Combining Multiple Knowledge Representation Technologies into Agent Programming Languages

Mehdi Dastani[1] and Koen V. Hindriks[2] and Peter Novák[3] and Nick A.M. Tinnemeier[1]

[1] Utrecht University, Utrecht, The Netherlands, {mehdi,nick}@cs.uu.nl
[2] Delft University of Technology, Delft, The Netherlands, k.v.hindriks@tudelft.nl
[3] Clausthal University of Technology, Clausthal-Zellerfeld, Germany, peter.novak@tu-clausthal.de

Abstract. In most agent programming languages in practice a programmer is committed to the use of a single knowledge representation technology. In this paper we argue this is not necessarily so. It is shown that rational agent programming languages allow for the combination of various such technologies. Specific issues that have to be addressed to realize such integration for rational agents that derive their choice of action from their beliefs and goals are discussed. Two techniques to deal with these issues which enable the integration of multiple knowledge representation techniques are presented: a meaning-preserving translation approach that maps one representation to another, and an approach based on so-called bridge rules which add additional inference power to a system combining multiple knowledge representation technologies.

1 Introduction

Rational agent programming has been motivated on several grounds. One of its motivations has been to provide for a high-level specification framework for agent programs based on common sense concepts such as beliefs, goals, actions, and plans. Such a programming framework comes with several benefits, among others that, though the programming framework is abstract, it can be realized computationally, and, that the programming framework is based on common sense intuitive concepts which nevertheless have a well-defined semantics.

In our view, rational agent programming is abstract in one sense in that it does not commit to a particular knowledge representation language. Though it is common in several concrete programming languages for rational agents to use Prolog like expressions to represent an agent's mental states (cf. Jason[1], 2APL[4], and GOAL[6]), this is more a de facto standard and is not implied by the concept of rational agent programming itself. Some agent frameworks such as Jadex[16] and JACK[18] have taken a much more pragmatic road and use object oriented technology to implement the beliefs and goals of an agent.

Even though in implemented agent programming systems there are ways to add a limited support for external KR technologies, such as accessing a database

engine, this is achieved on the technological level and the solution is bound to be proprietary w.r.t. the implemented application. Therefore as far as we know, it can be said that all of the existing programming frameworks for rational agents in principle commit to a particular kind of knowledge representation (in the sense of Definition 1; see Section 2). In our view, this is not so much a limitation implied by rational agent programming per se. Moreover, we believe that rational agent programming has the integrative potential to facilitate and support the design and construction of agents that use multiple and various knowledge representation languages. In this sense, we are inspired by similar ideas of using various knowledge representation languages that motivated the knowledge interchange format project (KIF)[15].

As a motivating example to allow multiple knowledge representations for building rational agents consider Ape (Airport Passenger E-assistant), a mobile robot with the task of helping passengers by providing them with flight schedules and getting them at the right gate on time. Ape needs to make decisions about, for example, who to serve first, and when to call for the assistance of airport personnel in case she cannot handle all requests in time. In practice, to support such decision-making, it may be best to use various representation technologies. For example, Ape could use Prolog to reason about various decision options, and may have access to a Geographic Information System (GIS) containing the topology of the airport and an Oracle database with the flight schedules.

The aim of this paper is to explore the use of multiple knowledge representations, not only between agents, but also *within a single agent*, both from a pragmatic, practical perspective as well as a theoretical perspective. One of the main motivations to do this stems from the fact that various knowledge representations come with various strengths and weaknesses which would be inherited by an agent program if such a program would be restricted to the use of only a single knowledge representation language. This point has also been particularly argued for by Marvin Minsky[13]. In this work, he argues that to organize intelligence multiple representations are required to do the job. We believe that rational agent programming may provide part of a solution for integrating such a multitude of representations in a clean and well-organized manner with applications in mind.

From a pragmatic point of view, we would also not like rational agent programming languages to commit a programmer to learn a new and specific knowledge representation language that comes with the agent programming language. Learning to program rational agents should not necessarily mean also learning a new and unfamiliar language for representing knowledge. The programmer should have (at least some) freedom to choose his or her favorite language for representing facts about the application domain. The motivation thus is practical, but at the same time it has been and still is quite hard to combine various knowledge representations in a useful manner. In practice, moreover, it should be possible to develop agent applications that incorporate legacy databases, such as, for example, an Oracle flight schedule database. To facilitate such integration, the use of a particular agent language should not imply the need to redesign

the original database but instead an agent language ideally would support and provide an Oracle interface.

In Section 2, we first define knowledge representation in a formally precise way. In order to illustrate the use of multiple knowledge representations within a single agent, we present and discuss in Section 3 the syntax and semantics of the GOAL[6] agent programming language. In Section 4, we introduce a technique to integrate different knowledge representations for the case that the beliefs and goals of a single agent are represented by means of different knowledge representations. In Section 5, we discuss another technique for the case that the belief base of a single agent is represented using different knowledge representations. Finally, in Section 6, we discuss related work and conclude the paper.

