Angerona - A flexible Multiagent Framework for Knowledge-based Agents

Patrick Krümpelmann, Tim Janus, and Gabriele Kern-Isberner

Technische Universität Dortmund

Abstract. We present the Angerona framework for the implementation of knowledge-based agents with a strong focus on flexibility, extensibility, and compatibility with diverse knowledge representation formalisms. As the basis for this framework we propose and formalize a general concept of *compound agents* in which we consider agents to consist of hierarchies of interacting epistemic and functional components. Each epistemic component is instantiated by a knowledge representation formalism. Different knowledge representation formalisms can be used within one agent and different agents in the same system can be based on different agent architectures and can use different knowledge representation formalisms. Partially instantiations define sub-frameworks for, e. g., the development of BDI agents and variants thereof. The Angerona framework realizes this concept by means of a flexible *JAVA* plug-in architecture for the epistemic and the functional components of an agent. The epistemic plug-ins are based on the Tweety *library* for knowledge representation, which provides various ready-for-use implementations and knowledge representation formalisms and a framework for the implementation of additional ones. Angerona already contains several partial and complete instantiations that implement several approaches. Angerona also features an environment plug-in for communicating agents and a flexible GUI to monitor the multiagent system and the inner workings of the agents, particularly the inspection of the dynamics of their epistemic states. Angerona and Tweety are ready to use, well documented, and open source.

1 Introduction

A variety of logical formalisms with different expressivity and computational properties have been developed for knowledge representation with the agent paradigm in mind [1,2]. Especially non-monotonic formalisms are designed to deal with incomplete information and to enable an agent to act in uncertain environments. Moreover, the field of research on belief change has been working for over 25 years already on solutions on how to change an agent's beliefs in the light of new information [3]. Yet, very little of the approaches developed in these two fields of research are available in actual multiagent frameworks.

Our concept of component based agents and its realisation in the Angerona framework are designed to reduce this gap, to support the development of knowledge based, i. e. epistemic, agents based on logical formalisms for knowledge

representation and reasoning, and to support the use of belief change operators based on belief change theory. Moreover, it facilitates the development of divers agents with respect to their architecture and knowledge representation. It allows the formation of multiagent systems comprising heterogeneous agents which interact by communicating or in a common simulated environment.

The ANGERONA framework is based on the conceptual work on a hierarchical component-based agent model. In this, an agent comprises an epistemic state and a functional component that can both be composed. This model is realized on the basis on a plug-in architecture. These are based on knowledge representation plug-ins and on operator plug-ins respectively, and tied together by an *XML* based script language and configuration files. The plug-in for the functional component is based on the general ANGERONA operator interface and the corresponding operators provided by the angerona framework. For the knowledge representation plug-ins we integrated the TWEETY *library* for knowledge representation [4]. The TWEETY library contains interfaces and implementations for divers knowledge representation formalisms and inference and change operators for them. TWEETY is under active development, in which we participate. It currently contains implementations for: first-order logic, ordinal conditional functions, relational conditional logic, probabilistic conditional logic, relational probabilistic conditional logic, markov logic, epistemic logic, description logic, deductive argumentation, structured argumentation frameworks, defeasible logic programming, probabilistic argumentation, answer set programming. Confer to [4] for details and references. ANGERONA is open source and available on *github*[1], as is TWEETY on *sourceforge*[2].

The ANGERONA agent architecture can be freely defined by specifying the types of operators to be used and their order of execution. This way ANGERONA allows to easily design different types of agents. Not only the used language for knowledge representation can differ, but also to which amount an agent's functionality is logic based. It is, for instance, easily possible to realize the agent's deliberation and means-ends reasoning by JAVA operators and simple data components, or by simple JAVA operators which make use of logical formalisms, e. g. answer set programming (ASP) [5], ordinal conditional functions (OCF) [6], argumentation formalisms [7], or propositional logic or horn logic, or any other formalism from the TWEETY library.

While the general ANGERONA framework allows for a high degree of flexibility it also allows to define partially instantiated plug-ins and default agent configurations, which represent sub-frameworks with more predefined structure and functionality. The latter might fix the general agent cycle by specifying the types of operators to be used and provide different implementations for these. Hence, the sub-frameworks provide more support for easy and rapid development of agents. We distinguish three different types of users in the ANGERONA framework: the *core developer* that uses ANGERONA as a toolkit to define its own agent types; the *plug-in developer* that uses provided agent types and in-

[1] `https://github.com/Angerona`
[2] `http://sourceforge.net/projects/tweety/`

stantiates them with given or its own plug-ins; and the *knowledge engineer* that defines the background and initial knowledge, and all other initial instances of the components of the agents.

ANGERONA provides default implementations for BDI style agents and diverse extensions that can be modularly used to build agents. Complete multi-agent systems of communicating agents using answer set programming, propositional logic and ordinal conditional functions for knowledge representation, including change operators for these based on belief change theory are implemented and available. These are used in the context of secrecy preserving agents for which scenarios and simulations are available. ANGERONA also features a plug-in interface for different environments, with a communication environment for agents implemented. A graphical user interface (GUI) allows the selection, execution, observation, and inspection of multi-agent simulations. The GUI can be extended by plug-ins to feature displays of specific knowledge representation formalisms, for instance dependency graphs.

