Epistemic Logic and Explicit Knowledge in Distributed Programming

(Short Paper)

Andreas Witzel Jonathan A. Zvesper
ILLC, University of Amsterdam, Plantage Muidergracht 24, 1018TV Amsterdam
and CWI, Kruislaan 413, 1098SJ Amsterdam, The Netherlands
{awitzel,jonathan}@illc.uva.nl

ABSTRACT

In this paper we propose an explicit form of knowledge-based programming. Our initial motivation is the distributed implementation of game-theoretical algorithms, but we abstract away from the game-theoretical details and describe a general scenario, where a group of agents each have some initially private bits of information which they can then communicate to each other. We draw on existing literature to give a formal model using modal logic to represent the knowledge of the agents as well as how that knowledge changes as they communicate. We sketch an implementation which enables processes in a distributed system to explicitly evaluate knowledge formulae. Then we prove that the implementation captures the formal model, and therefore correctly reflects the general scenario. Finally we look at how our approach lends itself to generalisations, and discuss application perspectives.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent systems*

General Terms

Algorithms, Economics, Theory, Verification

Keywords

Explicit knowledge programming

1. INTRODUCTION

In knowledge-based programming [7, 3], knowledge operators are used as a conceptual tool for the specificator of an algorithm. Knowledge operators do not, in such an approach, appear explicitly in the resulting so-called *standard* program. Rather, that program is proved to behave equivalently to the knowledge-based specification. The knowledge ascribed to the processes is never actually computed by the processes or otherwise made explicitly accessible to them.

Cite as: Epistemic Logic and Explicit Knowledge in Distributed Programming (Short Paper), Andreas Witzel and Jonathan Zvesper, *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Padgham, Parkes, Müller and Parsons (eds.), May, 12-16., 2008, Estoril, Portugal, pp. XXX-XXX.

We are motivated by the distributed implementation of game-theoretical algorithms that involve (or at the very least are facilitated by) *explicit* reasoning about knowledge and the effects of communication. Thus we propose, in contrast to the existing approach of knowledge-based programming, to make knowledge operators available for use in programs by giving algorithms that evaluate knowledge formulae.

Several rich platforms for multi-agent programming have been proposed (see e.g. [5] for a recent overview). While these often do provide for explicit knowledge operators, we are aware of no instances in which *higher-order* knowledge plays a role. Higher-order knowledge is knowledge about knowledge, as in 'I know that you know that p'.

In situations of interaction, this higher-order knowledge can be as important as factual knowledge for describing an agent's environment. The specific interactive setting we have in mind involves iterated elimination of dominated strategies [11] in pre-Bayesian games [1]. However, we abstract from these details in this short paper, to consider a more general scenario which still illustrates the underlying issues of communication and knowledge.

We start by designing a model of the epistemic situation of the agents, which also must represent the changes brought about by communication. Our model will be a Kripke model with some implicit temporal structure. We draw on existing literature [8, 12] for inspiration and philosophical grounding.

We introduce our general communication scenario and formal model in Section 2, describing several clear assumptions that we make. Some of these guide us in our choice of model, while others are added to simplify the implementation. Note that these assumptions serve to keep our illustrative model and implementation simple, and are not inherent in the idea of explicit knowledge programming. In Section 3 we describe the relevant parts of the implementation, i.e. the algorithm for updating and evaluating knowledge. We prove that these correspond to the relevant parts of the formal model by showing that the knowledge computed by a process essentially coincides with the knowledge ascribed to the agent by the model. Section 4 offers conclusions and perspectives for extensions and applications.

2. MODEL

We consider a scenario in which agents each know the value of some distinct bits. That is: each agent i has a set X_i of (names of) bits $x_{i,a}$ whose values he knows, the X_i's are disjoint, and initially no agent other than i knows the value of any bits in X_i. We also suppose that all of this is common knowledge[1]. We will denote by X the set of all bits in question $\bigcup_{i \in N} X_i$. Agents can then communicate about the values of bits in X.