2 KR Technology

Our interest in this paper is in basic representation and reasoning tools such as logic, Bayesian networks, or others more than in upper ontologies or domain-specific ontologies. In [5] such representation and reasoning tools are referred to as *knowledge technologies*, a convention we will adopt here as well. A knowledge representation is characterized by means of five roles. Here we are particularly interested in two roles associated with knowledge technologies:

1. Knowledge technologies provide a *fragmentary theory of intelligent reasoning*, i.e. a knowledge technology defines a notion of *inference* that enables drawing conclusions from other available information *represented by means of the same technology*, and
2. Knowledge technologies provide a *medium for pragmatically efficient computation*, i.e. a knowledge technology provides tools and techniques to *compute* with (or to use) the representations supported by the technology.

Other roles of knowledge representations discussed in [5] as being a set of ontological commitments or as providing a medium of human expression are less important in this context. The choice of ontology presumably is application driven, whereas the integration of various representation tools into agent programming languages poses more general challenges.

For our purposes, it is useful to more formally define what a knowledge technology is. Our aim is to relate the semantics of agent programming languages with a *very generic* concept of a knowledge technology. Informally, a knowledge technology is defined here as a language with a *well-defined semantics* that comes with an *inference relation* and *procedures to update* information stored in a knowledge base. Though update procedures may not commonly be regarded as part of a knowledge technology, in our view providing some support and concept of updating a knowledge base to maintain a correspondence with the entities being represented is essential. Moreover, in the context of agent programming update operators are essential, which provides our main motivation to include them in our definition.

Definition 1. (Knowledge Representation Technology)
A knowledge representation technology *is defined as a tuple:*

$$\langle \mathcal{L}, \models, \oplus \rangle$$

where $\mathcal{L}$ *is a representation language which can be used to express declarative sentences, with a given set* $L_q \subseteq L$ *of* query expressions*,* $\models : 2^L \times L_q \rightarrow \{\top, \bot\}$ *is an* inference relation*, and* $\oplus : 2^L \times L \rightarrow 2^L$ *is an* update operator*.*

Definition 1 is intentionally kept very abstract. Our concern here is with the roles that a knowledge representation can fulfill in the context of agent languages. The main roles in this context are that it can be used to *represent* an agent's mental states (e.g., beliefs and goals) and to *query* the agent's mental state, as well as that it allows for performing an *update* on stored representations to maintain a correspondence of the agent's mental state and its environment. These operations can be conceptualized as providing a TELL and ASK interface on a database of stored representations as discussed in [10]. We assume (though it is in our quite general setting not strictly necessary to do so) that provided consistency of a database Σ and a sentence ϕ the update operator $\oplus$ preserves consistency, i.e. $\Sigma \oplus \phi$ is consistent as well and the database remains unchanged if the sentence to update with is inconsistent ($\Sigma \oplus \bot = \Sigma$). $\bot$ denotes here falsity. Information is expressed by means of *sentences* from the language that can be *true or false*, but we also allow for semantics that incorporate more truth values as is typical in multi-valued logic. A *knowledge base*, or, alternatively, simply a *database*, is then defined as a set of sentences from the knowledge representation language.

The use of the inference relation and update operations in defining an agent language is clarified below where we show how the semantics of an agent language can be defined in terms of these operators. In concrete agent languages the update operator typically is not a single operator $\oplus$ but one or more operations to *add* and *remove* stored representations from a database (though it may be a single operator, as e.g. postconditions of actions can be interpreted in a STRIPS-like fashion as add/delete lists). The definition provided suits, however, for the purposes of this paper.

Some typical examples of knowledge representation technologies that fit the definition are logical languages such as first-order logic and description logics such as the Ontology Web Language (OWL), Frame languages [12], Prolog, Answer Set Programming, Constraint Programming, relational databases, and others, such as Bayesian Network also fit though the notion of an update in Bayesian Networks is rather limited and it is not common to view a Bayesian network as a database consisting of a set of sentences.

To illustrate the scope of our definition, we discuss the example of relational databases - one of the most common technologies for storing information in practice - in slightly more detail. In this case, the representational language $\mathcal{L}$ can be identified with languages such as Datalog [3] or SQL. Datalog is a declarative database query language whereas SQL is a declarative language for both querying and updating relations stored in a database. SQL query formulas

provide the query language $\mathcal{L}_q$ whereas SQL update formula can be used to specify the insertion or removal of relations from a database. Finally, the SQL interpreter (the relational database engine) implements the inference relation $\models$ and update operator $\oplus$. The correspondence between Datalog or SQL and the abstract KR language scheme of Definition 1 thus is rather straightforward.

3 Integrating Multiple KRTs into Rational Agents

The question of how to usefully integrate multiple knowledge representations into a software application in general poses several complex issues [15]. One particularly interesting question is how to facilitate the derivation of a conclusion from knowledge stored in different representations in various databases controlled or accessible by the agent. The combination of multiple knowledge technologies into a rational agent programming language, however, raises some specific issues of its own. To illustrate these, we first provide a very brief overview of the essential ingredients of the agent programming language GOAL [6], which are introduced here mainly for illustrative purposes. Only those parts of the GOAL language related to the subject of this paper are introduced. The interested reader is referred to [6] for a more extensive presentation of the language. GOAL agents - as agents programmed in related agent programming languages such as 2APL, Jason, and Jadex - derive their choice of action from their *beliefs* and *goals*. Beliefs and goals are represented by means of some knowledge technology (as mentioned earlier, typically Prolog is used), and, for the purpose of this paper, it will be particularly relevant to look into the relation between these two notions.