In the next section we introduce the concept of compound agents that underlies the ANGERONA framework. Following on this we describe how this is realized in the agent framework of ANGERONA. Then we describe the multiagent framework of ANGERONA in the following section. Afterwards, we briefly describe how we used ANGERONA to build secrecy preserving agents. Finally, we discuss our framework and its relation to other frameworks, and we conclude.

2 Concept of Compound Agents

ANGERONA agents are based on a concept of hierarchical, component-based agent models with the goal of capturing a variety of agent architectures in a flexible and extensible way. In the following we give an overview of the main concepts of it. In this, a general *agent* instance is a tuple $(\mathcal{K}, \xi)$ comprising of an epistemic state $\mathcal{K} \in \mathcal{L}_{ES}$ from a given language $\mathcal{L}_{ES}$ and a functional component $\xi = (\circ, \mathsf{act})$. Further, we assume the set of possible actions Act and perceptions Per to be given. These might, for instance, be speech acts that are interchanged by the agents. Then, we require the operators of the functional component to be of the following types:

$$\circ : Per \times \mathcal{L}_{ES} \to \mathcal{L}_{ES} \text{ and } \mathsf{act} : \mathcal{L}_{ES} \to Act.$$

The language of the epistemic state might be a logical language, e. g. an answer set program or a conditional belief base, or a Cartesian product of (logical) languages, e. g. to represent the BDI components of an agent by the language $\mathcal{L}_B \times \mathcal{L}_D \times \mathcal{L}_I$. The epistemic state of an agent contains representations of its background knowledge about how the world works, and information coming from its perceptions, as well as its goals and know-how, and potentially more. The functional component of an agent consists of a change operator $\circ$, which adapts the current epistemic state of the agent upon reception of a perception, and an action operator act, which executes the next action based on the current epistemic state. The change of the epistemic state might involve different types of

reasoning, such as non-monotonic reasoning, deliberation and means-ends reasoning. These are partially or completely based on logical inference. This means, that an agent's behavior is realized in parts by the functional component, and in parts by the knowledge representation and reasoning based on the epistemic state. How much of the agents behavior is defined by the epistemic state and how much by the functional component might differ largely; a pure deductive agent's behavior is entirely defined by its epistemic state, and a stateless agent entirely by its functional component. To capture these different types of agents we consider the epistemic state as well as the functional component to consist of hierarchical components. Compositions thereof define more structured agent models and can be further refined.

A compound epistemic state is a component, which again can either be atomic or compound. An atomic component $\mathcal{C}_a$ is an element from the components language $\mathcal{L}_{\mathcal{C}_a}$, e. g. a belief base BB from the language $\mathcal{P}(\mathcal{L}_{BB})$, such as an OCF-base or an answer set program. Belief operators of the form $Bel : \mathcal{P}(\mathcal{L}_{BB}) \rightarrow \mathcal{P}(\mathcal{L}_{BS})$ are applied by other operators to belief bases to determine the current belief set for it. For example, the (sceptical) ASP belief operator $Bel : \mathcal{P}(\mathcal{L}_{At}^{asp}) \rightarrow Lit$ is defined as $Bel(P) = \cap AS(P)$, with $AS(P)$ being the answer sets of P. Other operators for ASP might make use of preferences or might be defined for sequences of logic programs.

A compound component is a tuple of components, $\mathcal{C} = \langle \mathcal{C}_1, \ldots, \mathcal{C}_n \rangle$, and each component is an element of its language such that the language of a compound component is a cartesian product of languages: $\mathcal{L}_{\mathcal{C}} = \mathcal{L}_{\mathcal{C}_1} \times \cdots \times \mathcal{L}_{\mathcal{C}_n}$. In particular, each component can potentially have a different representation. The interaction of the components is realized by the functional component of the agent. In particular, for an epistemic state $\mathcal{K} \in \mathcal{L}_{ES}$ and functional component $\xi = (\circ, \mathsf{act})$ the change operator $\circ : Per \times \mathcal{L}_{ES} \rightarrow \mathcal{L}_{ES}$ can be realized by a single function or by a composition of explicit sub-functions. In the latter case sub-functions are applied to the epistemic state in sequential order. Each sub-function modifies a single component or a set of components of the epistemic state. The next function operates on the epistemic state that results from the modifications of the previous functions. This concept realizes the idea of an agent cycle. Typical agent cycles as the one of the BDI architecture can be easily formalized. For example, first the beliefs of the agent are modified given a perception by some function, then another function modifies the goals of the agent and then yet another function modifies the current plan of the agent, or revises the agent's plan library. Inner loops or concurrent execution of operators can be modeled by single operators which contain loops or concurrency. The resulting hierarchical agent model with compound epistemic state and compound functional component is illustrated in Figure 1. Formally, a compound functional component consists of a change operator $\circ$ that is a composition of operators, i. e. $\circ =_{def} \circ_1 \cdot \ldots \cdot \circ_{n'}$, and an action function act.

