

On Dynamically Generated Ontology Translators in Agent Communication*

Rogier M. van Eijk,† Frank S. de Boer, Wiebe van der Hoek,
John-Jules Ch. Meyer
*Institute of Information and Computing Sciences, Utrecht University,
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands*

In this paper, we consider communication between agents that employ different vocabularies to represent information. In particular, we develop a communication mechanism in which translators between the vocabularies of agents are generated. Instead of being defined in advance, these translators are dynamically constructed during execution of the system, and are based both on the information that the agents exchange and on their underpinning ontologies. Moreover, these translators are not necessarily defined for the total vocabulary of the agents, but instead, only for the parts that have been involved in communication steps. The framework can for instance be used to study and to analyze experiments as performed in the research on the origins of language, like language games, in which the purpose of communication is to come to a mutual understanding of the agents' vocabularies. © 2001 John Wiley & Sons, Inc.

1. INTRODUCTION

Homogeneous multiagent systems consist of agents that can communicate with each other in terms of a common vocabulary. However, in *heterogeneous* multiagent systems in which agents employ private vocabularies to represent information, the preexistence of such a shared vocabulary cannot be guaranteed. Consider for instance electronic market places in which agents from *different origins* are put together to interact with each other. Agents in such systems have usually been implemented in different institutions by different programmers and as such typically employ different vocabularies to represent information, such as different currencies to denote prices (dollar, euro), units to represent linear measures (inch, centimeter), units to represent temperatures (Fahrenheit, Celsius) and so on.

In this paper, we address the problem of how we can program such heterogeneous multiagent systems, in which agents that employ different vocabularies to represent information, communicate with each other in such a way

† This paper is a revised and extended version of Ref. 8.

* Author to whom correspondence should be addressed; e-mail: rogierr@cs.uu.nl.

that they are able to develop some form of mutual understanding of their individual vocabularies. We do this by means of a basic multiagent programming language, called ACPL, that is tailored to the information-processing aspects of agents. An important feature of this language is that it allows agents to communicate through the exchange of high-level forms of information (in contrast to the traditional communication of lower level data as expressions and values).

The exchange of information proceeds by bilateral communication steps in which an agent tells a particular piece of information that constitutes an answer to a particular question that is asked by another agent. Such steps of communication are implemented by means of an action $\text{tell}(c, \varphi)$ to tell the information φ along a communication channel c , and the action $\text{ask}(c, \psi)$ to ask, along the channel c , for the information ψ . We speak of successful communication in case the information φ constitutes an answer to the question ψ , or in other words, if ψ can be derived from φ , which we denote by $\varphi \vdash \psi$. One of purposes of the present paper is to generalize this basic communication mechanism with the dynamic generation of translators between the agents' vocabularies. The idea is that the generation of these translators is not only controlled by the information φ and ψ , but also by the underpinning ontologies, which define the relations that exist between the different symbols in a vocabulary.

The paper is organized as follows. In Section 2, we start by introducing the necessary concepts from first-order logic. Additionally, we define the syntax of translation formulas and characterize when a translation formula constitutes a translator from one formula to another. In Section 3, we study in what way the agents' underpinning ontology can be used to reduce the number of possible translators. The subject of Section 4 concerns the multiagent programming language ACPL, which is a language tailored to the information-processing aspects of the multiagent system. We refine the communication mechanism of this language with the dynamic generation of translators, and we illustrate the resulting framework by means of several examples. Finally, in Section 5, we wrap up by considering the relation with other approaches and we identify some issues for future research.

2. TRANSLATIONS BETWEEN VOCABULARIES

We distinguish two options to define translations between vocabularies. The first approach, which constitutes the usual road taken in multiagent frameworks,^{4,12} is to have *static* translation functions that are constructed *before* execution of the system. That is, a programmer specifies how information from one first-order system is translated to information in another first-order system. A disadvantage of this approach is that it is not entirely modular: for instance, if the vocabulary of an agent is replaced by a new vocabulary—because in the optimization of the agent program the latter appears to be more efficient—then we must also modify the translations to the vocabularies of the agent's communication partners. In this way, a local modification in the vocabulary of an agent induces a global change of the multiagent program. For instance, if in an agent's

vocabulary the predicate P is replaced by a predicate Q , then we must modify the translation function of any of its communication partners such that the predicates that are translated to P become translated to Q .

The second option, which is the option that underlies our approach in this paper, is to have a mechanism for the *dynamic* generation of translations during execution of a multiagent system. In this approach, translations are not defined by the programmer, but instead, are generated by the system itself during its execution. Consequently, in the design of a multiagent program, a change in the vocabulary of an agent does not imply a redefinition of the other agents in the system, because no translation functions need to be defined in advance.

The dynamic construction of translations takes place during successive communication steps, in which it is based on the particular information that is exchanged by the agents. That is, the information as provided by a telling agent is translated into the vocabulary of the asking agent such that the information that is asked for is entailed. Moreover, the generation of translations takes place in a gradual fashion: during each communication step, the translations from the previous communication rounds are refined to deal with the present exchange of information.

2.1. Preliminaries

In this section, we introduce the logical machinery that is needed in defining a communication mechanism with dynamic, system-generated translations. Let us first consider the notion of a vocabulary or *signature*, which collects the symbols that are used to denote entities and relations in a domain of discourse.

DEFINITION 2.1 (Signatures). *A signature Σ is a set of function symbols f and predicate symbols P , where the 0-ary function symbols c are referred to as constants.*

For each signature Σ , we use the notation $\text{func}(\Sigma)$ to denote the set of functions in Σ , which also includes the constant symbols, and $\text{pred}(\Sigma)$ to denote the set of predicates in Σ .

The elements of a signature are referred to as *nonlogical* symbols. Together with the *logical* operations for conjunction, negation, and existential quantification, they are the constituents of first-order languages.

DEFINITION 2.2 (First-Order Languages). *Let Var be a set of variables. The set $\text{term}(\Sigma)$ of terms and the set $\text{form}(\Sigma)$ of formulas over a signature Σ are defined to be the smallest sets that satisfy:*

- $\text{Var} \subseteq \text{term}(\Sigma)$.
- $\text{True} \in \text{form}(\Sigma)$.
- If $t_1, \dots, t_n \in \text{term}(\Sigma)$, and $f \in \text{func}(\Sigma)$ is a function symbol of arity n then $f(t_1, \dots, t_n) \in \text{term}(\Sigma)$. In particular, for each constant symbol c we have $c \in \text{term}(\Sigma)$.