3.1 GOAL: Syntax and Semantics

The main defining features of a rational agent programming language are constructs for defining the agent's mental state, including its *beliefs and goals*, its *action selection mechanism* used by the agent to derive a choice of action from its beliefs and goals, and a *commitment strategy* which determines when an agent will revise its goals given its beliefs. An example of a commitment strategy is to only drop a goal when the agent believes it has been achieved, a so-called *blind* commitment strategy (cf. [17]). The semantic interdependence of an agent's goals and beliefs differentiate rational agent programming languages from other high-level languages such as database languages. At the same time, however, this interdependency raises some special issues for integrating multiple knowledge representation languages into such a language. In order to clarify these issues some of the key semantic rules of GOAL that formalize these interdependencies are introduced.

In the following we use $\langle \Sigma, \Gamma \rangle$ to denote an arbitrary mental state of a rational agent where Σ is the *belief base* and Γ is the *goal base* of the agent. Although it is usual to assume both the belief base as well as the goal base consist of sentences from a single knowledge technology (e.g. Prolog), for the purposes of this paper, we are interested in relaxing this assumption in two ways. First, the

belief base and goal base do not need to be based on one and the same knowledge technology. Second, the belief base does not need to be monolithic and might instead consist of various databases based on various knowledge technologies. Similarly, a goal base might be based on multiple technologies, but we do not discuss this possibility explicitly in this paper since it seems less useful to us. However, the the same techniques we propose for handling belief base multiple belief bases could be applied also in this situation.

The issues that are introduced by relaxing this assumption can be illustrated after introducing some basic definitions.

First, we introduce a belief operator $\mathbf{bel}(\phi)$ and goal operator $\mathbf{goal}(\phi)$ and associated semantics which express that an agent has a belief or goal ϕ in a mental state $M = \langle \Sigma, \Gamma \rangle$. These operators enable the expression of conditions on mental states of an agent. Formally, a *mental state condition* is a boolean combination of belief and goal conditions, i.e.

$$m ::= \mathbf{bel}(\phi) \mid \mathbf{goal}(\phi) \mid \neg m \mid m \wedge m$$

The semantics of simple belief and goal conditions is defined next. In the standard way these can be extended to handle boolean combinations.

Definition 2. (Semantics of Mental State Conditions)
Let $M = \langle \Sigma, \Gamma \rangle$ be a mental state. Then the semantic clauses for **bel** *and* **goal** *are provided by:*

- $M \models \mathbf{bel}(\phi)$ *iff* $\Sigma \models \phi$
- $M \models \mathbf{goal}(\phi)$ *iff* $\exists \gamma \in \Gamma$ *s.t.* $\gamma \models \phi$ *and* $\Sigma \not\models \phi$.

According to this definition, an agent has a belief ϕ if ϕ is entailed by the agent's belief base and a goal ϕ if and only if ϕ is entailed "locally" from its goal base (i.e. from one of the agent's goals in its goal base) but does not follow from its belief base. What is important in this context to note is that this semantic clause requires us to both verify whether ϕ is entailed by the goal base as well as the belief base.

Second, we introduce a transition rule which defines the operational ("execution step") semantics for GOAL agents. This transition rule defines when the agent can perform an action. The execution of an action involves updating the agent's mental state and to formally define it we need a *transition function* $\mathcal{T}(\mathtt{a}, \Sigma)$. Presumably, this transition function can be defined in terms of the update operators associated with the knowledge representation language used to specify the belief base, e.g. we could define the function by $\mathcal{T}(\mathtt{a}, \Sigma) = \Sigma \oplus \psi$ where ψ is the postcondition of action $\mathtt{a}$.

Definition 3. (Action Execution Rule)
Let $\langle \Sigma, \Gamma \rangle$ be a mental state, c be a conditional action of the form **if** φ **then** $\mathtt{a}$ *where φ is a mental state condition. Then the execution of the conditional action c is defined by:*

$$\frac{\langle \Sigma, \Gamma \rangle \models \varphi}{\langle \Sigma, \Gamma \rangle \longrightarrow \langle \Sigma', \Gamma' \rangle}$$

where: $\Sigma' = \mathcal{T}(\mathtt{a}, \Sigma)$, and $\Gamma' = \Gamma \setminus \{\gamma \in \Gamma \mid \Sigma' \models \gamma\}$.

This rule defining the semantics of action execution also "implements" the blind commitment strategy discussed above. In the absence of other facilities to modify goals, an agent will drop a goal γ only if it is believed to be achieved (i.e. $\Sigma \models \gamma$). This automatic update of the goal base requires that each goal in the goal base is checked against the belief base to verify if it has been achieved (though in practice more efficient implementations are possible).

