
  [opt(role(Role,_))] => enact(R)
  [opt(objective(O)), org(role(R,Os), member(O,Os), rea(Me,R)), bel(me(Me))] => commit(O)
}
coordination {
  [+bel(visited(R))] : [org(rea(A,medic))] => send(A,bel(visited(R)))
  [+goal(X)] : [bel(me(Me)), org(rea(Me,R1), dependency(R1,R2,X), rea(A,R2))]
                                                                  => send(A, goal(X))
  [+bel(O)] : [org(role(R,Os), objective(O), member(O,Os), rea(A,R))] => send(A, bel(O))
  [+org(rea(A,R))] : [bel(agent(Ag))] => send(Ag, org(rea(A,R)))
}
```

**Fig. 4.** An example of an AORTA program.

Coord is applied to coordination rules, $(c, \phi, m)$, when $c$ is triggered by the state, and the set of agents entailed by $\phi$ is not empty. The message $m$ is then sent to each agent.

Chk takes new messages from the incoming message queue and adds them to the appropriate knowledge base[4].

For the purpose of this paper, we use a single linear strategy, which executes the state transition rules in a predefined order.

**Definition 8 (Linear strategy).** *The linear strategy is defined as follows:*

$$(\textsc{Chk})^*(\textsc{Ext})(\textsc{Opt})(\textsc{Act})(\textsc{Coord})^*,$$

*where* $(\textsc{Rule})^*$ *denotes that* Rule *is executed until the agent's state no longer changes.*

The strategy executes each of the transition rules, as explained above, changing the agent's state. The linear strategy is rather simple, but it is possible to implement strategies, which e.g. allows the agent to explore different paths before choosing one.

### 3.4 AORTA programs

An AORTA program consists of three sections: *options*, *actions* and *coordination*. An example program, which can be used in the first responders scenario, is shown in figure 4.

Options and actions are of the form $[\phi]$ `=>` $a$, where $[\phi]$ consists of a comma-separated list of reasoning formulas. The content of each reasoning formula (i.e., the query) is Prolog code. For example, the action rule

```
[opt(role(R,_))] => enact(R),
```

[4] For simplicity, we assume that the agents will not consider whether a sender is trustworthy, and thus whether a message is reliable.

states that if role$(R, _)$ is an option (i.e. entailed by $\Gamma_o$), the agent should enact $R$. Note that this is a simplification of the reasoning process required by agents to decide whether or not to enact a role in an organization. It is, however, possible to incorporate more sophisticated reasoning, e.g., by using the notion of social power. For example, in [3], various forms of power agents may have over each other are identified and formalized as rules. These power relations can be used in the reasoning process by adding the rules to the agents' organizational state.

The coordination section consists of coordination triples, of the form $[c]$ : $[\phi]$ => send$(Ag, \psi)$, where $c$ is a list of reasoning formulas, with either + or - in front of each, denoting that the trigger or its negation is now entailed by the agent's mental state. $\phi$ is identical to $\phi$ in option and action rules. $Ag$ corresponds to a variable in $\phi$ or $c$, and $\psi$ is the message to be sent. Thus, the following rule