- If $t_1, t_2 \in \text{term}(\Sigma)$ then $(t_1 = t_2) \in \text{form}(\Sigma)$.
- If $t_1, \dots, t_n \in \text{term}(\Sigma)$ and $P \in \text{pred}(\Sigma)$ is of arity n then $P(t_1, \dots, t_n) \in \text{form}(\Sigma)$.
- If $\varphi, \psi \in \text{form}(\Sigma)$ then $\neg \varphi, \varphi \wedge \psi \in \text{form}(\Sigma)$.
- If $x \in \text{Var}$ and $\varphi \in \text{form}(\Sigma)$ then $\exists x \varphi \in \text{form}(\Sigma)$.

The other logical operators $\vee, \rightarrow, \leftrightarrow$, the universal quantifier \forall , and the formula *false* are defined in terms of the operators \neg, \wedge, \exists , and the formula *true* in the standard way.

Finally, we use the notation $\text{sig}(t)$ and $\text{sig}(\varphi)$ to denote the signature of the term t and the formula φ , respectively, which consists of all constant, function, and predicate symbols that occur in it.

A first-order language together with an entailment relation and a revision operator to update information,¹⁰ is referred to as a first-order system.

DEFINITION 2.3 (First-Order Systems). *A first-order system is a tuple $\mathcal{E} = (\Sigma, \vdash, \circ)$, where Σ is a signature,*

- $\vdash \subseteq \text{form}(\Sigma) \times \text{form}(\Sigma)$ is an entailment relation.
- $\circ: (\text{form}(\Sigma) \times \text{form}(\Sigma)) \rightarrow \text{form}(\Sigma)$ is an update operator.

Additionally, we define a formula $\varphi \in \text{form}(\Sigma)$ to be consistent if $\varphi \not\vdash \text{false}$.

In the framework, first-order systems are used by *agents* to process information. To have some idea of what the basic constituents of an agent are, we give a preliminary definition here, which is refined in Section 4.

DEFINITION 2.4 (Agent). *An agent is a tuple $\langle \mathcal{E}, S, \varphi \rangle$, where \mathcal{E} is a first-order system, S is a program describing the agent's behavior, and φ is a first-order formula from \mathcal{E} that describes the agent's belief state.*

Thus, an agent has a private first-order system \mathcal{E} to process information, which defines the signature of its information, an entailment relation to draw conclusions from its belief state, and an operator to update the belief state with newly acquired information. The behavior of the agent is given by a program S of which we discuss the details in Section 4. Finally, the third constituent of an agent is its belief state, which we assume to be represented by a first-order formula φ .

2.2. Translation Formulas

Next, we consider the syntax of the formulas that are used to represent translators. Let us first consider Boolean combinations of predicates.

DEFINITION 2.5 (Boolean Combinations). For each signature Σ , we define the set $\text{bool}(\Sigma)$ of Boolean combinations of the predicates in Σ to be the smallest set that satisfies the following conditions:

- (i) If $P \in \text{pred}(\Sigma)$ and $\mathbf{x} \in \text{Vec}$ then $P(\mathbf{x}) \in \text{bool}(\Sigma)$.
- (ii) If $\varphi, \psi \in \text{bool}(\Sigma)$ then $\neg \varphi, \varphi \wedge \psi \in \text{bool}(\Sigma)$.

Where we assume that the vector \mathbf{x} is of the same length as the arity of P .

Boolean combinations are used in the definition of translation formulas, which are defined below.

DEFINITION 2.6 (Translation Formulas). For all signatures Σ and Δ , we define the set $\text{trans}(\Sigma, \Delta)$ of translation formulas to be the smallest subset of $\text{form}(\Sigma \cup \Delta)$ that satisfies the following conditions:

- (i) $\text{True} \in \text{trans}(\Sigma, \Delta)$.
- (ii) If $\theta \in \text{trans}(\Sigma, \Delta)$, $s \in \text{term}(\Sigma)$, $\mathbf{x} \in \text{Vec}$, and $t \in \text{term}(\Delta)$ then

$$\theta \wedge \forall \mathbf{x}(s = t) \in \text{trans}(\Sigma, \Delta)$$

where \mathbf{x} denotes the (free) variables of s and t . In particular, for constants c and d we have $\theta \wedge (c = d) \in \text{trans}(\Sigma, \Delta)$.

- (iii) If $\theta \in \text{trans}(\Sigma, \Delta)$, $X \in \text{bool}(\Sigma)$, $\mathbf{x} \in \text{Vec}$, and $Y \in \text{bool}(\Delta)$ then

$$\theta \wedge \forall \mathbf{x}(X \leftrightarrow Y) \in \text{trans}(\Sigma, \Delta)$$

where \mathbf{x} consists of the (free) variables of X and Y . In particular, for propositions P and Q we have $\theta \wedge (P \leftrightarrow Q) \in \text{trans}(\Sigma, \Delta)$.

Additionally, we define $\text{trans} = \bigcup_{\Sigma, \Delta} \text{trans}(\Sigma, \Delta)$.

A translation formula $\theta \in \text{trans}(\Sigma, \Delta)$ is used to translate the symbols in the source signature Σ to symbols in the target signature Δ . If there is no confusion, we usually omit the variables from notation; for instance, we write $\forall x(P(x) \leftrightarrow Q(x))$ simply as $P \leftrightarrow Q$.

It is worth noting here that a translation formula θ can also be viewed upon as a *partial function* that maps terms $s \in \text{term}(\Sigma)$ and Boolean combinations of predicates $X \in \text{bool}(\Sigma)$ in its domain $\text{dom}(\theta)$ to terms $t \in \text{term}(\Delta)$ and Boolean combinations of predicates $Y \in \text{bool}(\Delta)$ in its range $\text{range}(\theta)$, respectively.

2.3. Translators

In this section, we consider translation formulas that act as translators between the information ψ that is asked for by an agent and the information φ that is told by an answering agent.

DEFINITION 2.7 (Translators). *Let the formula ψ from a first-order system with entailment relation \vdash be given and let φ be a formula from an arbitrary first-order system. For all translation formulas $\theta \in \text{trans}$, we define:*

$$\varphi \vdash_{\theta} \psi \quad \Leftrightarrow \quad \text{if } \varphi \text{ consistent then } \varphi \wedge \theta \text{ consistent and } \varphi \wedge \theta \vdash \psi$$

The formula θ , which is called a *translator* of φ to ψ , is used to *abduce*⁹ the information ψ from φ , that is, it lays a connection between the vocabularies of φ and ψ such that the premise φ entails the conclusion ψ . In particular, the conjunction $\varphi \wedge \theta$ can be viewed upon as an *extrapolation* of φ such that it entails the formula ψ . We use the term extrapolation here to indicate the relation with the *interpolation* property of mathematical logic.⁷ This interpolation property states that given an entailment $\varphi \vdash \psi$ there exists a formula θ that is expressed in the intersection $\text{sig}(\varphi) \cap \text{sig}(\psi)$ of the vocabularies of φ and ψ , such that $\varphi \vdash \theta$ and $\theta \vdash \psi$. The major distinction between the interpolation property and the above introduced notion of an extrapolation is twofold. First, whereas interpolations are expressed in the intersection of the vocabularies of φ and ψ , extrapolations are defined in the union of their vocabularies. Second, whereas the entailment $\varphi \vdash \psi$ is a *premise* of the interpolation, the purpose of the extrapolation of φ and ψ is to have an entailment as its *result*.