The semantics introduced above enables us to introduce the issues raised by introducing multiple knowledge technologies into a rational agent more precisely. First, by allowing the belief base and the goal base to be based on different knowledge technologies (but still assuming each uses a *single* technology) we need a means to relate these technologies. The reason is that Definition 3 requires an agent to verify whether its goals γ have been achieved by verifying whether they are entailed by its belief base Σ. Definition 3 thus requires that a query γ specified using one knowledge technology can be resolved using a database Σ based on another knowledge technology.

Second, by allowing a belief base to consist of multiple databases (i.e. $\Sigma = D_1 \times \ldots \times D_n$) using different knowledge technologies the question arises what it means to query such a belief base. It may be useful to decompose an agent's belief base into several databases in practice, e.g. to integrate legacy databases, but how does the agent derive a conclusion that requires *combining information* from such a distributed set of databases?

Finally, the semantics of both Definition 2 and 3 pose certain requirements for the knowledge representation technologies used for goal bases. The point is that both definitions require that a goal base can be viewed as a *set*, either to check whether an element of the goal base entails some formula, or to remove achieved goals from the goal base (Df. 2). Though most logic-based knowledge technologies do support such a view not all do so naturally. For example, it is not clear how a Bayesian network could be viewed as a set, and, in practice, even Prolog systems allow for multiple occurrences of facts. Although the latter issue is easily circumvented, the use of Bayesian networks to represent goals in a goal base is practically excluded. In the remainder, we will assume that goal bases are always implemented with a KRT that allows for set-theoretic view of its associated databases.

There are several options to deal with the issues discussed above. It is unlikely that there is one and only one unique best solution for handling these issues. Our strategy here therefore will be to discuss some of the more promising options. We do not claim to discuss an exhaustive list of options. Our main interest is in the specific issues that are raised when dealing with the problem of combining different knowledge representations in the context of agent programming and our objective is to provide some generic solutions to be able to usefully combine knowledge representations into a rational agent.

A natural first suggestion is that if it would be possible to somehow translate one knowledge representation into another one. In that case, below we show that some of the issues specific to rational agents can be solved using translation operators. In the next section some variations on this topic are explored.

A second suggestion explored in this paper is to use so-called *bridge rules* to connect knowledge stored in various databases (or contexts) and derive a new conclusion from those knowledge sources much in the spirit of multi-context logic [9]. This technique is discussed in Section 5.

4 A Translation Approach to Combine KRT's

In this section, we assume that the belief and goal bases of an individual agent are represented using different knowledge representation technologies. This translation approach will be applied to define the semantics of the GOAL language to fit multiple KRT's. This approach is based on the assumption that the expressions of one knowledge representation language can be translated to expressions of the second language by means of a translation operator.

Definition 4. (translation operator) *Let L_1 and L_2 be two knowledge representation languages. A translation operator τ from L_1 to L_2 is a function from L_1 to L_2. The translation operator can be defined on sets of formula as follows: $\tau(\{\phi_1, \ldots, \phi_n\}) = \{\tau(\phi_1), \ldots, \tau(\phi_n)\}$.*

A translation operator can be used to connect knowledge representation technologies with each other if their entailment relations and update operators impose the same structures on the set of language expressions.

Definition 5. (KRT translation operator) *Let $\mathcal{K}_1 = \langle L_1, \models_1, \oplus_1 \rangle$ and $\mathcal{K}_2 = \langle L_2, \models_2, \oplus_2 \rangle$ be two knowledge representation technologies and $\tau^{\mathcal{K}_1 \to \mathcal{K}_2} : L_1 \to L_2$ be a translation operator. We write τ instead of $\tau^{\mathcal{K}_1 \to \mathcal{K}_2}$ if it is clear that $\tau : L_1 \to L_2$.*

τ is a KRT translation operator from $\mathcal{K}_1$ to $\mathcal{K}_2$ iff

- $\forall \Lambda \subseteq L_1, \forall \phi \in L_1 \ : \ \Lambda \models_1 \phi \ \rightarrow \ \tau(\Lambda) \models_2 \tau(\phi)$
- $\forall \Lambda \subseteq L_1, \forall \phi \in L_1 \ : \ \tau(\Lambda \oplus_1 \phi) \ = \ \tau(\Lambda) \oplus_2 \tau(\phi)$

In this paper, we will use a particular knowledge representation technology which is based on a propositional language, its corresponding well-known entailment relation and an update operator. Using this specific knowledge representation technology, we can study some logical properties of other knowledge representation technologies and investigate their behaviors when they are used in agent programming languages.

Definition 6. (Logically founded KRT) *Let $\mathcal{K}_p = \langle L_p, \models_p, \oplus_p \rangle$ be the propositional knowledge representation technology, where L_p is the language of propositional logic, $\models_p$ is its corresponding entailment relation, and $\oplus_p$ is an update function that satisfies some reasonable belief update postulates, amongst which the consistency preservation property.*

Let $\mathcal{K} = \langle L, \models, \oplus \rangle$ be an arbitrary knowledge representation technology. $\mathcal{K}$ is called logically founded if and only if there exists a KRT translation operator τ from $\mathcal{K}_p$ to $\mathcal{K}$. Moreover, we say $\Lambda \subseteq L$ is τ-consistent only if $\tau(\Lambda)$ is consistent.

