An Ontology for Mobile Agents in the Context of Formal Verification

Paulo Salem da Silva and Ana Cristina Vieira de Melo

University of São Paulo, Department of Computer Science, São Paulo – Brazil
salem@linux.ime.usp.br, acvm@ime.usp.br

Abstract. The increasing interest of developing and formally studying mobile agents has taken place in the last decade. Several formalisms and tools have been created to aid this enterprise. However, tools are mostly developed in isolation and, therefore, are hard to use together. The present work is an attempt to make such integration easier, through the provision of a common 'language' – an ontology – for verification tools reasoning about mobile agents.

1 Introduction

The interest for the development and formal study of mobile agents has grown in the last decade. With the creation of theories designed for this purpose, such as π-calculus [2, 21], Distributed Join-Calculus [20, 19] and Ambient Calculus [17], it has become possible to build software capable of reasoning about properties related to a given specification. These so called verification tools usually have a base theory and language, as well as some equivalence definitions and proof system.

Even for a single theory, verification tools are frequently developed in isolation and, thus, with different capabilities and notations. When users need to use various verification tools to solve a problem, they have to learn and use each verification tool individually. Many times, users prefer to partially solve their problems instead of learning various verification tools. On the other hand, new verification tools are created from scratch and, then, reimplement services already available in existing tools, instead of reusing them.

We believe that these problems can be partially solved. Since verification tools differ only in their conventions, but not in their application domains, it is reasonable to assume that it is possible to create a *common* 'language' on the top of the native 'languages' of *each* verification tool. Such common 'language', together with the relation among its elements, is called an ontology [3, 4]. From a general point of view, an ontology is a language that specifies an application domain in a way that different tools may exchange information about it. In our case, an ontology for mobile agents verification tools must describe not only the domain of these agents, but also the capabilities of the verification tools. In other words, it must describe the objects being studied (i.e., mobile agents) as well as what the verification tools can reason about (i.e., the class of properties and equivalences the verification tools may check).

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 1500–1516, 2005.

Establishing an ontology for mobile agents can help in integrating existing verification tools. It can be used to denote the language and capabilities of verification tools in a mobile agents environment:

- verification tools may be registered into the system using an ontology;
- users[1] may find an appropriate verification tool among the registered ones using ontology based search criteria.

In this paper, we present an ontology for the mobile agents domain, the first step toward the system described above. The ontology describing the verification tools capabilities remains as work in progress and, thus, it is not presented here.

Mobile agents and, therefore, mobility itself, may be defined in several ways. In our work, we use the π-calculus theory and tools as the mobility paradigm. However, as it will soon become clear, our ontologies are extensible in order to work with other mobility theories.

At last, we point out that there are other formal approaches to agent modeling, such as the SMART framework [5]. However, as far as we know, these models have not been used to foster verification tools interoperability.

In the remainder of this paper, we present the notations and tools we used (Sect. 1.1), the ontology itself (Sect. 2) and an usage example (Sect. 3). We also discuss our results and the work that remains to be done (Sect. 4).

1.1 Notation and Tools

We have chosen the Web Ontology Language (OWL) [12] as the notation for our ontologies. We had considered other options, such as UML, but found them to be very limited. OWL, on the other hand, provides a rich logic language based on Description Logic [13] as well as supporting tools, since it is becoming a mainstream industry[2] resource.

Among existing tools for OWL, we opted for Protégé-2000 [14]. Protégé not only supports OWL design, but also provides an API (Application Programming Interface) that allow external software to reuse many of its resources. This characteristic might be useful at a later stage of our work.

To make easier understanding the ontology, in this paper we present a plain English description of it, as well as pictures generated using the OntoViz [15] Protégé plugin. The formal OWL definitions, along with the Protégé project files, can be obtained in the following URL:

`http://www.ime.usp.br/~salem/papers/mobile_agent_ontology.zip`

2 The Mobile Agent Ontology

The purpose of our mobile agent ontology is to describe the possible components of a mobile agent and its environment, and to provide a way to describe actual formalisms structures, such as those from π-calculus.

[1] Both humans and other software systems.

[2] OWL is a new W3C recommendation and W3C is mainly industry oriented.

Notice that the ontology does not aim to describe the actual structure of agents, this would be equivalent to create a new theory for mobility analogous to the π-calculus. Our aim here is to describe the properties of agents structure (e.g., elements that can be used to build an agent, not the actual blueprint of the agent).

Tools using this ontology to exchange information about agents are benefited because:

- They only need to represent the search criteria once, using the ontology. Without the ontology, it could be necessary to specify several search criteria, each one designed to match the proprietary notation employed by each tool it is trying to communicate with;
- They may be developed using other theories rather than π-calculus, as long as the structures from these theories can be mapped into our ontology;
- They may provide a more user-friendly description of agents, since the ontology is designed at a high abstraction level.

The ontology is divided in two subontologies:

- the high level description of the mobile agent domain;
- the description of the formalisms employed by the tools;

A mapping of the first onto the second subontology is also provided (Sect. 2.3).

In the remainder of this section we shall present both subontologies and the mapping in details.

2.1 Subontology 1: Mobile Agents

This subontology describes the domain of mobile agents. It is based on the 'agent' concept found in [1], which, in short, defines an agent as an entity having sensors, actuators and that exists in an environment.