Example 2.8 (Blocks that Are Blue). Consider a first-order system comprising a predicate $\text{BlueBlock}(x)$ to denote that an individual x in the domain of discourse is a blue block, and a first-order system which comprises a predicate $\text{Blue}(x)$ to denote that an individual x is blue and a predicate $\text{Block}(x)$ to denote that an individual x is a block. We have

$$\text{BlueBlock}(\text{object861}) \vdash_{\theta} \text{Blue}(\text{o3a}) \wedge \text{Block}(\text{o3a})$$

where the translator θ equals $(\text{object861} = \text{o3a}) \wedge \text{BlueBlock} \Leftrightarrow (\text{Blue} \wedge \text{Block})$.

A property of translators is that they abstract from the syntax of information; for instance, we have for all translators θ :

$$R \vdash_{\theta} Q \quad \Leftrightarrow \quad (P \wedge \neg P) \vee R \vdash_{\theta} Q$$

This holds in general, as stated below, where we use the notation $\varphi = \varphi'$ to express that $\varphi \vdash \varphi'$ holds and also $\varphi' \vdash \varphi$.

OBSERVATION 2.9 (Abstraction from Syntax). *For all formulas φ , φ' , ψ , and ψ' that satisfy $\varphi = \varphi'$ and $\psi = \psi'$ we have*

$$\varphi \vdash_{\theta} \psi \quad \Leftrightarrow \quad \varphi' \vdash_{\theta} \psi'$$

for all translation formulas θ .

The result follows from the fact that a translator is defined in terms of the entailment relation \vdash , and not in terms of the signature of the involved information.

In general, translators are not uniquely defined, as shown in the next example.

Example 2.10 (Translators Are not Unique). Consider a situation of two agents; one of them asks for the information $R(c) \wedge S(c)$, while the other provides the information $P(c) \wedge Q(c)$. Communication is successful, if the following entailment holds

$$P(c) \wedge Q(c) \vdash_{\theta} R(c) \wedge S(c)$$

for some translator θ . This translator is not uniquely defined; for instance, we identify (among others) the following instances:

- (i) $(P \wedge Q) \leftrightarrow (R \wedge S)$.
- (ii) $P \leftrightarrow (R \wedge S)$.
- (iii) $Q \leftrightarrow (R \wedge S)$.
- (iv) $(P \leftrightarrow R) \wedge (Q \leftrightarrow S)$.
- (v) $(P \leftrightarrow S) \wedge (Q \leftrightarrow R)$.

Note that the most of the above translators are logically incomparable, except for the cases (iv) and (v), which are logically stronger versions of (i).

Without having further information it is not clear which of the translators in the above example we should prefer. Therefore, to be able to cut down the number of possible translators, it would be beneficial to possess further information about the laws that the predicates and functions of a particular vocabulary satisfy. For instance, if we would have the information that $\forall x(P(x) \rightarrow Q(x))$ holds for the telling agent, and $\forall x(S(x) \rightarrow R(x))$ holds for the asking agent, then most of the above cases can be discarded as a translator since they do not respect these laws, as explained in the next section.

3. ONTOLOGIES

To give direction to the generation of translators, it is beneficial to specify the laws that the agents' vocabularies abide by. In other words, for each agent that is involved in the communication, we should specify the corresponding ontology of its vocabulary.

The term *ontology* originally stems from philosophy, in which it refers to the study of what exists.¹⁹ In the field of artificial intelligence, the term is used for a slightly different purpose, namely, to denote a specification of a conceptualization of a particular domain.¹³ That is, in this area, an ontology provides a precise definition of which entities exist in the domain and what relations exist between them.⁶ A widely employed way of representing ontologies is through the construction of an *hierarchical classification* of concepts, in which the general and domain-independent concepts occur at the top-levels of the taxonomy, while the more specific and domain-dependent concepts appear at the lower levels.⁵

In this paper, we do not assume a particular representation mechanism for ontologies, but we simply assume them to be given by a first-order theory and in

the typical situation in which this theory is finite, simply by a first-order *formula*. However, not every first-order formula can be thought of as representing an agent's ontology. The characteristic feature of an *ontology* is that it comprises the *universal laws* that hold for the agent. In particular, an ontology makes up the agent's indisputable knowledge that is not subject to change. Examples of such universal laws include the implication $\forall x((\text{Living}(x) \vee \text{Nonliving}(x)) \rightarrow \text{Thing}(x))$ from the ontology Wordnet¹⁶ and the equality $\forall x(\text{force}(x) = \text{mass}(x) \times \text{acceleration}(x))$ from physics.

The use of having the agents' universal laws made explicit lies in the dynamic construction of translations between their vocabularies. To explain this, let us consider a communication step between an agent that sends some information φ and an agent that asks for some information ψ , where O_1 and O_2 constitute the respective and underpinning ontologies of the agents. The situation is depicted in Figure 1. The purpose of the ontologies O_1 and O_2 in the communication step is to *guide* the generation of a suitable translator θ from φ to ψ . That is, not all translators from φ to ψ are marked as appropriate candidates. The space of translators is reduced to the translators that *preserve* the laws as collected in the agent's ontologies O_1 and O_2 . Put in other words, a translator θ should induce a *partial homomorphism*, which is denoted as \lesssim_θ , between the agents' ontologies.

DEFINITION 3.1 (Partial Homomorphisms Between Ontologies). *Let the ontology O_1 from a first-order system with entailment relation \vdash_1 be given, together with an ontology O_2 from a first-order system with entailment relation \vdash_2 . For all translation formulas $\theta \in \text{trans}$, we have $O_1 \lesssim_\theta O_2$ if*

$$O_1 \wedge \theta \vdash_1 \varphi \quad \Leftrightarrow \quad O_2 \vdash_2 \varphi \quad \text{for all } \varphi \in \text{form}(\Delta)$$

where $\Delta = \text{sig}(\text{range}(\theta))$ denotes the signature of the formulas in the range of θ .