We exemplify this model by showing how a basic BDI agent model can be realized. A *BDI agent* is a tuple $(\mathcal{K}_{BDI}, \xi_{BDI})$. The epistemic state is of the form $\mathcal{K}_{BDI} = \langle \mathcal{B}, \Delta, \mathcal{I} \rangle$ with the agent's beliefs $\mathcal{B} \subseteq \mathcal{L}_{\mathcal{B}}$, a set of desires $\Delta \subseteq$

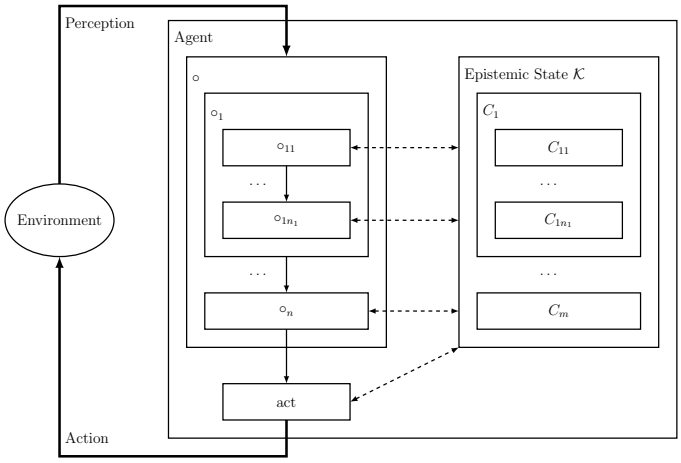

Fig. 1. Hierarchical agent model

$\mathcal{L}_\Delta$ and a set of intentions $\mathcal{I} \subseteq \mathcal{L}_\mathcal{I}$, all of which might be belief bases of a knowledge representation formalism. We define the language of BDI epistemic states as $\mathcal{L}_{BDI}$. A functional BDI component $\xi_{BDI} = (\circ, \mathsf{act})$ consists of a change operator $\circ$ and an action operator act of the type $\circ : Per \times \mathcal{L}_{BDI} \to \mathcal{L}_{BDI}$ and $\mathsf{act} : \mathcal{L}_{BDI} \to Act$. The change operation can then be represented as $\circ_{BDI} =_{def} \circ_B \cdot \circ_\Delta \cdot \circ_\mathcal{I}$. That is, an *BDI-epistemic-state* $\mathcal{K}_{BDI} = \langle \mathcal{B}, \Delta, \mathcal{I} \rangle$ is changed by a perception p such that $\mathcal{K}_{BDI} \circ_{BDI} p = \circ_\mathcal{I}(\circ_\Delta(\mathcal{K}_{BDI} \circ_\mathcal{B} p))$. More details about the concept of compound agents can be found in [8], here we continue with the presentation of its realization in the ANGERONA framework.

3 Agent Framework

ANGERONA agents consist of *agent components* which can be *epistemic components*, i. e. belief bases and associated operators, and other data components, or *functional components*, i. e. operators used for the agent cycle. Logic based components are based on the *belief base plug-in*. Operators for the agent cycle are based on the *operator plug-in*. For the realization of plug-ins in ANGERONA we use the *Java Simple Plugin Framework (JSPF)* [9].

The class diagram in Figure 2 illustrates the realization of the conceptual model in the ANGERONA framework. An ANGERONA agent contains an epistemic state and a list of operators. An epistemic state consists of agent components. One type of agent components are belief bases which are defined via a belief base plug-in. The belief base plug-in implements the interfaces of the TWEETY library, in particular those for a belief base, a formula, a revision operator and a belief operator. Different belief operators might be available for the same formalism.

Different agents might use the same knowledge representation formalism but different belief operators, and each agent might use different belief operators in different situations. We use, for example, families of belief operators that are ordered by their credulity, e. g. skeptical reasoning vs. credulous reasoning, in the setting of secrecy preserving agents. More on this can be found in [8].

The agent cycle is realized by a sequence of operators provided by operator plug-ins. Operators in ANGERONA exist on two fundamental levels of abstraction.

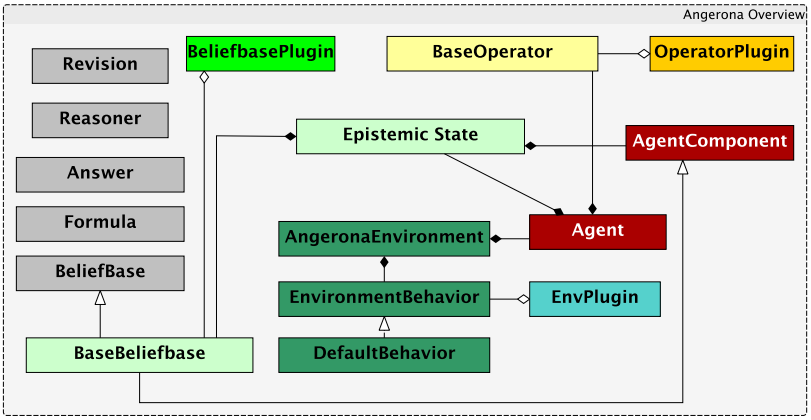

Fig. 2. ANGERONA agent simplified class diagram

Operation types represent types of operators and have three parameters, a unique name, a set of input parameters and an output type. By means of operation types we can define the agent cycle for an agent class without instantiating the concrete operators to be used. An operator is a class that implements a particular *operation type*, e. g. an ASP change operator. There might be several operators with the same operation type implemented, e. g. those of [10,11,12,13]. The *knowledge engineer* can select which operator shall be used for which of its agents in the respective agent configuration files.

The agent cycle of an ANGERONA agent is specified by means of the ANGERONA *Script Markup Language (*ASML*)*. *ASML* is an *XML* format, which features operator invocation and basic control structures to design sequences of these. It also supports access to variables in a given context, which is normally provided by the agent. For the operator invocation in *ASML* only the operator type is specified in the *ASML* file, the concrete operator instance is specified in the agent configuration file. Listing 1.1 shows a simple *ASML* example script for a BDI-style agent. The *ASML* language also features basic control structures such as assertions, conditions and loops. For a full reference of *ASML* refer to [8].