Within this general paradigm we consider the specific case in which a number of additional assumptions are imposed. These assumptions make the model and the implementation more straightforward, while still allowing us to illustrate the connection between the model and the implementation.

The assumptions are the following:

1. The communication is private, and the agents do not have access to a 'global clock', and so cannot know, at any point, whether any actions have taken places that do not involve them.
2. The communication is synchronous, and takes place between two agents who can identify each other. A communicated message thus becomes common knowledge between the two involved agents.
3. All communication is of the following type: Agent i tells agent j (truthfully) the value of some $x_{i,a} \in X_i$.
4. The values of the bits do not change over time.
5. These assumptions are common knowledge.

The first two assumptions are inherent in our choice to base our implementation on the formalism of Communicating Sequential Processes (CSP, [10]). The third assumption is a restriction on the kinds of messages that we will model, and constitutes a natural place for future generalisations. The last assumption, intuitively speaking, is a consequence of the fact that anything that holds everywhere in our model is common knowledge. None of these assumptions are crucially tied to the basic idea of explicit knowledge programming that we are presenting here.

A natural interactive model for knowledge, described for example in [8], is given by (multi-agent) Kripke models. A ***Kripke model*** consists of a set of "worlds" Ω along with, for each agent i, a relation $R_i \subseteq \Omega \times \Omega$ stating which worlds are indistinguishable for i, and a function V which gives, for each bit $x_{i,a}$, the set of worlds at which $x_{i,a}$ is 1.

Consider the standard multi-modal language $\mathcal{L}_N$ (cf. [4]):

$$\varphi ::= x_{i,a} \mid \varphi \wedge \varphi \mid \neg\varphi \mid \Box_i \varphi$$

This language can be evaluated using the standard modal semantics. The key semantic clause is that for the knowledge modality: $\omega \vDash \Box_i \varphi \overset{\text{df}}{\Leftrightarrow} \forall \omega' (\omega R_i \omega' \Rightarrow \omega' \vDash \varphi)$ for $\omega \in \Omega$.

In order to define the Kripke model that corresponds to the epistemic situation of the agents in our scenario, we will make use of the notion of a *protocol*.[2] Given a set of atomic events Σ, we denote by Σ^* the finite sequences of elements of Σ. We will call these ***histories***. A ***protocol*** over a set Σ is just a set of histories, i.e. a subset of Σ^*. A protocol specifies which sequences of events could in principle happen, that is, which events are 'legal' in a given context. Now, in our scenario the events involving the agents are private messages from some agent i informing another agent j of the value of some bit $x_{i,a}$. Notice that there are in some sense two different events here: the one in which $x_{i,a} = 1$ and the one in which $x_{i,a} = 0$. We will denote those events $e_a^{i \to j}$ and $e_{\neg a}^{i \to j}$ respectively, and use $e_{*a}^{i \to j}$ when we mean either of these. Those events will be legal in different contexts, depending on whether they are truthful or not.

[1]A fact p is common knowledge in a group just if everyone knows that p, everone knows that everyone knows that p, and so on ad infinitum

[2]Our presentation is technically similar to [12], and is in the same spirit. See also [8] for a general discussion of the kind of history-based structures we use as a model.

Thus the protocol depends on what is true, so we will introduce an additional type of event, which is not in the control of any agent in our scenario, but can be thought of as a 'move of nature'. In such a move $Y \subseteq X$, the bits Y whose values are 1 are chosen, and all others get the value 0. It should be clear that (i) the fact that messages are truthful imposes the restriction that $e_a^{i \to j}$ can occur just if $x_{i,a} = 1$ (and $e_{\neg a}^{i \to j}$ just if $x_{i,a} = 0$). We also know that (ii) the values of bits are fixed. It turns out that these two facts (i) and (ii) are enough to define a protocol which represents the scenario we have described. Let Σ_Y be the set of events $e_{*a}^{i \to j}$ that are compatible with Y; i.e. $\Sigma_Y \overset{\text{df}}{=} \{e_a^{i \to j} \mid x_{i,a} \in Y\} \cup \{e_{\neg a}^{i \to j} \mid x_{i,a} \notin Y\}$. Then we can define the protocol $\mathcal{H}$ as follows:

$$\mathcal{H} \overset{\text{df}}{=} \{\varepsilon\} \cup \{(Y, e_1, \ldots, e_l) \mid Y \subseteq X \ \wedge \ \forall 1 \leq k \leq l, e_k \in \Sigma_Y\}$$

where ε denotes the empty sequence. Again inspired by [12], we will consider the ***local events*** E_i of each agent i. In our case $E_i = \{e_{*a}^{i \to j} \mid x_{i,a} \in X_i\} \cup \{e_{*a}^{j \to i} \mid x_{j,a} \in X_j\}$: those events in which i participates. Of all other communication events, i is completely ignorant: she is not unaware that other events *might* be taking place, but they would all be taking place 'behind her back'. Using local events we define ***local histories***: $\lambda_i : \Sigma^* \to \Sigma^*$ is the projection defined recursively with $\lambda_i(\varepsilon) \overset{\text{df}}{=} \varepsilon$, $\lambda_i(Y) \overset{\text{df}}{=} X_i \cap Y$, and

$$\lambda_i(h, e) \overset{\text{df}}{=} \begin{cases} (\lambda_i(h), e) & \text{if } e \in E_i \\ \lambda_i(h) & \text{otherwise} \end{cases}$$

We can now define the Kripke model that captures the scenario we want to model. We let $\mathcal{M} = (\mathcal{H}, R_i, V)_{i \in N}$, where

- $h R_i h' \overset{\text{df}}{\Leftrightarrow} \lambda_i(h) = \lambda_i(h')$;
- $V(x_{i,a}) \overset{\text{df}}{=} \{(Y, h) \in \mathcal{H} \mid x_{i,a} \in Y\}$.

We take the philosophical grounding of Kripke models to be sufficient to claim that the world (Y, h) captures the intuitively described scenario, in the sense that any formula φ holds at (Y, h) in the model $\mathcal{M}$ just if it intuitively should.

In the next section we will describe part of the implementation and show that it, in turn, captures this formal model.

3. IMPLEMENTATION

We implement each agent $i \in N$ by a process with the same name i and mark its local variables by superscript i. Each process $i \in N$ has the following local variables:

- $x_{j,a}^i$ for $j \in N, x_{j,a} \in X_j$, holding ¿ or the value of $x_{j,a}$,
- $comm_{j,a}^i$ for $j \in N \setminus \{i\}, x_{i,a} \in X_i$, holding $\top$ or $\bot$.

The variables are initialised as follows:

$$x_{j,a}^i := \begin{cases} x_{i,a} & \text{if } j = i \\ ¿ & \text{otherwise} \end{cases}$$

$$comm_{j,a}^i := \bot.$$

To reason about this implementation, it will be useful to talk about ***program states***, which are tuples consisting of all the variables of each process. We use the letter ρ to refer to program states, and will think of them as functions giving values to variables, so we can write for example $\rho(x_{j,a}^i) = 1$.

Messages are of the form $m_{*a}^{i\to j}$, where i informs j about the value of $x_{i,a}$. Upon exchange of $m_{*a}^{i\to j}$, the involved processes update some of their variables as follows:

$$x_{i,a}^{j} := \begin{cases} 0 & \text{if } `*a' = `\neg a' \\ 1 & \text{if } `*a' = `a' \end{cases}$$

$$comm_{j,a}^{i} := \top \ .$$

A message $m_{*a}^{i\to j}$ is ***truthful*** in a program state ρ just if $`*a' = `\neg a' \Leftrightarrow \rho(x_{i,a}^{i}) = 0$. In accordance with our assumptions, we only allow truthful messages.