A translator θ thus induces a partial homomorphism from O_1 to O_2 if for all formulas φ that are expressed in the vocabulary of the range of θ the following holds: φ is a consequence of the conjunction of the ontology O_1 and the translator if and only if φ follows from the ontology O_2 .

Let us come back to Example 2.10. Consider the ontology O_1 that consists of the law $\forall x(P(x) \rightarrow Q(x))$ and the ontology O_2 that comprises the law

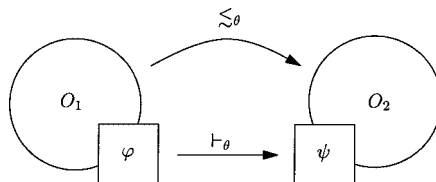


Figure 1. Translator θ as a partial homomorphism.

$\forall x(S(x) \rightarrow R(x))$. We assume that both ontologies come from a first-order system with the same entailment relation. The translator $(P \leftrightarrow R) \wedge (Q \leftrightarrow S)$ given in case (iv), which maps the predicate P to R and Q to S , is discarded as an appropriate translator, since it does not yield a partial homomorphism from O_1 to O_2 . We have

$$\begin{aligned} \forall x(P(x) \rightarrow Q(x)) \wedge (P \leftrightarrow R) \wedge (Q \leftrightarrow S) \vdash \forall x(R(x) \rightarrow S(x)) \\ \forall x(S(x) \rightarrow R(x)) \not\vdash \forall x(R(x) \rightarrow S(x)) \end{aligned}$$

Thus, if for the ontology O_1 we translate the predicate P to R and the predicate Q to S , then we can derive the law $\forall x(R(x) \rightarrow S(x))$, which however, does not hold for the ontology O_2 . The same holds for the translators given in cases (i)–(iii), only the translator $(P \leftrightarrow S) \wedge (Q \leftrightarrow R)$ of case (v) constitutes a partial homomorphism.

Example 3.2 (Agents on the Same Wavelength). Let us consider an agent that is assigned the ontology O_1 , which sends the proposition P , and additionally an agent S_2 that is assigned the ontology O_2 , which asks for the proposition Q . Communication between these agents is successful if there exists a translator θ such that $P \vdash_{\theta} Q$, where we assume that both agents employ the same entailment relation \vdash . An obvious candidate for this translator θ is the formula $P \leftrightarrow Q$. Let us examine for several instances of the ontologies O_1 and O_2 whether this translator θ induces a partial homomorphism, that is, $O_1 \lesssim_{\theta} O_2$.

- (i) $O_1 \equiv P$ and $O_2 \equiv Q$. In this case, we have that $O_1 \lesssim_{\theta} O_2$ holds, since for all φ with $\varphi \in \text{form}(Q)$ we have:

$$P \wedge (P \leftrightarrow Q) \vdash \varphi \quad \Leftrightarrow \quad Q \vdash \varphi$$

Note that we only have to check the equation for formulas φ in $\text{form}(Q)$, which are first-order formulas over the signature that consists of only the predicate Q . For instance, we do not need to consider the formula $P \wedge Q$ since $(P \wedge Q) \notin \text{form}(Q)$.

- (ii) $O_1 \equiv \text{true}$ and $O_2 \equiv \text{true}$. In this case, communication is successful as for all $\varphi \in \text{form}(Q)$ we have:

$$(P \leftrightarrow Q) \vdash \varphi \quad \Leftrightarrow \quad \text{true} \vdash \varphi$$

- (iii) $O_1 \equiv P$ and $O_2 \equiv \text{true}$. In this case, communication is not successful as θ does not constitute a partial homomorphism from O_1 to O_2 , as shown by the following facts:

$$P \wedge (P \leftrightarrow Q) \vdash Q, \quad \text{but} \quad \text{true} \not\vdash Q$$

- (iv) $O_1 \equiv P \wedge R$ and $O_2 \equiv Q \wedge \neg R$. In this case, we have $O_1 \lesssim_{\theta} O_2$, since:

$$(P \wedge R) \wedge (P \leftrightarrow Q) \vdash \varphi \quad \Leftrightarrow \quad (Q \wedge \neg R) \vdash \varphi$$

holds for all $\varphi \in \text{form}(Q)$.

Communication is successful if the agents have homomorphic ontologies with respect to the vocabulary of the exchanged information. A possible situation is the one reflected by the first case in which the agents have nonempty ontologies. Another situation is the one reflected by the second case in which both agents are completely blank, where the crucial point is that there exists a homomorphism between the ontologies with respect to the information that is involved in the communication. Put less formally, we could say that successful communication takes place in case the agents *are on the same wavelength*. Case (iii) reflects a situation in which the agents are not on the same wavelength, that is, their ontologies are not homomorphic. Finally, case (iv) shows that agents with conflicting ontologies can still successfully communicate with each other as long as the communication concerns the part of their ontologies that are not in conflict with each other.

4. PROGRAMMING WITH TRANSLATORS

To recapitulate, in this paper, we consider multiagent systems in which there is no globally shared ontology, but each agent has a private ontology. Initially, before execution of the system, no connections between the agents' individual ontologies are assumed. Instead, the connections are constructed *dynamically* during execution and moreover, established on a *demand-driven* basis. That is, only the connections between the ontologies of two agents that are needed in a communication step between them, are established, where they are used to translate the information that is provided by the sending agent to the vocabulary of the information that is asked for by the receiving agent. The connections are thus *partially* defined for only the symbols that have been referred to in communication steps. Moreover, in case several different connections are possible, one of them is chosen *arbitrarily*.

4.1. Syntax of the Programming Language

In this section, we give a short overview of the multiagent programming language ACPL, which stands for Agent Communication Programming Language.³ This language is tailored to the information-processing aspects of multiagent systems and builds upon the concurrent constraint programming paradigm²⁰ and the paradigm of communicating sequential processes.¹⁵ In the next section, we refine the operational model of the language with a mechanism for the dynamic construction of translations between vocabularies.

Let us start with identifying the actions that agents in the language ACPL can perform.

DEFINITION 4.1 (Basic Actions). *Given a first-order system \mathcal{E} the basic actions of an agent are defined as*

$$a ::= \text{query}(\varphi) \mid \text{update}(\varphi) \mid \text{tell}(c, \varphi) \mid \text{ask}(c, \varphi)$$

where φ ranges over the formulas in \mathcal{E} and c over communication channels in the set Chan .