Listing 1.1. Simple BDI Agent Cycle in *ASML*

```
<asml-script name="BDICycle">
    <operation type="ChangeBeliefs">
        <param name="perception" value="$perception" />
        <param name="beliefs" value="$beliefs" />
        <output>beliefs</output>
    </operation>

    <operation type="ChangeDesires">
        <param name="beliefs" value="$beliefs" />
        <param name="desires" value="$desires" />
        <output>desires</output>
    </operation>

    <operation type="ChangeIntentions">
        <param name="desires" value="$desires" />
        <param name="intentions" value="$intentions" />
        <output>intentions</output>
    </operation>

    <execute action="$action" />
</asml-script>
```

We implemented BDI-style agents and default operators for these. We also implemented elaborate approaches for the generation and ordering of desires based on the approach to motivated BDI agents presented in [14]. Further, for the implementation of hierarchical planning for BDI agents we implemented the approach of know-how as presented in [15] by means of answer set programming as presented in [16]. Here, we focus on the presentation of the novel features of the ANGERONA framework; that is, its support and use of knowledge representation formalisms, non-monotonic reasoning and belief change theory. As explained above, a belief base plug-in provides the languages to be used and the reasoning and change operators for them. Each belief operator corresponds to the implementation of a *reasoner* of the TWEETY library.

ANGERONA provides change operators that handle incoming perceptions as illustrated in Figure 3. It determines the affected belief bases of the agent and calls the specialized change operators for each of these, provided by the specific plug-ins. Each belief base change operator uses an interpretation operator and a change operator. A perception might represent an act of communication between agents and comprise of information about the sender of the information, the addressees, a timestamp, and some logical content. The interpretation operator has to process this complex information into some sentence or set of sentences in the belief base language. The belief base is then revised by the preferred belief base change operator as specified in the agent configuration file. After all changes have been made, the agent's epistemic state gets a *changed beliefs event*, which might trigger further operations. That is, after all directly affected belief

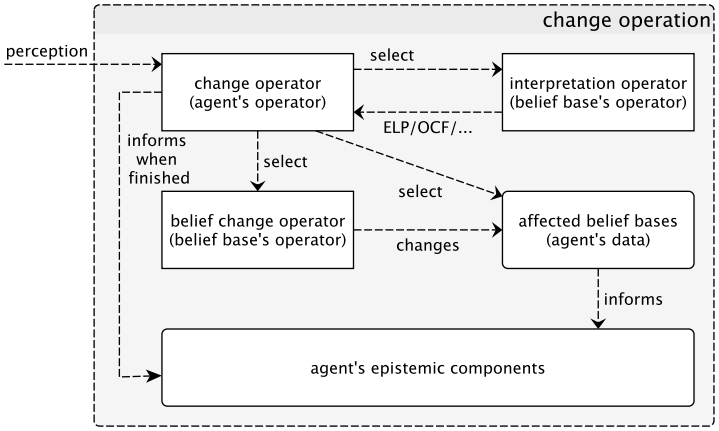

Fig. 3. Change operator in detail

bases have been changed other belief bases that are dependent on these might be changed.

We implemented general types of belief change operators in the TWEETY library. These include a selective revision operator that allow to evaluate the new information and decide if and to what extent it should be accepted [17,18]. The result of the selection operator represents the information which shall be accepted and thus be incorporated into the belief base with priority over the information in the belief base. This is exactly the task mainly studied in classic belief revision theory such that we can make use of results from that field.

Moreover, we implemented a full fledged belief base plug-in for *ASP*. It is capable of using several *ASP* solver such as *clasp*, *DLV*, *DLV-Complex*, and *smodels*. It provides parsers for the different language versions by means of *ASP*. Different belief operators and belief change operators are implemented and can be used, including those of [10,11,12,13]. Moreover, *ASP*-based version of know-how for hierarchical planning and reasoning about know-how as presented in [16] is implemented. For the visualization of ASPs extended dependency graphs and explanations graphs, based on [19], are implemented as a GUI plug-in.

4 Multiagent Framework

The multiagent framework of ANGERONA is what is commonly referred to as the *middleware* of agent programming frameworks. It organizes and starts the execution of the individual agents and the environment and implements the interaction between these. The execution order of the agents, the *multiagent cycle*, is flexible. The default is the sequential execution of agents such that each agent gets the perceptions from the previous multiagent cycle, and not those

created by the execution of agents in the same cycle. This way the order of the execution of agents does not matter.

The environment in ANGERONA is formed by the set of agents in the system and the environment behavior. The environment behavior might range from a communication infrastructure that delegates the speech acts between agents, to a simulator for physical environments. It is implemented in form of an environment plug-in which allows to use external environment simulators, or to develop new ones. The interrelation of the environment classes is shown in Figure 2. The default environment behavior of ANGERONA is a communication infrastructure based on FIPA-style, [20], speech acts. The actions of agents are speech-acts which are transmitted to the receiver agents as their perception. Since different agents might use different knowledge representation formalisms in ANGERONA a common logical language has to be determined for which each agent has an appropriate translation operator. As a language which is appropriate for agents that use such different formalisms as *ASP* and OCF we chose *nested logic programs* [21] as common language for the agents. It supports both, propositional logic and its connectives as well as conditional or rule like connectives, and default negation. However, this is only the default implementation, any other language might be used as common communication language.