An essential part of the implementation is the evaluation of ***knowledge formulae***. These are formulae from a limited modal language $\mathcal{L}_K$, defined as follows:

$$\varphi ::= x_{i,a} = 1 \mid x_{i,a} = 0 \mid K_i\varphi$$

The intended meaning of $K_i\varphi$ is that process i knows φ. Given an arbitrary sequence $s = (i_1, \ldots, i_l)$ of agents from N, we write $K_s\varphi$ to abbreviate $K_{i_1} \ldots K_{i_l}\varphi$, and we define $s_{\{\}} := \{i_1, \ldots, i_l\}$.

Each process i can evaluate any $\mathcal{L}_K$-formula $K_s x_{j,a} = v$ as follows (note that possibly $j = i$):

$$\begin{cases} x_{j,a}^{i} = v & \text{if } s_{\{\}} \subseteq \{i,j\} \\ comm_{k,a}^{i} \wedge x_{i,a}^{i} = v & \text{if } s_{\{\}} = \{i,k\} \text{ and } k \neq i = j \\ \bot & \text{otherwise.} \end{cases}$$

We do not have space to give the detailed intuitions behind this case distinction, instead we now give a more rigorous demonstration to the effect that the implementation is correct. The aim of the implementation is *to make the implicit knowledge of the processes explicitly available to them.*

The remainder of this section is devoted to showing that the implementation does this in a coherent way. That is, we will demonstrate that the program implements the epistemic model which we described in Section 2.

If the program state is ρ then we denote by $\kappa_i(\rho)$ those $\mathcal{L}_K$-formulae that i evaluates to be $\top$.

What we want to show is that at any point in any run reflecting a program execution, all knowledge formulae are evaluated with the same result as in a certain corresponding world of the Kripke model. We reach this world by mapping any initial program state to the intuitively corresponding world and then, for any sequence of truthful messages, applying the equivalent sequence of events. We then show that, in the obtained program state ρ, the knowledge formulae $\varphi \in \kappa_i(\rho)$ which i evaluates as true are exactly those for which $\mathcal{M}, \omega \models \Box_i\varphi$ in the corresponding world ω, that is, that i knows in ω.

An ***initialisation*** is the set $\{x_{i,a} \in X \mid x_{i,a} = 1\}$. Notice that this is the same as a 'move of nature', as described in Section 2. A ***run*** σ is a sequence $(Y, m_1, \ldots, m_k)$, where Y is an initialisation and each m_l is a truthful message. Any run uniquely determines on the one hand a program state ρ_σ, and on the other hand a world ω_σ. The program state is obtained according to the algorithms described above. The world is obtained straightforwardly, by translating each message $m_{*a}^{i\to j}$ to an event $e_{*a}^{i\to j}$. Thus the run $(Y, m_{*a_1}^{i_1\to j_1}, \ldots, m_{*a_k}^{i_k\to j_k})$ is mapped to the world $(Y, e_{*a_1}^{i_1\to j_1}, \ldots, e_{*a_k}^{i_k\to j_k})$.

Given some world ω in a model $\mathcal{M}$, let $Th_i(\omega) = \{\varphi \in \mathcal{L}_N \mid \mathcal{M}, \omega \vDash \Box_i\varphi\}$ be the set of $\mathcal{L}_N$-formulae which i knows to hold. Now such a set clearly also defines a set of $\mathcal{L}_K$-formulae. Specifically, for any set Γ of $\mathcal{L}_N$-formulae, let Γ^K be the result of replacing, for all $j \in N$, all instances of $\Box_j$ with K_j and then restricting to $\mathcal{L}_K$. We are now ready to state formally that the computed knowledge is correct:

Proposition. $Th_i^K(\omega_\sigma) = \kappa_i(\rho_\sigma)$.