```
[+org(rea(A,R))] : [bel(me(A),agent(Ag))] => send(Ag, org(rea(A,R)))
```

states that when the agent enacts a role, it should inform all other agents in the system.

The implementation of Opt and Act is deterministic: the rules in each section are simply processed linearly, and the first matching rule is executed. Coord is implemented such that every triggered triple in a state will be executed in a single step.

### 3.5 Implementation overview

The architecture is depicted in figure 5. The system is implemented in the class `Aorta`, which contains a list of the agents in the system and a reference to the original organizational specification. Each AORTA-agent is associated with an instance of `AortaAgent`, which contains the agent's state, `AgentState`, and in which the reasoning cycle is implemented. The reasoning cycle performs two steps: executing the strategy and sending messages from the outbox.

### 3.6 Integration considerations

The agent state contains the agent's the knowledge bases, rules and message boxes. Furthermore, it contains an `ExternalAgent` and an `AortaBridge`. The external agent corresponds to the message box and knowledge bases of the agent using AORTA. That is, whenever the agent commits to a new goal or updates its beliefs, these changes are propagated via the external agent into AORTA using Ext. The bridge lets AORTA manipulate the agent's mental state. For example, successful execution of commit$(\phi)$ will add $\phi$ to the agent's goal base using the bridge.

When integrating AORTA into an existing agent platform, there are thus three things to take care of.

**Bridge** AORTA uses the bridge to send updates to the agent's goal and belief bases, so an agent platform-specific bridge should be implemented (by implementing the `AortaBridge` interface), such that the knowledge bases can be synchronized.

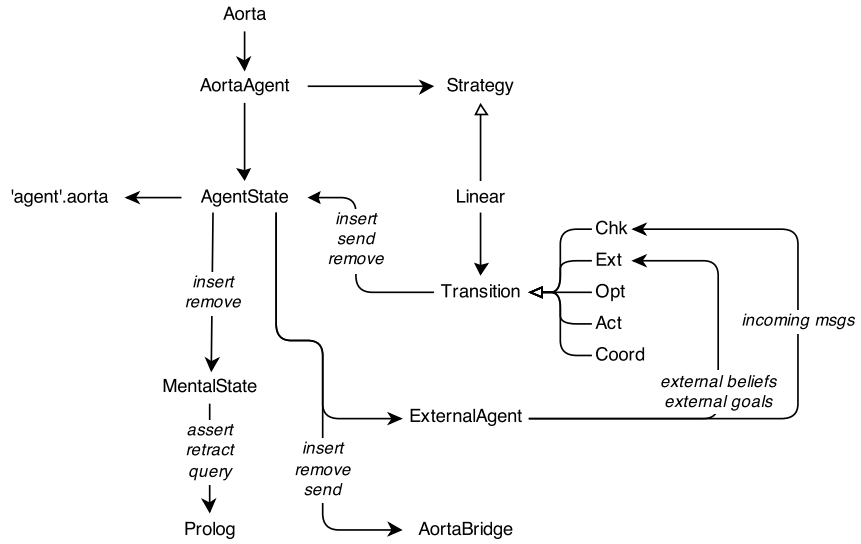


**Fig. 5.** Implementation overview with the most important classes. A filled arrowhead indicates an association between classes. An unfilled arrowhead indicates inheritance.

**External agent** When the agent updates its goal or belief base, it should inform AORTA by invoking the appropriate methods of `ExternalAgent`.

**Translation** AORTA makes use of tuProlog, so the contents of the agent's knowledge bases should be translated into Java objects supported by tuProlog.

## 4 *Jason*+AORTA

We now show how AORTA can be implemented in an existing agent platform, the *Jason* platform [1]. *Jason* is a Java-based interpreter for an extended version of AgentSpeak. *Jason* is based on the beliefs-desires-intentions (BDI) model, is open source and highly extensible, making it a reasonable choice for the integration of AORTA.

The AgentSpeak language is a Prolog-like logic programming language, which allows the developer to create a plan library for each agent in a system. A plan in AgentSpeak is of the form

```
+triggering event : context <- body.
```

If an event matches a trigger, the context is matched with the current state of the agent. If the context matches the current state, the body is executed; otherwise the engine continues to match contexts of other plans with the same trigger. If no plan is applicable, the event fails. Triggering events can amongst other things be addition or deletion of beliefs ($+l$ and $-l$) and addition or deletion of goals

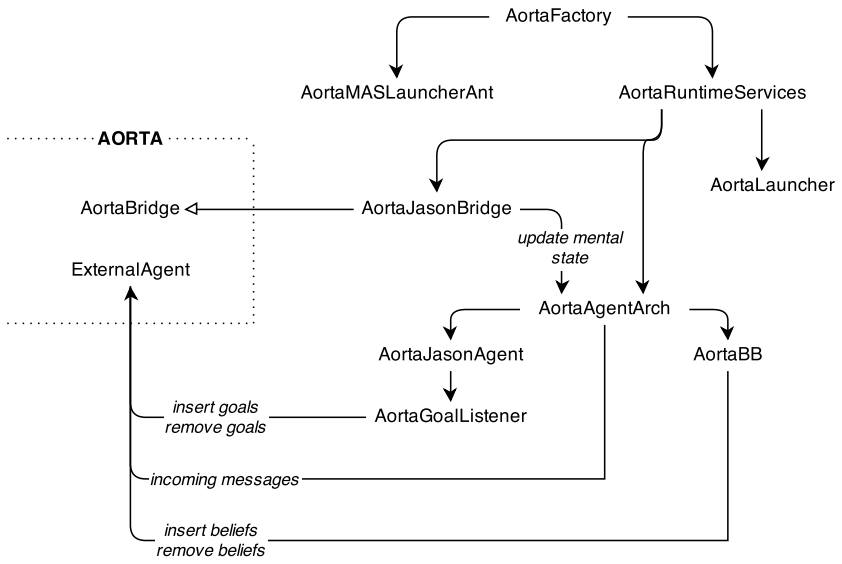


**Fig. 6.** *Jason*+AORTA. A filled arrowhead indicates an association between classes. An unfilled arrowhead indicates inheritance.

(`+!`$l$ and `-!`$l$). The body contains a sequence of actions the agent should perform and goals to adopt. When adopting a goal in the body of a plan, the agent will attempt to achieve the new goal before continuing executing the current plan.

Note that when a plan for a goal has been completed, the goal is considered finished. This means that it will be removed from the agent's mental state. Since commit($\phi$) is only defined if $\phi$ is not already a goal and is not believed by the agent, the agent will be able to commit to a goal multiple times, until it believes it has been achieved.

The AORTA integration in *Jason* is shown in figure 6. The integration consists of an extended agent architecture, which implements the actual integration with AORTA, and an infrastructure, which makes it possible to create an AORTA-project in *Jason* without having to deal with the specifics of the integration. This is done by specifying the infrastructure as follows:

```
MAS projectname {
   infrastructure: AORTA(organization(location, type))
   ...
}
```

The infrastructure takes two parameters: `location` refers to the location of the organizational specification, and `type` refers to the type of organizational model (currently, the a generic organization based on the metamodel is supported).

AORTA does not make any changes to the *Jason* language, and any existing implementations of multi-agent systems in *Jason* should be compatible with *Jason*+AORTA. The integration does two things: (1) when the belief base or goal base of the AORTA-agent changes, these changes are propagated to the *Jason*-agent (via `AortaJasonBridge`), and an addition/deletion event is triggered and (2) when the *Jason*-agent's mental state changes, AORTA receives those changes (via the `ExternalAgent`). The *Jason*-agent is connected to the `ExternalAgent` in three places:

`AortaAgentArch` Organizational messages are filtered and sent to AORTA for processing. The normal procedure for checking an agent's mailbox is extended to check whether incoming messages are wrapped in the organizational wrapper.

`AortaBB` Whenever the *Jason*-agent's belief base is changed (i.e., a belief is added or removed), those beliefs are sent to AORTA to ensure synchrony between the mental states.

`AortaGoalListener` When a goal changes state (i.e., when a plan for it has started, failed, or stopped), the goal listener is responsible for sending the changes to AORTA.

Furthermore, *Jason* formulas are converted to AORTA formulas. Note that while *Jason* supports annotations on literals (e.g., denoting the source of a belief, `injuredFound[source(alice)]`), they are lost in conversion to AORTA formulas, since they are not supported. This should generally not be a problem, since formulas will not propagate back and forth between the systems. That is, if a belief originates from *Jason*, it will be sent to AORTA, which will not send it back to *Jason*, e.g. `+injuredFound[source(alice)]` $\rightarrow$ `bel(injuredFound)` $\rightarrow$ `+injuredFound` does not happen.

The AORTA reasoning cycle is executed in *Jason* via the method `reasoningCycleStarted()` in `AortaAgentArch`, which is called in the beginning of a *Jason* reasoning cycle. This means that the agent will execute the AORTA reasoning strategy in the beginning of each cycle.

### 4.1 The first responders scenario

We now discuss how AORTA can be used to let agents participate in the first responders scenario. We use the Blocks World for Teams [13] testbed to simulate the first responders scenario by considering the drop zone being the ambulance, colored blocks being injured fans, and agents playing the roles of fans, medics and police officers. Fans are fighting just outside some of the rooms and they can stop the medic from rescuing injured fans by entering a room just before the medic does so. Police officers will look for areas where fans are standing, while medics will check the rooms to find injured fans.

Consider an agent, *Bob*, playing the role of a medic ($\Sigma_o \models \mathsf{rea}(bob, medic)$), using the program in figure 4. He is considering the objective *injuredFound* ($\Gamma_o \models \mathsf{objective}(injuredFound)$), to which he has not yet committed. The following action rule can then be executed.

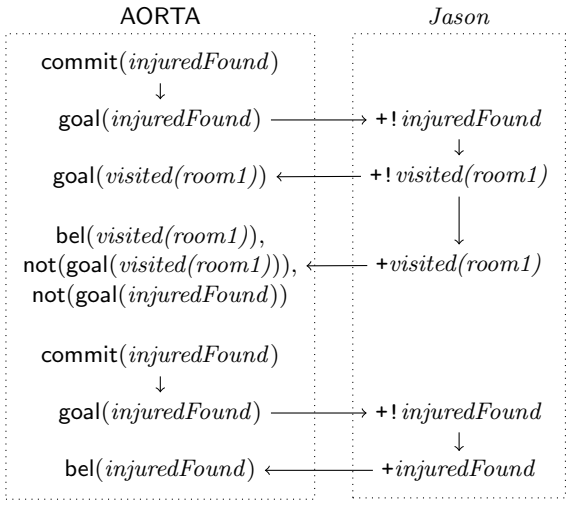


**Fig. 7.** The flow of execution starting when *Bob* performs the organizational action commit(*injuredFound*). not means that the formula is removed from the mental state.