The execution of the basic action $\text{query}(\varphi)$ consists of checking whether the belief state of the agent entails φ . Additionally, the execution of $\text{update}(\varphi)$ is given by an update of the current belief state with φ . The execution of the output action $\text{tell}(c, \varphi)$ consists of sending the information φ along the channel c . Below we see that the action has to synchronize with a corresponding input $\text{ask}(c, \psi)$, for some ψ that satisfies $\varphi \vdash_{\theta} \psi$ for some translator θ . In other words, the information φ can be sent along a channel c only if some information entailed by φ (modulo a translation), is requested. The execution of an input action $\text{ask}(c, \varphi)$, which consists of asking for the information φ along the channel c , has to synchronize with a corresponding output $\text{tell}(c, \psi)$, where $\psi \vdash_{\theta} \varphi$ for some translator θ , as we see in the operational model below.

DEFINITION 4.2 (Statements). *The behavior of an agent is described by a statement S :*

$$S ::= a \cdot S \mid S_1 + S_2 \mid S_1 \& S_2 \mid p(\bar{x}) \mid \text{skip}$$

Statements are built up from the basic actions using the following standard programming constructs: action prefixing, which is denoted by “ \cdot ”; nondeterministic choice, denoted by “ $+$ ”; internal parallelism that is modeled by interleaving, denoted by “ $\&$ ”; and (recursive) procedure calls of the form $p(\bar{x})$, where \bar{x} denotes a sequence of variables which constitute the actual parameters of the call. A procedure declaration is of the form $p(\bar{x}) :- S$, where \bar{x} denote the formal parameters of p and S constitutes its body. Additionally, skip denotes the empty statement. We note that in comparison with the definition of ACPL in Ref. 3, for simplicity, we have omitted the declarations of local variables here.

DEFINITION 4.3 (Multiagent Systems). *A multiagent system A is defined as*

$$A ::= \langle \mathcal{E}, O, D, S, \varphi \rangle \mid A_1 \mid A_2 \mid \delta_H(A)$$

where \mathcal{E} ranges over first-order systems and O and φ over formulas in \mathcal{E} . Additionally, D ranges over the sets of procedure declarations, S over statements and H over subsets of Chan .

In comparison with the preliminary Definition 2.4, an agent is additionally assigned an ontology O that comprises the agent’s universal laws. The difference between the ontology O and the information store φ , is that O constitutes the indisputable knowledge of the agent that is not subject to change, while φ collects the information that the agent currently believes to hold, but which can be refuted at a later stage of its execution. The reason for the distinction

between these two types of information, lies in the dynamic generation of translations between the agent's vocabulary and the vocabulary of its communication partners, which is based upon the agent's indisputable knowledge rather than on the information that is subject to change. The other extra constituent of an agent is a set D of procedure declarations. Moreover, as the first-order system \mathcal{E} , the ontology O and the set D of procedure declarations do not change during execution, they are usually omitted from notation, in which case the agent is denoted by $\langle S, \varphi \rangle$ rather than by $\langle \mathcal{E}, O, D, S, \varphi \rangle$.

Multiagent systems are constructed by means of the parallel composition “ \parallel ,” which is modeled by interleaving, and the encapsulation operator δ_H with $H \subseteq \text{Chan}$. The latter operator stems from the process algebra ACP ,² and is used to define local communication channels. That is, $\delta_H(A)$ denotes a multiagent system in which the communication channels in H are local and hence, cannot be used for communication with agents outside the system.

4.2. Operational Semantics

Next, we consider the operational model of the programming language, which is defined in terms of a *local* and a *global* transition system.

Agents. A local transition is of the form,

$$\langle S, \varphi \rangle \xrightarrow{l} \langle S', \psi \rangle$$

which denotes a computation step of the agent $\langle S, \varphi \rangle$ where ψ denotes the new belief state and S' the part of the program S that remains to be executed. The label l is either equal to τ in case of an *internal* computation step, that is, a computation step which consists of the execution of a basic action of the form $\text{query}(\varphi)$ or $\text{update}(\varphi)$, or l is of the form $c!\varphi$ or $c?\varphi$, in case of a communication step. First, we give the local transitions of the basic actions, where we employ the symbol E to denote successful termination.

DEFINITION 4.4 (*Transitions of Basic Actions*). *Given a first-order system $\mathcal{E} = (\Sigma, \vdash, \circ)$, we have the following transitions:*

- $\langle \text{query}(\varphi), \psi \rangle \xrightarrow{\tau} \langle E, \psi \rangle$, if $\psi \vdash \varphi$.
- $\langle \text{update}(\varphi), \psi \rangle \xrightarrow{\tau} \langle E, \psi \circ \varphi \rangle$.
- $\langle \text{tell}(c, \varphi), \psi \rangle \xrightarrow{c!\varphi} \langle E, \psi \rangle$.
- $\langle \text{ask}(c, \varphi), \psi \rangle \xrightarrow{c?\varphi} \langle E, \psi \rangle$.

First, the execution of $\text{query}(\varphi)$ succeeds if φ follows from the current belief state ψ , otherwise its execution is suspended. Additionally, the execution of the action $\text{update}(\varphi)$ amounts to an update of the belief state ψ with the information φ . Conflicts between the belief state and the new information need

to be resolved by the operator \circ . However, the implementation of such belief revision operators is beyond the scope of the current paper.

In the third transition, the label $c!\varphi$ expresses that the information φ is told along the channel c . Note that we do not assume the *sincerity condition*;¹⁸ that is, the communicated information φ is not required to follow from the agent's belief state ψ . However, a programmer can accomplish sincerity by prefixing the communication action with a test that φ indeed follows from the agent's beliefs:

$$\text{query}(\varphi) \cdot \text{tell}(c, \varphi)$$

In this case, the communication action will only be executed in case the test $\text{query}(\varphi)$ has been successfully executed first.

Finally, in the fourth transition, the label $c?\varphi$ denotes that the information φ is asked for along the channel c . Note that the information φ is not automatically added to the belief state, which however can be established by a subsequent execution of the update operator:

$$\text{ask}(c, \varphi) \cdot \text{update}(\varphi)$$

Furthermore, there are transition rules that define the operational behavior of action prefixing, internal parallel composition, nondeterministic choice, recursive procedure calls, and the empty statement.

DEFINITION 4.5 (*Transition Rule for Prefixing*). *We have*

$$\frac{\langle a, \varphi \rangle \xrightarrow{l} \langle E, \psi \rangle}{\langle a \cdot S, \varphi \rangle \xrightarrow{l} \langle S, \psi \rangle}$$

The computation step of a prefixed statement $a \cdot S$ corresponds to the execution of its prefix a . That is, the transition of the prefixed statement $a \cdot S$ is inferred from the transition of the action a , in which the label l is propagated together with the change of the belief state from φ to ψ . The statement S is identified to be the part of $a \cdot S$ that needs to be executed next.