The choice of propositional language in the above definition is not strict. In fact, we may use an expressive but computational subset of predicate logic or KIF[15]. We use the propositional language to simplify the presentation of the relevant part of our approach. In the rest of this section, we use the translation operator and present two ways to adapt the semantics of the GOAL programming language.

4.1 Intermediate KRT Translation Approach

One approach is to assume that an agent programming language comes with a propositional knowledge representation technology without assuming how the belief and goal bases are represented. The proposition knowledge representation technology is used to express the query and update expressions. For example, in the GOAL programming language, the propositional formulae are used to implement the pre- and post-conditions of actions without making any assumption on the belief and goal languages. We call such a programming language *generic*.

In particular, one and the same language for query expressions (e.g., a propositional language L_p) is assumed, whereas the representation languages used to represent the belief and goal bases may differ from each other and from the generic language L_p embedded in the agent language. The entailment relation for the propositional language L_p is well-known. Moreover, various update operators are studied for propositional language and some postulates are proposed that should be valid for such operators. For example, if we consider only propositional atoms and their negations, then an update operator can be defined in terms of addition and deletion of atoms.

Additionally we assume that the knowledge representation technologies used for beliefs and goals are logically founded. This implies that there exists a KRT translation operator that maps propositional expressions to the expressions of the languages used in the knowledge representation technologies. In order to illustrate this approach, we apply it to the GOAL programming language. The semantic clauses of the GOAL programming language as defined above can be modified to allow for the integration of multiple knowledge representation technologies.

Definition 7. (Semantics for Generic GOAL) *Let* $\mathcal{K}_b = \langle L_b, \models_b, \oplus_b \rangle$ *and* $\mathcal{K}_g = \langle L_g, \models_g, \oplus_g \rangle$ *be logically founded KRT, based on KRT translation operator* τ_b *and* τ_g*, respectively. Let also* $M = \langle \Sigma, \Gamma \rangle$ *be a mental state with* $\Sigma \subseteq L_b$*,* $\Gamma \subseteq L_g$*,* $\phi \in L_p$ *be a proposition, and* $c =$ **if** φ **then** a *be a conditional action. Let* $\psi \in L_p$ *be a proposition representing the postcondition for action* a*. The semantics of the generic GOAL language can be defined as follows:*

- $M \models \mathbf{bel}(\phi)$ *iff* $\Sigma \models_b \tau_b(\phi)$
- $M \models \mathbf{goal}(\phi)$ *iff* $\exists \gamma \in \Gamma : \gamma \models_g \tau_b(\phi)$ *and* $\Sigma \not\models_b \tau_b(\phi)$
- *Action execution:*

$$\frac{\langle \Sigma, \Gamma \rangle \models \varphi}{\langle \Sigma, \Gamma \rangle \longrightarrow \langle \Sigma', \Gamma' \rangle}$$

where: $\Sigma' = \Sigma \oplus_b \tau_b(\psi)$

$\Gamma' = \Gamma \setminus \{\tau_g(\psi) \in \Gamma \mid \Sigma' \models_b \tau_b(\psi)\}$

The semantics of the GOAL programming language as defined above has some interesting properties. In particular, despite using different knowledge representation technologies for belief and goal bases, it can be shown that when executed the agent's belief base remains consistent if the initial belief base of the agent is consistent. Moreover, it can be shown that the agent will never have goals that are already achieved.

Proposition 1. *Let* $\mathcal{K}_b = \langle L_b, \models_b, \oplus_b \rangle$ *and* $\mathcal{K}_g = \langle L_g, \models_g, \oplus_g \rangle$ *be logically founded KRT based on* τ_b *and* τ_g*, respectively. Let* $\langle \Sigma_0 \subseteq L_b, \Gamma_0 \subseteq L_g \rangle$ *be an agent's initial state, and* $\langle \Sigma_i, \Gamma_i \rangle$ *(for* $i > 0$*) be a state generated by executing the agent according to the semantics as defined above. Then,*

- *if* Σ_0 *is* τ_b*-consistent then* Σ_i *is* τ_b*-consistent for* $i > 0$
- *if* $\Sigma_i \models_b \tau_b(\phi)$ *then* $\Gamma_i \not\models_g \tau_g(\phi)$ *for* $\phi \in L_p$ *and* $i > 0$.

An advantage of this approach is that agent programs, which are implemented in the generic version of the GOAL programming language, are independent from the employed knowledge representation technologies. Consequently, changing the employed knowledge representation technologies requires only a modification of the translation operators such that nothing needs to be changed in the agent programs. Furthermore, an agent program can be designed before a final choice for a specific knowledge representation technology is made. A disadvantage is that we should specify the translation operator in terms of the set of belief queries which can be a large set.

4.2 Direct KRT Translation Approach

In this subsection, we assume that the adapted agent programming language is defined in terms of two distinct knowledge representation technologies, one to implement the belief base and its corresponding query expressions and one to implement the goal base and its corresponding query expressions. The idea is thus to represent each mental attitude (goals and beliefs) and its corresponding queries with one and the same knowledge representation technology. In this approach, the knowledge representation technologies form integral constituents of the definition of the agent programming language. We illustrate this approach by applying it to the GOAL programming language. As the query languages depend on the knowledge representation technologies, we first redefine the syntax of the GOAL programming language by allowing expressions of different knowledge representation technologies to be used as query expressions.