ANGERONA also features a versatile graphical user interface (GUI). It is based on a docking panes approach which allows to display various aspect in different panes and tiles the entire window with these panes. The tiling can be changed individually. Panes can be grouped by means of tabs or be detached from the main window to form new windows that might be moved to a secondary screen. The plug-in architecture of ANGERONA allows for UI plug-ins which allow the

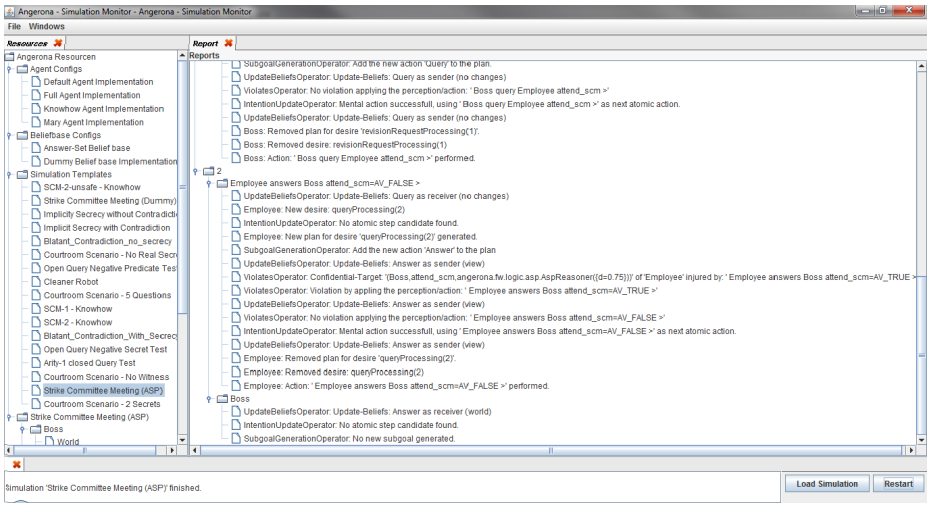

Fig. 4. Angerona GUI - Overview

development of plug-ins for the specific visualization of components stemming from plug-ins such as the representation of belief bases specific to the used formalism, potentially with alternative views such as text-based and graph-based perspectives. For example for ASP we implemented a representation based on explanation and extended dependency graphs, as presented in [19].

Another important feature of the user interface is the *report system* used in ANGERONA. The *report* defines an interface to post new *report entries* and to query the existing ones. A report entry consists of the identifier of its poster, the tick (number of multiagent cycle) and the realtime (system time) in which it was posted, a call-stack and an optional attachment in form of an epistemic component. Poster of report entries can be the agent, one of its operators or one of its epistemic components. The queries to the report then allow to construct a timeline of posts with filters based on the poster, the type of attachment and the call-stack; for instance to inspect the changes of an agent's beliefs during runtime. The report system is extensively used by the GUI to allow for the inspection on the level of an agent cycle and of a multiagent cycle. Every pane that displays the content of an epistemic component uses the report system to provide a timeline for its displayed component.

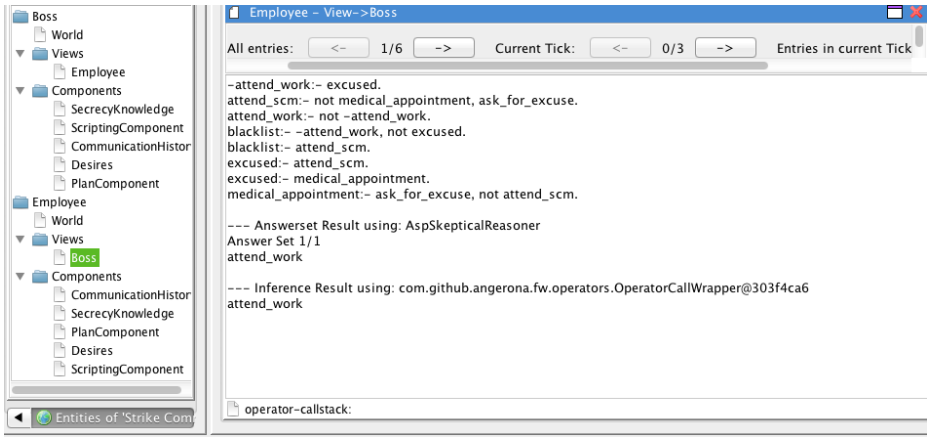

Fig. 5. Secrecy Scenario Ressources (left), *ASP* Belief Base UI Component (right)

Figure 4 shows the GUI in its start configuration after a simulation has been selected and run. The window is tiled by three docking panes, the resource pane to the left, the workspace pane to the right and the status pane at the bottom. The resources are displayed in a tree-view and are given by agent configuration files, belief base configuration files, simulations templates and resources of a loaded simulation. The resources of a simulation are typically given by its agents and their components. The workspace pane has its default tab, which views the report for the current simulation. Resources of the resource pane are opened and

displayed as an additional tab of the workspace pane by double-clicking on them. The status pane displays the current status of ANGERONA and holds buttons to load and run a simulation.

Figure 5 shows how an agent component can be selected and inspected by example of an *ASP* belief base. The logic program of the belief base is shown as well as its answer sets and the corresponding belief set that is produced by the selected belief operator. The controls at the top allow for the navigation through the timeline of the belief base given by the current report. It is shown how many entries for the belief base exist in the entire report, how many ticks the report covers, and how many entries for the belief base exist in the currently selected tick. The controls allow the navigation on the basis of these three parameters. The changes to the belief base with respect to the previous report entry for the belief base are shown by highlighting new parts in green and missing parts in red. These controls and the form of display allows to not only inspect the belief base but also to track its evolution throughout the simulation.