DEFINITION 4.6 (*Transition for Internal Parallelism*). *Then,*

$$\frac{\langle S_1, \varphi \rangle \xrightarrow{l} \langle S'_1, \psi \rangle}{\langle S_1 \& S_2, \varphi \rangle \xrightarrow{l} \langle S'_1 \& S_2, \psi \rangle}$$

$$\langle S_2 \& S_1, \varphi \rangle \xrightarrow{l} \langle S_2 \& S'_1, \psi \rangle$$

A derivation rule that has two transitions below the line, is a shorthand notation for two derivation rules that each have one of these two transitions as its conclusion. The execution of a parallel statement $S \& T$ is modeled as an

interleaving of the computation steps of S and T . That is, an execution step of the composed statement $S \& T$ is given by a computation step of one of the statements S and T . Therefore, in the above transition rule, the transition of the statement S_1 induces a transition of the compound statement $S_1 \& S_2$ in which it acts as the left operand, as well as a transition of the compound statement $S_2 \& S_1$ in which it acts as the right operand. The statements $S'_1 \& S_2$ and $S_2 \& S'_1$ then denote the part of the composed statements that remains to be executed, respectively.

Additionally, with respect to termination, we define $E \& E$ to be equal to E . That is, if the two substatements of a parallel composition have terminated then the parallel composition itself has terminated as well.

DEFINITION 4.7 (*Transition Rule for Nondeterministic Choice*). *So,*

$$\frac{\langle S_1, \varphi \rangle \xrightarrow{l} \langle S'_1, \psi \rangle}{\langle S_1 + S_2, \varphi \rangle \xrightarrow{l} \langle S'_1, \psi \rangle}$$

$$\langle S_2 + S_1, \varphi \rangle \xrightarrow{l} \langle S'_1, \psi \rangle$$

The computation steps of a nondeterministic choice $S + T$ are given by the transitions of either of the statements S and T . Hence, in the transition rule above, the transition of S_1 yields a transition for the compound statement $S_1 + S_2$ as well as for the compound statement $S_2 + S_1$. The part of the nondeterministic choice that remains to be executed is given by S'_1 . Note the difference with the rule for internal parallelism in which the statement S_2 remains to be executed as well.

DEFINITION 4.8 (*Transition for Procedure Calls*). *We have*

$$\langle D, p(\mathbf{y}), \psi \rangle \xrightarrow{\tau} \langle D, S[\mathbf{y}/\mathbf{x}], \psi \rangle$$

where D contains the declaration $p(\mathbf{x}) :- S$.

The transition of a procedure call $p(\mathbf{y})$ amounts to an internal computation step in which the call is replaced by the body S of the procedure in which the actual parameters \mathbf{y} are substituted for the formal parameters \mathbf{x} , denoted by $S[\mathbf{y}/\mathbf{x}]$.

Finally, we give the transition rule for the statement `skip`.

DEFINITION 4.9 (*Transition for the Empty Statement*). *So,*

$$\langle \text{skip}, \varphi \rangle \xrightarrow{\tau} \langle E, \varphi \rangle$$

The rule reflects how the statement `skip` always succeeds and has no effects on the information store φ .

Multiagent Systems. Communication steps between agents are defined modulo a generated translation between their vocabularies. These generated translations are reused in later communication steps and therefore are stored with the communication channel along which the communication takes place. Therefore, a configuration of a multiagent system is of the form (A, T) , where A denotes a multiagent system and $T: \text{Chan} \rightarrow \text{trans}$ is a translation function. This function maps each communication channel c to a translator that has been generated in successive communication steps along c . To prevent the mixing of vocabularies, we assume that for each channel there is only one agent that sends information along the channel and there is only one basic agent that asks for information along it. Additionally, we use the notation $T\{c \mapsto \theta\}$ to denote the translation function that behaves like the function T except for the input c , for which it yields the output θ .

A *global* transition is of the form,

$$(A, T) \xrightarrow{l} (A', T')$$

which denotes a computation step of the multiagent system A that results in the new configuration (A', T') , where l indicates whether the transition involves an internal computation step, that is, $l = \tau$, or a communication, that is, $l = c! \varphi$ or $l = c? \varphi$.

DEFINITION 4.10 (*Transition Rule for Communication*). *Let O_1 and O_2 denote the ontologies of the telling and the asking agent, respectively, and let the entailment relation \vdash come from the first-order system of the asking agent. The transition rule for communication is then as*

$$\frac{A_1 \xrightarrow{c! \varphi} A'_1 \quad A_2 \xrightarrow{c? \psi} A'_2}{(A_1 \parallel A_2, T) \xrightarrow{\tau} (A'_1 \parallel A'_2, T\{c \mapsto \theta\})}$$

provided that the following hold:

- (i) $\varphi \vdash_{\theta} \psi$.
- (ii) $O_1 \lesssim_{\theta} O_2$.
- (iii) $\theta \vdash T(c)$.
- (iv) For all $\theta' \in \text{trans}$ that satisfy (i)–(iii), if $\theta \vdash \theta'$ then $\theta' \vdash \theta$.

This rule for communication reflects how the translation information $T(c)$ that has been generated during previous communication steps along the channel c is strengthened to the formula θ . This generated translation formula is such that it (i) constitutes a translator from φ to ψ and (ii) induces a partial homomorphism from the ontology O_1 to the ontology O_2 . Moreover, it is an extension of $T(c)$,

as required by (iii), that is as weak as possible (iv). The latter condition ensures that θ does not impose unnecessary connections that can complicate subsequent communication steps. That is, in the entailment $P \vdash_{\theta} Q$, we for instance prefer the instance $P \leftrightarrow Q$ to the logically stronger instance $(P \leftrightarrow Q) \wedge (R \leftrightarrow Q)$.

The transition rule for parallel execution is as follows.

DEFINITION 4.11 (*Transition Rule for Parallelism*). Then,

$$\frac{(A_1, T) \xrightarrow{l} (A'_1, T')}{(A_1 \parallel A_2, T) \xrightarrow{l} (A'_1 \parallel A_2, T')}$$

To facilitate the operational model, we assume that the parallel operator “ \parallel ” is *commutative*, which means that the transitions of the agent system $A_2 \parallel A_1$ are the same as of the system $A_1 \parallel A_2$, for all A_1 and A_2 . Additionally, we assume it to be associative, that is, the transitions of $(A_1 \parallel A_2) \parallel A_3$ are the same as of $A_1 \parallel (A_2 \parallel A_3)$. The above rule then defines that the transition of a parallel composition can be derived from the transition of one of its parallel components; that is, parallel execution is modeled by an arbitrary interleaving of the agents’ actions.

We illustrate the transition system by means of the following example.

Example 4.12 (*Derivation of a Transition*).