Definition 8. (Syntax for Multiple KRTs) *Let* L_b *and* L_g *be representation languages for belief and goal expressions, respectively. Let* $\Sigma \subseteq L_b$ *and* $\Gamma \subseteq L_g$*. The GOAL programming language based on Multiple KRT's can be defined as follows:*

– *'if φ then* `a`*', where*
  - *if* $\mathbf{bel}(\phi)$ *occurs in* φ*, then* $\phi \in L_b$
  - *if* $\mathbf{goal}(\phi)$ *occurs in* φ*, then* $\phi \in L_g$
  - $PostCondition(\mathtt{a}) \in L_b$

Here we assume a translation operator that translates L_g into L_b. Given such translation operator, the semantics for the GOAL programming language can be redefined as follows.

Definition 9. (Semantics for Multiple KRTs) *Let* $\tau : L_g \rightarrow L_b$*,* $\mathcal{K}_b = \langle L_b, \models_b, \oplus_b \rangle$ *and* $\mathcal{K}_g = \langle L_g, \models_g, \oplus_g \rangle$*. Let also* $M = \langle \Sigma, \Gamma \rangle$ *be a mental state with* $\Sigma \subseteq L_b$*,* $\Gamma \subseteq L_g$*,* $\phi_b \in L_b$*,* $\phi_g \in L_g$*, and* $c = \mathbf{if}\ \varphi\ \mathbf{then}\ \mathtt{a}$ *be a conditional action. Let* $\psi \in L_b$ *be the postcondition for action* `a`*. The semantics of the GOAL programming language based on Multiple KRT's can be defined as follows:*

– $M \models \mathbf{bel}(\phi_b)$ *iff* $\Sigma \models_b \phi_b$
– $M \models \mathbf{goal}(\phi_g)$ *iff* $\exists \gamma \in \Gamma : \gamma \models_g \phi_g$ *and* $\Sigma \not\models_g \tau(\phi_g)$
– *Action execution:*

$$\frac{\langle \Sigma, \Gamma \rangle \models \varphi}{\langle \Sigma, \Gamma \rangle \longrightarrow \langle \Sigma', \Gamma' \rangle}$$

where: $\Sigma' = \Sigma \oplus_b \psi$
$\Gamma' = \Gamma \setminus \{\psi \in \Gamma \mid \Sigma' \models_b \tau(\psi)\}$

Like the previous approach, it is shown that the consistency of the belief base can be preserved and its relation with the goal base can be maintained.

Proposition 2. *Let* $\mathcal{K}_b = \langle L_b, \models_b, \oplus_b \rangle$ *be a KRT,* $\mathcal{K}_g = \langle L_g, \models_g, \oplus_g \rangle$ *be logically founded based on* τ_p *and let* τ *be a KRT translation operator from* L_g *to* L_b*. Let also* $\langle \Sigma_0 \subseteq L_b, \Gamma_0 \subseteq L_g \rangle$ *be an agent's initial state, and* $\langle \Sigma_i, \Gamma_i \rangle$ *(for* $i > 0$*) be a state generated by executing the agent according to the agent semantics as defined above. Then,*

– *if* Σ_0 *is* τ_p*-consistent then* Σ_i *is* τ_p*-consistent for* $i > 0$*.*
– *if* $\Sigma_i \models_b \tau(\phi)$ *then* $\Gamma_i \not\models_g \phi$ *for* $\phi \in L_g$ *and* $i > 0$*.*
– $\mathcal{K}_b = \langle L_b, \models_b, \oplus_b \rangle$ *is logically founded.*

An advantage of this approach is that only a translation has to be made for the goals of the agents, which are known at design time. A disadvantage is that if the employed knowledge representation technologies for the goal or belief bases are changed, not only a new translation function has to be defined, but also the code of the agent program should be updated.

5 Integrating Multiple KRTs into a belief base

Although in principle the techniques of translating sentences specified using different KRTs also can be applied to handle inferencing on a composed belief base

that consist of multiple belief bases (cf. previous section and e.g. [15]), we propose another technique to deal with such inferencing. One reason is that translation may work well only for certain application types that use relatively small knowledge bases. Another reason is that we believe that the technique to handle multiple KRTs should facilitate drawing conclusions that *combine* information from several of the databases a belief base may be composed of. As a simple example, consider the airport service robot again which this time needs to give lost luggage back to a passenger. The robot will need to combine information from several databases to derive the quickest way to do this. A Prolog-like query to obtain this information might look like `loc(luggage,L,passenger),loc(passenger,P),route(L,P,R)`. A translation approach that has to deal with a query like this would give rise to redundant processing and search for the right source of information that can answer (part) of the query. It is a priori not clear from the query itself to identify the right database to pose (part of) the query to. A technique is needed that allows an agent programmer to "guide" the reasoning of an agent.