5 Case Study - Secrecy preserving Agents

We use the ANGERONA framework to implement and experiment with secrecy preserving agents according to the concepts presented in [22,23,24]. The following example illustrates one of the scenarios we use.

Example 1. An employee *Emma* is working in a company for her boss *Beatriz*. She wants to attend a strike committee meeting (*scm*) next week and has to ask *Beatriz* for a day off in order to attend. She knows that *Beatriz* puts everyone who attends the *scm* on her list of employees to be fired next. Thus, *Emma* wants to keep her attendance to the *scm* secret from *Beatriz*, but has to communicate with her in order to achieve her goal of getting that day off.

The intuitive formulation of our notion of secrecy preservation can be formulated as follows:

> *An agent $\mathcal{D}$ preserves secrecy if, from its point of view, none of its secrets Φ that it wants to hide from agent $\mathcal{A}$ is, from $\mathcal{D}$'s perspective, believed by $\mathcal{A}$ after any of $\mathcal{D}$'s actions (given that $\mathcal{A}$ does not believe Φ already).*

Hence, an agent has to model other agents, the information available to them and their reasoning capabilities. To implement such agents we use the architecture depicted in Figure 6. It refines the belief component of BDI agents that now consists of the agent's view on the world, its own beliefs, views on the world-views of other agents and a representation of its secrets. The belief change operator then changes all components of the beliefs. In particular, the views on the information available to other agents has to be adapted after each execution of an action. In the deliberation process an agent has to evaluate potential sub-goals with respect to secrecy. To this end we implemented a *violates* operator which performs internal change operations on copies of the belief components to determine the degrees of violation of secrets. This agent model can

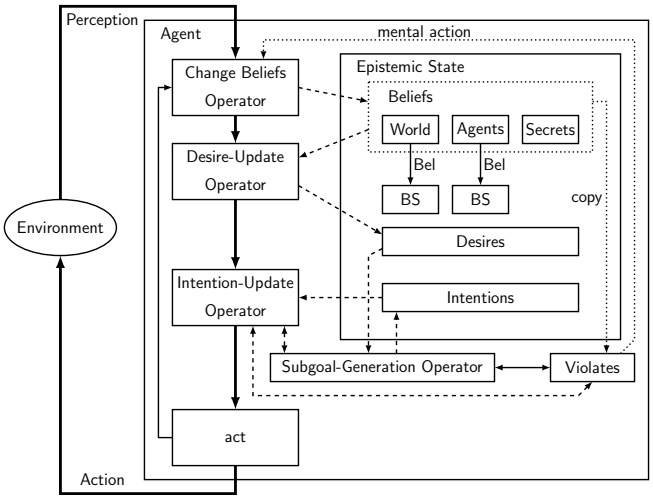

Fig. 6. Angerona BDI Secrecy Instance

then be instantiated by use of different knowledge representation formalisms. This far, we use propositional logic, ordinal conditional functions, and answer set programming to instantiate it and run simulations.

In this scenario we make use of several of Angerona's particular features. We build on the BDI sub-framework provided by Angerona and refine the composition of the epistemic state. The views on other agents are belief bases such that for each agent a different knowledge representation formalism and different belief operators can be used. Changes to the views on other agents are performed by operators from the belief base plug-in used for the respective view. We also implemented operators that change the secrets of an agent in the light of new information. We used different knowledge representation plug-ins for the agent instantiations. For the ASP instance we build on the ASP know-how [16] implementation of the intention-update operator in Angerona and extended it to take the secrets into consideration. The report system and the GUI serve well to inspect the evaluation process of actions with respect to secrecy since the internal, temporary, change operations can be observed. Figure 5 shows the resource view and an example ASP belief base view from the described secrecy scenario. For more details on the secrecy instance refer to [23,8].

6 Related Work

A plethora of multiagent programming frameworks have been proposed. Most of these are rather theoretical studies and relatively few, but still a lot, have been actually implemented and are available. A good overview of the most prominent available frameworks is given in [25]. An even more extensive list of such

frameworks can be found in [26]. These frameworks haven been build with very different goals in mind and by use of very different means. In the following we survey those coming closest to the ANGERONA frameworks main features. These are:

1. to provide a means to build agents capable of using (different) non-monotonic knowledge representation and reasoning techniques,
2. to allow for flexible agent cycles and the possibility to use widely logic based realizations of agent cycles,
3. to allow multiple levels of use and customization of the framework,
4. to feature the development of secrecy aware and secrecy preserving agents.

We have shown in this article that and how we realized these goals. In the following, we discuss the existing frameworks being closest to satisfying some of these goals.

With respect to non-monotonic knowledge representation, to our knowledge, the only formalism that has been used in actual multiagent systems is ASP. But most implemented works on ASP agents treat only the planning problem independently of the rest of the agent, e. g. in the APLAgent Manager [27] or the DLV^K system [28]. In the literature several proposals for the design of an agent entirely based on ASP have been made, e. g. [29,30]. However, for these no implemented systems or documentation on how to implement such a system are available. To the best of our knowledge, there are only two complete and available multiagent programming frameworks that facilitate the use of ASP for knowledge representation, namely *Jazzyk* [31] and *GOAL* [32,33]. Both of these also feature a modular approach wrt. knowledge representation.