PROOF SKETCH. Take any formula $\varphi \in \mathcal{L}_K$; then φ is of the form $K_s x_{j,a} = v$; without loss of generality consider the case where $v = 1$. First show that if the cardinality of $s_{\{\}} \cup \{i,j\}$ is greater than 2, then $\varphi \notin \kappa_i(\rho_\sigma)$, *and* $\varphi \notin Th_i^K(\omega_\sigma)$. Then we consider the various possible cases when $\#(s_{\{\}} \cup \{i,j\}) \leq 2$. That is, if $s_{\{\}} \cup \{i,j\} = \{i,k\}$, we have one of the following (a) $i = j = k$; (b) $i \neq j = k$; or (c) $i = j \neq k$, for which for the proofs are similar. For example, in case (b) we show that:

$$\varphi \in \kappa_i(\rho_\sigma) \Leftrightarrow \rho_\sigma(x_{j,a}^{i}) = 1 \Leftrightarrow$$
$$\sigma = (\ldots, m_a^{j\to i}, \ldots) \Leftrightarrow \omega_\sigma = (\ldots, e_a^{j\to i}, \ldots) \overset{\cdots}{\Leftrightarrow}$$
$$\omega_\sigma \models \Box_i x_{j,a} \overset{\cdots}{\Leftrightarrow} \omega_\sigma \models \Box_i\Box_s x_{j,a} \Leftrightarrow$$
$$K_s x_{j,a} = 1 \in Th_i^K(\omega_\sigma).$$

The more involved parts, including the dotted equivalences ($\overset{\cdots}{\Leftrightarrow}$), use some standard modal logic reasoning. $\Box$

4. CONCLUSIONS AND OUTLOOK

We have presented an approach that enables processes to explicitly use knowledge formulae in order to reason about their environment, which includes each other's knowledge. We have illustrated this approach sketching a simple implementation, and have shown the correctness of the implementation with respect to an epistemic model.

The simplicity of the implementation is mainly due to the clear assumptions that we stated in Section 2. For example, the fact that the order in which messages are sent does not matter simplifies the mechanism needed to keep track of them. The facts that bits do not change over time and that communication is synchronous removes the necessity of temporal reasoning. Furthermore, we consider a very restricted kind of knowledge formulae. Remember that *these restrictions are not inherent to explicit knowledge programming*, but serve to keep the presentation simple.

Still, the level of generality which we describe already allows for concrete and interesting applications. Our initial motivation, mentioned in Section 1, is a game-theoretical setting where each player initially only knows his own payoffs. In order to eliminate dominated strategies, players also need to reason about which strategies other players will eliminate, and to that end they need to reason about which strategies other players will think yet other players will eliminate, etc. For this kind of reasoning it is sufficient to evaluate knowledge formulae of exactly the form we have considered (where the bits concern payoffs).

Algorithms like this can now be implemented in a general way using knowledge formulae. When more types of messages are allowed or other assumptions are lifted, only the knowledge evaluation function needs to be extended, while the actual algorithms can stay unchanged.

More generally, our approach suggests a different view on program synthesis for knowledge-based programming. Instead of following the suggestion from [7] to synthesise a complete standard program from a knowledge-based one, one would implement a module containing the knowledge evaluation function and the epistemic operations (for example, the effects of communication). The program itself would remain knowledge-based.

In order to achieve this kind of program synthesis, it is necessary to have a natural and flexible model. We think that Kripke structures lend themselves to this task. However, we do not claim that the model we have proposed is necessarily natural or flexible enough. We are aware of the parallels with Interpreted Systems [8], and also the connections with Dynamic Epistemic Logic (DEL, [2]), and we believe that both of these approaches are of relevance in working out such a method of program specification and synthesis. For example, it is straightforward to give a so-called 'action model' from DEL which can be used to generate the model in Section 2; thus variations of the model could be designed using the modular approach of DEL.

Another way to extend our results would be to enrich the language $\mathcal{L}_K$ whose formulae the processes evaluate. Certain extensions could be straightforwardly implemented, for example allowing for negated K operators; others require more changes. The language available to the processes should be tailored for the specific application. In our motivating setting, it suffices to be able to evaluate $\mathcal{L}_K$-formulae.

A feature that is notably absent from our model is that of questions, i.e. requests for information rather than just exchange of information. These requests might themselves be viewed as carrying information, and so in many scenarios (including the game-theoretical scenario which motivates our model), their epistemic effects can be significant. In a more realistic communication setting there will also be strategic aspects to communication [9], especially if one wants to lift the truthfulness assumption.