The approach suggested in this section proposes to connect various knowledge bases by means of so-called *bridge rules*. Instead of translating languages, the main idea of bridge rules is to add additional *inference power* on top of the two or more knowledge technologies that are to be integrated into the agent application. The mechanism to do so should also provide a means to connect pieces of knowledge represented by different knowledge technologies. The relation suggested by calling these rules bridge rules with *multi-context logic* is intentional [9]. Multi-context logic provides a framework that can be used to achieve our objective to integrate various knowledge technologies in the sense of Definition 1 (cf. also [8] for a similar proposal).

Bridge rules are particular kind of inference rules. They sanction an additional inference to a conclusion represented using one knowledge technology given available inferences and associated conclusions using other knowledge technologies. More formally, a bridge rule can be defined as a rule of the following form:

$$\varphi_1, \ldots, \varphi_2 \ \vdash \ \psi$$

where each φ_i and ψ are representations from a particular knowledge representation language $\mathcal{L}$. The intended semantics is that a bridge rule allows the inference of ψ if all φ_i can be derived somehow given the inference relations $\models_{\mathcal{K}_i}$ associated with each φ_i. A bridge rule thus sanctions the inference of ψ given these other inferences, and allows ψ to be used in other inferences to draw certain conclusions again. It does not require such inferences to be made, nor does it require any updates on knowledge bases or the like; these rules only provide additional inference power.

Continuing the example of the service robot, suppose information about passengers is stored "ad hoc" in the robots' belief bases implemented in Prolog, lost luggage information is stored in a SQL database, and routing information may be requested from a GIS system implemented using OO database technology. In that case, a bridge rule could be used to compute a route by directing queries

to these various information sources by a rule that such as the following:

$$
\begin{array}{l}
loc(passenger, P), \\
\texttt{SELECT } L \texttt{ FROM} \\
\quad LostLuggage \texttt{ WHERE } Pgnr = passenger, \\
mapGIS.get\_route(L, P, R) \\
\vdash route(R)
\end{array}
$$

It will be clear that the syntax of the bridge rules provides clues how to resolve a particular (part of a) query.

The idea thus is to allow a programmer to add specific bridge rules to an agent program to facilitate inferences using multiple knowledge technologies. The programmer is supposed to be able to design such rules given his knowledge about the application and the use that the various knowledge technologies have been put to. Bridge rules only add additional inference power and give rise to a new inference relation $\models^*$. The inference relation $\models^*$ defines when a query $\phi \in \mathcal{L}$ from some knowledge representation language $\mathcal{L}$ is entailed by multiple knowledge bases using various knowledge technologies which are possibly related by a set of bridge rules $\mathcal{B}$.

Definition 10. (Induced Inference Relation)
Let a set of knowledge bases $KB_1, \ldots, KB_n$ with associated knowledge technologies $\mathcal{K}_i = \langle \mathcal{L}_i, \models_i, \oplus_i \rangle$ for $i = 1, \ldots, n$ be given. Furthermore, assume a set of bridge rules $\mathcal{B}$ consisting of formulas of the form $\varphi_1, \ldots, \varphi_m \vdash \psi$ with φ_i, ψ each taken from one of the knowledge representation languages $\mathcal{L}_i$. Then the induced inference relation $\models^*$ is defined by:

$$KB_1, \ldots, KB_n, \mathcal{B} \models^* \phi \text{ iff } \exists i : 1 \leq i \leq n \wedge KB_i^* \models_i \phi$$

where the KB_i^* are defined by simultaneous induction as the smallest set such that:

- *$KB_i \subseteq KB_i^*$, and*
- *whenever $\varphi_1, \ldots, \varphi_m \vdash \psi \in \mathcal{B}$ with $\psi \in \mathcal{L}_i$ and for all $j = 1, \ldots, m$ there is a k such that $KB_k^* \models_k \varphi_j$, then $\psi \in KB_i^*$.*

The semantics indicates that each knowledge base with an additional set of bridge rules can be computed incrementally, and that bridge rules can be viewed as a kind of completion operator. An implementation using backward chaining would make this approach a practical option for integration into agent languages.

It should be clear that a translation approach and an approach using bridge rules do not exclude each other. In fact, both can be used to address the issue discussed in the previous section - to facilitate inference when a belief base and goal base use different KRTs - as well as the issue discussed in this section - handling inference in a composed belief base. Bridge rules thus can be viewed as kind of a translation operators but provide a programmer with more flexibility whereas the approach using translation operators is more generic.

6 Conclusion and related work

The paradigm of rational agents and multi-agent systems provides an integrative view on a multitude of topics in AI. Agents can usefully exploit the entities to strengths of various technologies, especially w.r.t. knowledge representation and control. To our knowledge, the problem of integrating heterogeneous knowledge bases in a single agent system arose in the agent-oriented programming community only recently. Most state-of-the-art agent oriented programming frameworks do prescribe employment of a single knowledge base in a fixed KR language. Most of the time it is either a logical language (*Prolog*), or a programming language in which the particular framework is developed (*Java*). Homogeneous KBs in such systems do not pose a problem, as formulas of different KBs come from the same language, hence the same entailment/update operators can be used with them.