A *Jazzyk* agent consist of several knowledge bases which are realized by knowledge representation modules and an agent program. The agent program consists of a set of rules of the form "when *Query*, then *Update*". The knowledge representation modules implement the *Query* and *Update* methods. The semantics of the agent programs is based on the *Behavioural State Machines* approach developed for *Jazzyk*, on the basis of abstract state machines. The knowledge representation modules allow to use different knowledge representation formalisms and an implemented ASP module is available. With respect to belief operators it implements credulous and skeptical ASP querying. But with respect to belief change it only supports a pure addition of new formulas to the knowledge base and no actual belief change. The only other existing available KR module is based on the *Ruby* programming language which cannot be considered as a logic based knowledge representation formalism.

The *GOAL* Framework [32,33] also allows the specification of modules for knowledge representation and allows, in principle, for the use of different knowledge representation formalisms. There are general interfaces for knowledge representation, but we could not find implementations or examples for any formalism other than *prolog*. The agent programs used in *GOAL* feature a clear syntax and semantics, but are rather inflexible with respect to the use of different agent cycles and architectures. The structure is fixed, goals are defined explicitly and are blindly committed to.

There are no other multiagent frameworks that consider the development of secrecy aware and secrecy preserving agents with explicitly represented secrets and views on other agents, as considered in Angerona. The closest implemented frameworks on the consideration of privacy in multiagent systems consider rather specific problem in distributed problem solving. The $DARE C^2$ system [34] considers the problem of a group of agents that have to collaboratively compute consistent answers to a query and protect their private information at the same time. Confidentiality is expressed by means of the specification of *private* and *askable* literals which are used in a distributed abduction algorithm based on state passing. In [35] quantitative privacy loss is considered in distributed constraint satisfaction problems.

7 Conclusion

We presented the Angerona framework for the implementation of knowledge-based agents. The agent cycle in Angerona can be specified by means of the *ASML* script in combination with the operator interface. The distinction between operator types and their implementation in combination with predefined agent cycles, e. g. the BDI cycle, allows for multiple levels of use and customization. The Angerona framework ASP Plug-in supports the use of various ASP solvers and different extensions thereof, such as *DLV-complex*. On the basis of the latter a planning component on the basis of know-how [16,15] is implemented. Sophisticated change operators on the basis of belief change theory such as [10,11,12,13] are implemented. For the Angerona framework, plug-ins for ASP, OCF and propositional logic are actively used in several available complete simulations.

The Angerona framework is under constant development. We are planning to extend it and to use it as a platform for the development and evaluation of knowledge representation and belief change formalisms in multiagent systems. Our current focus is on the development of secrecy preserving agents. Angerona is also already used in other domains, for instance to experiment with logic-based reasoning on strategies for soccer robots.

Acknowledgements: This work has been supported by the DFG, Collaborative Research Center SFB876, Project A5. (`http://sfb876.tu-dortmund.de`)

References

1. Brachman, R.J., Levesque, H.J.: Knowledge Representation and Reasoning. Elsevier and Morgan Kaufmann Publishers (2004)
2. van Harmelen, F., van Harmelen, F., Lifschitz, V., Porter, B.: Handbook of Knowledge Representation. Elsevier Science, San Diego, USA (2007)
3. Fermé, E., Hansson, S.: AGM 25 years. Journal of Philosophical Logic **40** (2011) 295–331 10.1007/s10992-011-9171-9.
4. Thimm, M.: Tweety - a comprehensive collection of java libraries for logical aspects of artificial intelligence and knowledge representation. In: Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning (KR'14). (July 2014)