We will now briefly sketch two intriguing long-term application perspectives. The first concerns the area of multi-agent planning. Most existing approaches (see e.g. [6]) fall into two categories. In one, agents cooperate and so the focus is on how to optimally distribute the steps necessary to reach their goals. In the other, agents assume that other agents (or the environment) always act in the worst possible way, which game-theoretically corresponds to a zero-sum game. However, in the more general case, preferences will neither be totally aligned nor totally opposing. Rational agents should be able to model other agents, figure out how they will act and react, and take this into account in their plans. Higher-order reasoning about knowledge and preferences is crucial for this *game-theoretical* planning. Having such facilities on the level of the programming language may greatly ease implementation of these algorithms, yet as mentioned in Section 1, existing multi-agent platforms usually focus on factual knowledge.

The other application perspective is to computer games. While the game-theoretical planning described above could generally be of interest for computer-controlled opponents in strategy games, there is a much broader range of possible applications. Just to pick an example, any computer game which aims at creating a convincing social interaction needs to simulate the kinds of higher-order reasoning which go on in real-life social interaction. Concretely, if a computer-controlled character C in a role-playing game only offers information to the (human) player P of which he cannot deduce that P knows that C knows that P knows it, the player will perceive the scene to be much more natural than if C always stupidly repeats the same phrases. While computer games have implemented this kind of intelligence to certain degrees, using higher-order knowledge formulae directly in the algorithms would, again, ease the task of the programmer and enhance the possibilities of behaviour of computer-controlled characters.

While the kind of mechanisms we propose could be implemented ad hoc, it seems natural to provide higher-order knowledge operators as an abstraction layer, simplifying the remaining program and increasing modularity. We intend our implementation to be a simple but clear foundation which will, due to the explicit assumptions we have made and the formal model we have constructed, enable us to see what extensions are feasible and how they can be realised.

5. ACKNOWLEDGEMENTS

We would like to thank our supervisor Krzysztof Apt for his inspiring ideas, support and helpful comments.

This research was supported by a Marie Curie Early Stage Research fellowship in the project GLoRiClass (MEST-CT-2005-020841).

6. REFERENCES

[1] I. Ashlagi, D. Monderer, and M. Tennenholtz. Resource selection games with unknown number of players. In *5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-06)*, pages 819–825, Hakodate, Japan, 2006.

[2] A. Baltag, L. S. Moss, and S. Solecki. The logic of public announcements, common knowledge and private suspicions. Technical Report SEN-R9922, Centrum voor Wiskunde en Informatica, 1999.

[3] M. Bickford, R. C. Constable, J. Y. Halpern, and S. Petride. *Knowledge-Based Synthesis of Distributed Systems Using Event Structures*, volume 3452 of *Lecture Notes in Computer Science*, pages 449–465. Springer, 2005.

[4] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Cambridge University Press, 2001.

[5] R. H. Bordini, L. Braubach, M. Dastani, A. E. F. Seghrouchni, J. J. Gomez-Sanz, J. Leite, G. O'Hare, A. Pokahr, and A. Ricci. A survey of programming languages and platforms for multi-agent systems. *Informatica*, 30:33–44, 2006.

[6] B. J. Clement, editor. *Workshop on Multiagent Planning and Scheduling, The 15th International Conference on Automated Planning and Scheduling (ICAPS-05)*, June 2005.

[7] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. Knowledge-based programs. *Distributed Computing*, 10:199–225, July 1997.

[8] R. Fagin, J. Y. Halpern, M. Y. Vardi, and Y. Moses. *Reasoning about knowledge*. MIT Press, 1995.

[9] J. Gerbrandy. Communication strategies in games. *Journal of Applied Non-Classical Logics*, 17:197–211, 2007.

[10] C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 21:666–677, 1978.

[11] M. J. Osborne and A. Rubinstein. *A Course in Game Theory*. MIT Press, 1994.

[12] R. Parikh and R. Ramanujam. A knowledge based semantics of messages. *Journal of Logic, Language and Information*, 12:453–467, 2003.