We are aware of only two efforts in the context of agent oriented programming which aimed at mixing heterogeneous knowledge representations in a single agent system. Project *IMPACT* [7] aimed at integration of heterogeneous legacy knowledge bases accessible to an agent. *IMPACT* treats each underlying KB as an opaque *body of software code*, modeled as a set of predefined functions providing access to the underlying data objects capturing a part of the current agent's (mental) state. The agent logic program consists of a set of *if-then-else* rules regarded as a logic program. However, as *IMPACT* did aim for integration of heterogeneous information sources in the first place, *IMPACT* agents, by default, do not maintain any stronger semantic conditions on their knowledge bases (such as e.g. blind commitment strategy). That implies no special need for translation of formulas from different KBs. In terms of approaches introduced in this paper, *IMPACT* can be seen as an instance of a system implementing a mechanism similar to bridge-rules discussed in Section 5. *Modular BDI architecture* [14] is another recent attempt to approach combining heterogeneous KR technologies in an BDI-inspired agent system. Even though the agent system dynamics and semantics differs from that of *IMPACT*, the approach to integration of heterogeneous KBs in a single agent is very similar.

In this paper we explored several approaches to integrate various knowledge representation technologies so that these can be exploited in a single agent system in a consistent way. This is as an initial attempt to study the problem. We believe that an implemented proof of concept for the presented integration approaches is necessary. Moreover, in our future research we want to compare our approach with the results regarding translating database schemes, such as [11].

We would like to emphasize that the use of propositional logic as an intermediary knowledge representation technology was for simplicity reasons and in order to focus on the problem of integration of knowledge representation technologies. We believe that for developing practical agent systems the propositional knowledge representation technology can easily be extended with first-order elements (such as variables) or even with representation technologies as developed in KIF[15].

References

1. R. Bordini, J. Hübner, and R. Vieira. *Multi-Agent Programming - Languages, Platforms and Applications*, chapter Jason and the Golden Fleece of agent-oriented programming. Springer, 2005.
2. R. H. Bordini, M. Dastani, J. Dix, and A. E. F. Seghrouchni. *Multi-Agent Programming Languages, Platforms and Applications*, volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*. Kluwer Academic Publishers, 2005.
3. S. Ceri, G. Gottlob, and L. Tanca. What you always wanted to know about datalog (and never dared to ask). *IEEE Trans. of KDE*, 1(1), 1989.
4. M. Dastani and J. Meyer. A Practical Agent Programming Language. In *Proc. of PROMAS*, 2007.
5. R. Davis, H. E. Shrobe, and P. Szolovits. What is a knowledge representation? *AI*, 14(1):17–33, 1993.
6. F. de Boer, K. Hindriks, W. van der Hoek, and J.-J. Meyer. A Verification Framework for Agent Programming with Declarative Goals. *Journal of Applied Logic*, 2007.
7. J. Dix and Y. Zhang. *Multi-Agent Programming - Languages, Platforms and Applications*, chapter IMPACT: A Multi-Agent Framework with Declarative Semantics. Springer, 2005.
8. A. Farquhar, A. Dappert, R. Fikes, and W. Pratt. Integrating Information Sources Using Context Logic. In C. Knoblock and A. Levy, editors, *Information Gathering from Heterogeneous, Distributed Environments*, 1995.
9. F. Giunchiglia and L. Serafini. Multilanguage hierarchical logics or: How we can do without modal logics. *AI*, 65(1):29–70, 1994.
10. H. Levesque. Foundations of a functional approach to knowledge representation. *AI*, 23:155–212, 1984.
11. J. A. Makowsky and E. V. Ravve. Translation schemes and the fundamental problem of database design. In B. Thalheim, editor, *Conceptual Modeling - ER'96, 15th International Conference on Conceptual Modeling, Cottbus, Germany, October 7-10, 1996, Proceedings*, volume 1157 of *Lecture Notes in Computer Science*, pages 5–26. Springer, 1996.
12. M. Minsky. A framework for representing knowledge. In J. Haughland, editor, *Mind Design*, pages 95–128. MIT Press, 1981.
13. M. Minsky. *The society of mind*. Simon & Schuster, Inc., New York, NY, USA, 1986.
14. P. Novák and J. Dix. Modular BDI architecture. In H. Nakashima, M. P. Wellman, G. Weiss, and P. Stone, editors, *Proc. AAMAS 2006*, pages 1009–1015. ACM, 2006.
15. R. S. Patil, R. E. Fikes, P. F. Patel-Schneider, D. McKay, T. Finin, T. R. Gruber, and R. Neches. The DARPA knowledge sharing effort: Progress report. In C. Rich, B. Nebel, and W. Swartout, editors, *Princ. of KR and Reasoning: Proc. of the Third Int. Conf.* Morgan Kaufmann, 1992.
16. A. Pokahr, L. Braubach, and W. Lamersdorf. *Jadex: A BDI Reasoning Engine*, chapter 6, pages 149–174. Volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations* [2], 2005.
17. A. S. Rao and M. P. Georgeff. Modeling rational agents within a BDI-architecture. In *Proc. of the 2nd Int. Conf. on Princ. of KR and Reasoning*, pages 473–484, 1991.
18. M. Winikoff. *JACK(TM) Intelligent Agents: An Industrial Strength Platform*, chapter 7, pages 175–193. Volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations* [2], 2005.