(i) Consider the following transition,

$$\langle \text{ask}(c, \psi) \cdot S_3, \varphi_3 \rangle \xrightarrow{c? \psi} \langle S_3, \varphi_3 \rangle$$

where the question ψ equals Blue(o3a) and the underpinning ontology is given by true. We write abbreviate this transition to: $A_3 \xrightarrow{c? \psi} A'_3$.

(ii) Additionally, we have the transition,

$$\langle \text{tell}(c, \varphi) \cdot S_1, \varphi_1 \rangle \xrightarrow{c! \varphi} \langle S_1, \varphi_1 \rangle$$

where the answer φ is given by BlueBlock(object861) and the agent’s underpinning ontology is also equal to true. We abbreviate this transition to: $A_1 \xrightarrow{c! \varphi} A'_1$.

(iii) Consider the translation function T with

$$T(c) = \text{BlueBlock} \leftrightarrow (\text{Block} \wedge \text{Blue})$$

Via (i), (ii) and the rule for communication we have

$$(A_1 \parallel A_3, T) \xrightarrow{\tau} (A'_1 \parallel A'_3, T)$$

where $T' \equiv T\{c \mapsto (\text{BlueBlock} \leftrightarrow (\text{Block} \wedge \text{Blue}) \wedge (\text{object861} = \text{o3a}))\}$.

(iv) Then from (iii) and the transition rule for parallelism, we derive

$$((A_1 \parallel A_3) \parallel A_2, T) \xrightarrow{\tau} ((A'_1 \parallel A'_3) \parallel A_2, T')$$

(v) Finally, from (iv) and the associativity and commutativity of “ \parallel ,” we derive:

$$((A_1 \parallel A_2) \parallel A_3, T) \xrightarrow{\tau} ((A_1 \parallel A_2) \parallel A_3, T')$$

Finally, to define the operational semantics of the encapsulation operator, we extend the syntax of the programming language with a construct of the form $\delta_c^\theta(A)$, which stores the translation formula θ that has been generated along the local communication channel c . Additionally, using this notation, a multi-agent system $\delta_H(A)$ where $H = \{c_1, \dots, c_n\}$ is mimicked by the construct $\delta_{c_1}^{\text{true}}(\dots \delta_{c_n}^{\text{true}}(A) \dots)$.

DEFINITION 4.13 (*Transition Rule for Encapsulation*). Now,

$$\frac{(A, T\{c \mapsto \theta\}) \xrightarrow{l} (A', T')}{(\delta_c^\theta(A), T) \xrightarrow{l} (\delta_c^{T'(c)}(A'), T'\{c \mapsto T(c)\})} \text{ if } c \notin \text{chan}(l)$$

where $\text{chan}(\tau) = \emptyset$ and $\text{chan}(c!\varphi) = \text{chan}(c?\psi) = \{c\}$.

A transition of the agent system $\delta_c^\theta(A)$ with a translation function T can be derived from a transition of A with the translation function $T\{c \mapsto \theta\}$ in which the translation formula of the global channel c is overwritten with that of the local channel c . The resulting translation function T' of the system A also denotes the result of $\delta_c^\theta(A)$ except for the translation formula for the global channel c for which the original translation formula $T(c)$ is restored. The translation formula $T'(c)$ for the local channel c is stored with the local channel construct for later use.

Let us end this section with a final example.

Example 4.14 (*Language Games*). We mention the following problem: given two ontologies O_1 and O_2 and a partial homomorphism θ from O_1 to O_2 , construct an agent $A_1 = \langle S_1, \varphi_1 \rangle$ that is assigned the ontology O_1 and an agent $A_2 = \langle S_2, \varphi_2 \rangle$ that is assigned the ontology O_2 , such that the agent system $(A_1 \parallel A_2, T)$ with $T(c) = \text{true}$ generates the above translation formula θ via communication along the channel c ; that is,

$$(A_1 \parallel A_2, T) \xRightarrow{\tau} (E, T') \text{ such that } T'(c) = \theta$$

where $\xRightarrow{\tau}$ denotes the reflexive, transitive closure of $\xrightarrow{\tau}$. This program can be viewed upon as implementing a *language game* between the agents in which the aim is to come to a *mutual understanding* of the individual vocabularies through the generation of a suitable translator θ between them. To illustrate the problem, we consider the following instance:

Let $O_1 \equiv P \wedge Q$, $O_2 \equiv R \wedge S$, and $\theta \equiv (P \leftrightarrow R \wedge Q \leftrightarrow S)$, then we have

$$S_1 \equiv \text{tell}(c, P) \cdot \text{tell}(c, Q)$$

$$S_2 \equiv \text{ask}(c, R) \cdot \text{ask}(c, S)$$

Note that for instance the following is not a solution,

$$S_1 \equiv \text{tell}(c, P \wedge Q)$$

$$S_2 \equiv \text{ask}(c, R \wedge S)$$

since this leads to $\theta \equiv (P \wedge Q) \leftrightarrow (R \wedge S)$.

5. CONCLUSIONS

In this section, we discuss several related approaches in which translations between vocabularies play a central role, and we address several issues for future research.

Related Work. First of all, there is the work of Antoniou and Kehagias on the refinement of ontologies.¹ In their framework, agents have a private ontology that can be translated into a globally shared ontology. A central issue is that of ontology change: if an agent's private ontology is modified, then in general, this induces a modification of the global ontology as well. One of their observations is that if a local ontology is changed into a new ontology that is a *conservative extension*, then the global ontology can be *refined* in such a way that the translators of the other agents to the global ontology remain the same. In other words, in such case the ontology revision remains local. A major difference with our framework lies in our focus on translators that are dynamically constructed during execution of a multiagent system. These translators are based upon the particular information that the agents exchange, instead of being predefined mappings between ontologies like in framework of Antoniou and Kehagias.

Second, we mention the language KIF, which stands for knowledge interchange format.¹¹ It is a formalism that is based on the language of first-order logic with some additional features to enhance its expressiveness. The purpose of this language is to constitute a standard representation language for the definition of ontologies. That is, ontologies defined in this language are portable over different systems. This is achieved through a mechanism called Ontolingua, which translates ontologies that are defined in KIF to several more specialized and implemented representation frameworks.¹⁴

In the field of contextual reasoning, a framework has been developed for the modeling of interactions between logical theories.¹² This framework comprises a mechanism of *bridge rules*, which are used to translate information from one context to information in another context. Similar bridge rules can be found in the multiagent specification language CDDL in which they are used in descriptions of dialogues between agents.¹⁷

In the DESIRE multiagent framework, we can find an analogous translation mechanism.⁴ In this framework, agents interact with each other via the exchange of information along a network of interconnecting links. A link is a unidirectional communication channel that interconnects a sending and a receiving agent. It is assigned a subsignature of the sender as its domain and a subsignature of the receiver as its co-domain. Additionally, it is assigned a translation table that specifies what information in the domain is to be translated to what information in the co-domain. In case the sending agent concludes some information that is expressed in the domain signature, this information is translated, using the associated translation table, to information in the co-domain signature and subsequently is added to the knowledge base of the receiving agent of the link.