```
[opt(objective(O)), bel(me(Me)),
 org(role(R,Os), member(O,Os), rea(Me,R))] => commit(O)
```

In the resulting state, *injuredFound* is added as a goal ($\Gamma_a \models$ *injuredFound*), and is sent via the bridge to the *Jason*-agent. This will trigger an event, +! *injuredFound*, and if the agent has a plan matching this trigger, it will execute the body of the plan. *Bob* has the following simplified plan library, making him capable of searching for injured fans.

```
+!injuredFound : room(R) & not(visited(R)) <- !visited(R).
+!injuredFound <- +injuredFound.

+!visited(R) : in(R) <- +visited(R).
+!visited(R) : not(state(traveling)) <- goTo(R); !visited(R).
```

*Bob* is situated in an environment with a single room, *room1*. The flow of the execution is shown in figure 7. *Bob* commits to finding the injured, which leads to the subgoal of visiting *room1*. When he believes he has visited the room (when he is inside the room), both goals will finish, since ! *injuredFound* waited on the completion of ! *visited(room1)*. Since the main goal, *injuredFound*, has not yet been completed, *Bob* can execute the same action rule again, thus committing to the goal once more. Since there are no more rooms to visit, only the second plan is applicable, and he believes that all the injured fans have been found.

When *injuredFound* is achieved, several things happen. First, the following coordination mechanism is triggered:

```
[+bel(O)]
    : [org(role(R,Os), objective(O), member(O,Os), rea(A,R))]
    => send(A, bel(O))
```

Since bel(*injuredFound*) is added to the agent's mental state, and *injuredFound* is an objective, the agent will inform all agents responsible for that objective, that it has been completed. Second, the next objective, *injuredSaved*, becomes an option, and *Bob* will then commit to completing it. The flow of execution is similar to that of figure 7 and will not be described in detail.

If, during the rescue, a room is blocked by a fan, the agent may adopt a goal, *removeBlocker*, which will trigger the following coordination mechanism:

```
[+goal(X)]
    : [bel(me(Me)), org(rea(Me,R1), dependency(R1,R2,X), rea(A,R2))]
    => send(A, goal(X))
```

Since the agent commits to a goal for which there is a dependency, he sends a request to the agents enacting the role `R2` (in this case the *officer* role). An officer should then commit to achieving the goal, and inform the medic when it has been done.

## 5 Related work

The $\mathcal{M}$OISE$^+$ model is based on three organizational *dimensions*: the structural, functional and deontic dimensions [9]. Development of organized multi-agent systems using the $\mathcal{M}$OISE$^+$ model is separated into a system and an agent level. The system level, $\mathcal{S}$-$\mathcal{M}$OISE$^+$, provides an interface (a middleware) between the agents and the organization using a special agent, the OrgManager, to change the organizational state, ensuring organizational consistency . The agent level, $\mathcal{J}$-$\mathcal{M}$OISE$^+$, joins *Jason* and $\mathcal{M}$OISE$^+$, by making organizational actions available to agents, such that they can reason about (and change, using the OrgManager) an organization.

Similar to AORTA-agents, agents in $\mathcal{J}$-$\mathcal{M}$OISE$^+$ receive objectives (missions) that they can achieve using Jason plans. The main difference is that the organization-oriented reasoning is done as a part of the agent's normal reasoning process, whereas AORTA-agents perform the organizational reasoning inside AORTA, and then decides how to complete their objectives at a different level. The main advantage of keeping the reasoning apart in AORTA is that it allows agents on different agent platforms to perform the same kind of organizational reasoning without any extra development required.

The ORA4MAS (Organizational Artifacts for Multi-Agent Systems) approach [8] is another attempt to build a bridge between an organization and the agents in it. It is a general approach suitable for different kinds of organizational models. They use artifacts, which they claim brings the control back to the agents (as compared to using a middleware), since the agents can, via their autonomy, choose whether to interact with the organizational artifacts of the system.

We argue that the ultimate way of bringing the control back to the agents is to allow the agents *themselves* to perform the organizational reasoning. By integrating AORTA in agents, they are provided with organizational reasoning capabilities, while they are still able to, e.g., decide not to commit to certain organizational objectives.

## 6 Conclusion and future work

We have described the AORTA architecture and have shown how it can be integrated in the *Jason* platform. The example shows how *Jason*-agents gain capabilities to reason about which organizational objectives to commit to, and how to coordinate completing them.

AORTA lets the developer focus on implementing the agents' domain-specific capabilities, while commitment to organizational objectives, coordination, and communication can be done entirely by AORTA. Furthermore, since AORTA can be integrated in different agent platforms, the same AORTA programs can be used for several different implementations in different agent programming languages. The use of the simple, generic language makes AORTA readily useful, however, the support of an existing, and more powerful, organizational language, such as $\mathcal{M}$OISE$^+$ or OperA, is a natural extension to the architecture.

The decoupling of AORTA and the agent platform means that synchronization is required. However, the linear strategy makes sure that external changes are synchronized before options and actions are considered (via the EXT transition rule). As mentioned, the requirement is a translation between AORTA formulas and the formulas of the connected agent (e.g. AgentSpeak formulas). Furthermore, organizational reasoning is done in AORTA and is thus separated from the agent's normal reasoning. This is because the organizational state is only available to AORTA, as it is not shared with the agent. This means that the agent cannot reason about organizational matters, such as role enactment and organizational objectives without using the rules of AORTA. However, if necessary, in the case of *Jason*, it is possible to allow this kind of reasoning by introducing an *internal action*, e.g. `.org(Fml)` which succeeds if `Fml` can be translated to an AORTA formula and is entailed by the organizational state.

In the future, we plan to investigate other strategies that could improve the reasoning, such as a strategy that explores different paths of execution, and makes a decision based on this. Furthermore, since agents may have objectives that do not coincide with the organizational objectives, they need a way to decide which objectives to pursue, for example using a preference ordering [2] or individual agent preferences [10].

Finally, we are investigating how to incorporate norms in the semantics, such that the agents are able to deliberately follow paths that violate the organization, while possibly being sanctioned by other agents in the organization.

## References


1. Bordini, R.H., Hübner, J.F., Wooldridge, M.: Programming multi-agent systems in AgentSpeak using Jason. John Wiley & Sons (2007)
2. Broersen, J., Dastani, M., Hulstijn, J., Huang, Z., van der Torre, L.: The BOID architecture: conflicts between beliefs, obligations, intentions and desires. Proceedings of the fifth international conference on autonomous agents (2001) 9–16
3. Carabelea, C., Boissier, O., Castelfranchi, C.: Using Social Power to Enable Agents to Reason About Being Part of a Group. In: Engineering Societies in the Agents World V. (2005) 166–177

4. Denti, E., Omicini, A., Ricci, A.: tuProlog: A light-weight Prolog for Internet applications and infrastructures. Practical Aspects of Declarative Languages (2001) 184–198
5. Dignum, V.: A model for organizational interaction: based on agents, founded in logic. PhD thesis, Utrecht University (2004)
6. Esteva, M., de la Cruz, D., Sierra, C.: Islander: An electronic institutions editor. In: Proc. AAMAS '02. (2002)
7. Hindriks, K.V.: Programming Rational Agents in GOAL. Multi-Agent Programming: Languages, Tools and Applications (2009) 119–157
8. Hübner, J.F., Boissier, O., Kitio, R., Ricci, A.: Instrumenting multi-agent organisations with organisational artifacts and agents. Autonomous Agents and Multi-Agent Systems **20**(3) (April 2009) 369–400
9. Hübner, J.F., Sichman, J.S., Boissier, O.: Developing organised multiagent systems using the MOISE+ model: programming issues at the system and agent levels. International Journal of Agent-Oriented Software Engineering **1**(3) (2007) 370–395
10. Jensen, A.S.: Deciding between conflicting influences. In Cossentino, M., Fallah Seghrouchni, A., Winikoff, M., eds.: Engineering Multi-Agent Systems. Volume 8245 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2013) 137–155
11. Jensen, A.S., Aldewereld, H., Dignum, V.: Dimensions of organizational coordination. In: Proceedings of the 25th Benelux Conference on Artificial Intelligence, Delft University of Technology (2013) 80–87
12. Jensen, A.S., Dignum, V.: AORTA: Adding Organizational Reasoning to Agents. In: Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems. (2014) to appear
13. Johnson, M., Jonker, C., van Riemsdijk, M.B., Feltovich, P.J., Bradshaw, J.M.: Joint activity testbed: Blocks world for teams (BW4T). In: Proceedings of the 10th International Workshop on Engineering Societies in the Agents World. (2009) 254–256
14. van Riemsdijk, M.B., Hindriks, K.V., Jonker, C.M.: Programming organization-aware agents. Proceedings of the 10th International Workshop on Engineering Societies in the Agents World (2009) 98–112