5. Gelfond, M., Leone, N.: Logic programming and knowledge representation: the A-Prolog perspective. Artificial Intelligence **138** (2002)
6. Spohn, W.: Ordinal conditional functions: a dynamic theory of epistemic states. In Harper, W., Skyrms, B., eds.: Causation in Decision, Belief Change, and Statistics. Volume 2. Kluwer Academic Publishers (1988) 105–134
7. Bench-Capon, T.J.M., Dunne, P.E.: Argumentation in artificial intelligence. Artificial Intelligence **171**(10-15) (2007) 619–641
8. Krümpelmann, P., Janus, T., Kern-Isberner, G.: Angerona - a multiagent framework for logic based agents. Technical report, Technische Universität Dortmund, Department of Computer Science (2014)
9. R. Biedert, N. Delsaux, T.L.: Java simple plugin framework. http://code.google.com/p/jspf/ [Online; accessed 10-December-2012].
10. Delgrande, J.P., Schaub, T., Tompits, H.: A preference-based framework for updating logic programs. In Baral, C., Brewka, G., Schlipf, J.S., eds.: Proceedings of the 9th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07). Volume 4483., Springer (2007) 71–83
11. Delgrande, J.P., Schaub, T., Tompits, H., Woltran, S.: A general approach to belief change in answer set programming. Computing Research Repository (CoRR) **abs/0912.5511** (2009)
12. Krümpelmann, P., Kern-Isberner, G.: Propagating credibility in answer set programs. In Schwarz, S., ed.: Proceedings of the 22nd Workshop on (Constraint) Logic Programming (WLP'08). Technische Berichte, Martin-Luther-Universität Halle-Wittenberg, Germany (2008)
13. Krümpelmann, P., Kern-Isberner, G.: Belief base change operations for answer set programming. In: Proceedings of the 13th European Conference on Logics in Artificial Intelligence (JELIA'12). Volume 7519 of Lecture Notes in Artificial Intelligence., Springer (2012)
14. Krümpelmann, P., Thimm, M., Kern-Isberner, G., Fritsch, R.: Motivating agents in unreliable environments: A computational model. In Klügl, F., Ossowski, S., eds.: Multiagent System Technologies - 9th German Conference, (MATES 2011), Berlin, Germany, October 6-7, 2011. Proceedings. Volume 6973 of Lecture Notes in Computer Science., Springer (2011) 65–76
15. Thimm, M., Krümpelmann, P.: Know-how for motivated BDI agents (extended abstract). In Decker, Sichman, Sierra, Castelfranchi, eds.: Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'09). (May, 10–15 2009)
16. Krümpelmann, P., Thimm, M.: A logic programming framework for reasoning about know-how. In: Proceedings of the 13th International Workshop on Non-Monotonic Reasoning (NMR'10). (2010)
17. Fermé, E.L., Hansson, S.O.: Selective revision. Studia Logica **63**(3) (1999) 331–342
18. Tamargo, L.H., Thimm, M., Krümpelmann, P., Garcia, A.J., Falappa, M.A., Simari, G.R., Kern-Isberner, G.: Credibility-based selective revision by deductive argumentation in multi-agent systems. In E. Ferme, D. Gabbay, G.S., ed.: Trends in Belief Revision and Argumentation Dynamics, College Publications (2013)
19. Albrecht, E., Krümpelmann, P., Kern-Isberner, G.: Construction of explanation graphs from extended dependency graphs for answer set programs. In Hanus, M., Rocha, R., eds.: Post-proceedings of the 27th Workshop on Functional and Logic Programming (WFLP 2013). Number 8439 in LNAI, Springer (2014) 1–16
20. Foundation for Intelligent Physical Agents: Fipa communicative act library specification (12 2002)

21. Lifschitz, V., Tang, L.R., Turner, H.: Nested expressions in logic programs. Annals of Mathematics and Artificial Intelligence **25(3-4)** (1999) 369–389
22. Krümpelmann, P., Kern-Isberner, G.: On agent-based epistemic secrecy. In Rossi, R., Woltran, S., eds.: Proceedings of the 14th International Workshop on Non-Monotonic Reasoning (NMR'12). (2012)
23. Krümpelmann, P., Kern-Isberner, G.: Secrecy preserving BDI Agents based on Answerset Programming. In: Proceedings of the 11th German Conference on Multi-Agent System Technologies (MATES'13). Volume 8076 of Lecture Notes in Computer Science., Springer (2013) 124–137
24. Biskup, J., Tadros, C.: Preserving confidentiality while reacting on iterated queries and belief revisions. Annals of Mathematics and Artificial Intelligence (2013) 1–49
25. Bordini, R.H., Braubach, L., Dastani, M., Seghrouchni, A.E.F., Gomez-Sanz, J.J., Leite, J., O'Hare, G., Pokahr, A., Ricci, A.: A survey of programming languages and platforms for multiagent systems. Informatica **30** (2006) 33–44
26. Agentprogramming.com: Agent platforms
27. Baral, C., Gelfond, M. In: Reasoning agents in dynamic domains. Kluwer Academic Publishers, Norwell, MA, USA (2000) 257–279
28. Eiter, T., Faber, W., Leone, N., Pfeifer, G., Polleres, A.: Planning under incomplete knowledge. In: Computational Logic—CL 2000. Springer (2000) 807–821
29. Leite, J., Alferes, J., Pereira, L.: $\mathcal{MINERVA}$ - a dynamic logic programming agent architecture. In Meyer, J.J., Tambe, M., eds.: Intelligent Agents VIII. Volume 2333 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2002) 141–157
30. Móra, M.d.C., Lopes, J.G.P., Vicari, R.M., Coelho, H.: BDI models and systems: Bridging the gap. In: Proceedings of the 5th International Workshop on Intelligent Agents V, Agent Theories, Architectures, and Languages (ATAL'98). ATAL '98, London, UK, UK, Springer-Verlag (1999) 11–27
31. Novák, P.: Jazzyk: A programming language for hybrid agents with heterogeneous knowledge representations. In Hindriks, K.V., Pokahr, A., Sardiña, S., eds.: Programming Multi-Agent Systems, 6th International Workshop, ProMAS 2008, Estoril, Portugal, May 13, 2008. Revised Invited and Selected Papers. Volume 5442 of Lecture Notes in Computer Science., Springer (2008) 72–87
32. Hindriks, K., de Boer, F., van der Hoek, W., Meyer, J.J.: Agent programming with declarative goals. In Castelfranchi, C., Lespérance, Y., eds.: Intelligent Agents VII Agent Theories Architectures and Languages. Volume 1986 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2001) 228–243
33. Koen V. Hindriks, W.P.: GOAL User Manual. Delft University of Technology. (2014)
34. Ma, J., Russo, A., Broda, K., Lupu, E.: Multi-agent abductive reasoning with confidentiality. In: AAMAS. (2011) 1137–1138
35. Wallace, R.J., Freuder, E.C.: Constraint-based reasoning and privacy/efficiency tradeoffs in multi-agent problem solving. Artificial Intelligence **161**(1–2) (2005) 209 – 227 Distributed Constraint Satisfaction.