Name-space context graphs also constitute a means to manage the symbols employed by communicating agents.²¹ In this framework, each agent belongs to some context and makes use of the symbols that are associated with this context. By default, the same symbols in different contexts denote different concepts, but the framework covers a mechanism to inherit names from other contexts as well as a mechanism to express equalities between symbols from different contexts. Such equalities are used in translations of the symbols employed by a sending agent to those associated with the context of the receiving agent.

Future Research. The use of dynamically generated translators gives rise to a new study of the *deadlock behavior* of the programming framework ACPL. This is due to the fact that in general, in the dynamic generation of translators different possibilities exist from which only one translator can be chosen. During further execution of the program a chosen translator may prove to be a wrong choice, namely, when we run into a deadlock situation in which the generated translator cannot be faithfully extended without becoming inconsistent. An interesting issue is then to develop a compositional semantics for the language that takes account of its deadlock behavior, which is additionally fully abstract²³ with respect to an appropriate notion of observable behavior.

Another topic of future research, is the development of algorithms for the construction of translators. An obvious algorithm is a brute-force approach in which, given the told formula φ and the asked formula ψ , all translation formulas θ in the set $\text{trans}(\text{sig}(\varphi), \text{sig}(\psi))$ are considered, of which there are finitely many logically distinct ones. Subsequently, it is checked which of them constitute a translator from φ to ψ . Probably, it is possible to come up with algorithms that are more efficient than this brute-force approach.

Finally, we mention the multiagent systems that are studied in the research on the *origins of language*.²² Systems in this area consist of a physical environment and several operating robotic agents each of which has a private vocabulary to denote objects in the environment. The idea is that the agents communicate with each other to develop a shared vocabulary. Communication takes place in the form of a language game in which the agents identify an object in the environment and aim to match the words that they use for this object. This matching procedure is done via a form of reinforcement learning. An interesting

issue for future research is to study to what extent our general framework can be used to model and to analyze the experiments as performed in this area.

References

1. Antoniou G, Kehagias A. A note on the refinement of ontologies. *Int J Intell Syst* 2000;15(7):623–632.
2. Bergstra JA, Klop JW. Process algebra for synchronous communication. *Inf Contr* 1984;60:109–137.
3. de Boer FS, van Eijk RM, van der Hoek W, Meyer J-JCh. Failure semantics for the exchange of information in multi-agent systems. In: Palamidessi C, editor. *Proceedings of the Eleventh International Conference on Concurrency Theory (CONCUR 2000)*. Lecture Notes in Computer Science. Heidelberg: Springer-Verlag; 2000. Vol 1877, p 214–228.
4. Brazier F, Dunin-Keplicz B, Jennings N, Treur J. Formal specification of multi-agent systems: a real-world case. In: *Proceedings of the International Conference on Multi-Agent Systems (ICMAS'95)*. Cambridge, MA: MIT Press; 1995. p 25–32.
5. Chandrasekaran B, Josephson JR, Benjamins VR. What are ontologies and why do we need them? *IEEE Intell Syst* 1999;14(1):20–26.
6. Chen PP. The entity-relationship model—toward a unified view of data. *ACM Trans Database Syst* 1976;1(1):9–36.
7. Craig W. Three uses of the herbrand-gentzen theory in relating model theory and proof theory. *J Symbolic Logic* 1957;22:269–285.
8. van Eijk RM, de Boer FS, van der Hoek W, Meyer J-JCh. Systems of communicating agents. In: Prade H, editor. *Proceedings of the Thirteenth Biennial European Conference on Artificial Intelligence (ECAI'98)*. Chichester, UK: John Wiley & Sons; 1998. p 293–297.
9. Flach PA, Kakas AC, editors. *Abduction and induction: Essays on their relation and integration*. Dordrecht, The Netherlands: Kluwer Academic; 2000.
10. Gärdenfors P. *Knowledge in flux: Modelling the dynamics of epistemic states*. Cambridge, MA: Bradford Books, MIT; 1988.
11. Genesereth MR, Fikes RE. *Knowledge Interchange Format, Version 3.0 Reference Manual*. Technical Report Logic-92-1, Computer Science Department, Stanford University, 1992.
12. Giunchiglia F, Serafini L. Multilanguage hierarchical logics (or: How we can do without modal logics). *Artif Intell* 1994;64:29–70.
13. Gruber TR. Toward principles for the design of ontologies used for knowledge sharing. In: Guarino N, Poli R, editors. *Formal ontology in conceptual analysis and knowledge representation*. Dordrecht/Norwell, MA: Kluwer Academic; 1993.
14. Gruber TR. A translation approach to portable ontology specifications. *Knowledge Acquisition* 1993;5(2):199–220.
15. Hoare CAR. Communicating sequential processes. *Commun ACM* 1978;21(8):666–667.
16. Miller GA. Wordnet: An online lexical database. *Int J Lexicography* 1990;3(4):235–312.
17. Noriega P, Sierra C. Towards layered dialogical agents. In: Müller JP, Wooldridge M, Jennings NR, editors. *Intelligent agents III—Agent theories, architectures, and languages (ATAL'96)*. Lecture Notes in Artificial Intelligence. Berlin/New York: Springer-Verlag; 1997. Vol 1193, p 173–188.
18. Pitt J, Mamdani A. Some remarks on the semantics of Foundation for Intelligent Physical Agents (FIPA) agent communication language. *Autonomous Agents Multi-Agent Syst* 1999;2(4):333–356.
19. Quine WVO. *On what there is*. In: *From a logical point of view*. Cambridge, MA: Harvard Univ. Press; 1948.

20. Saraswat VA, Rinard M. Concurrent constraint programming. In: Proceedings of the Seventeenth Association for Computing Machinery (ACM) Symposium on Principles of Programming Languages (POPL'90), 1990, p 232–245.
21. Singh N, Tawakol O, Genesereth M. A name-space context graph for multi-context, multi-agent systems. In: Proceedings of the 1995 American Association for Artificial Intelligence (AAAI) Fall Symposium, Boston, MA, 1995.
22. Steels L. The origins of ontologies and communication conventions in multi-agent systems. *Autonomous Agents Multi-Agent Syst* 1998;1(2):169–194.
23. Stoughton A. Fully abstract models of programming languages. *Research Notes in Theoretical Computer Science*. London: Pitman; 1988.