The scope of our ontology is restricted to the *formal methods* area and to an special kind of agent, the *mobile* one. Therefore, there are two important aspects which make our ontology unique:

- First, there are several characteristics from the agents domain which are not useful to us. For instance, it is not of our interest to define 'rationality', since it is not clear if such concept would be useful for formal verification. We have, thus, excluded concepts that are either irrelevant or obscure;
- On the other hand, the general theory about agents found in [1] lacks several elements that could be useful for an appropriate representation of mobility. For example, we crafted the concept of 'message' as a perception.

Now we will describe the classes of this ontology. To make the concepts clearer, we will first consider a cell phone as an example of mobile agent and show how this device is described by the ontology.

Example: Cell Phone. Cell phones are classical examples of mobile agents. To understand why, let us consider the fundamental idea behind them. If you are already familiar with that, you may wish to skip this section.

In order to enable cell phone operation in a certain region, the region is divided in several subregions, or cells. Each cell provides a connection point to the several cell phones inside its area. This access point is usually a station with several antennas to which the cell phones may connect using ordinary radio waves. Once connected, the station takes care to establish communication with the rest of the telecommunication network. See Fig. 1.

Mobility arises when a cell phone changes the cell it is in. When this happens, the phone must terminate the connection to the previous station and start one with the station covering the new cell. See Fig. 2

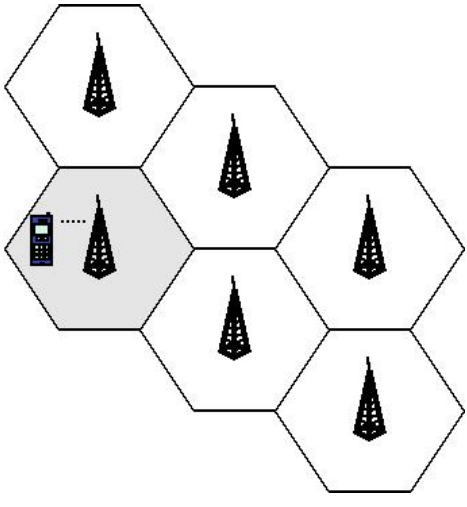

Fig. 1. Cell division of a region. The highlighted cell shows a cell phone connected to the cell's station.

Ontology Classes. As we pointed out earlier, the description of the ontology given in this paper is in plain English.

Fig. 3 depicts the ontology graphically.

Ontology class 1 (Agent). *Agents are entities which perceive and act in a particular environment. In our example, the cell phone itself belongs to this class. They have the following properties.*

hasSensors. *Zero or more instances from the Sensor class.*
hasActuators. *Zero or more instances from the Actuator class.*
isReconfigurable. *Boolean value. May the agent change its internal state during its existence? The cell phone, for instance, is reconfigurable, since at any given time it may be connected to a different antenna and be performing calls to different phones.*

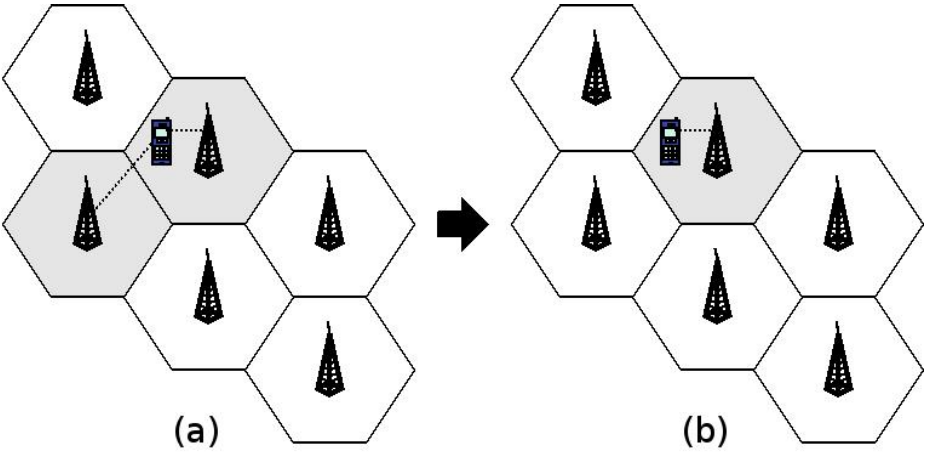

Fig. 2. Moving into another cell: (a) the phone moves physically into another region, preparing the connection with the new station without leaving the old connection; (b) the transition is complete and the phone is connected only to the new station

isDeterministic. *Boolean value. Does the current state of affairs determine the next action of the agent? The cell phone might be modeled either as a deterministic or nondeterministic agent. If we consider the phone as an agent that answers to its user's commands, then it is reasonable to consider it deterministic, for its behavior would not be random, it wouldn't, for instance, call a number unless it was requested to do so. On the other hand, if user interaction was hidden (i.e., implicit), then the phone could be seen as nondeterministic, since the user's behaviors (e.g., making calls) would provide a source of randomness.*
isCloneable. *Boolean value. Can the agent produce a copy of itself or of its internal parts? The cell phone is not cloneable, since it cannot duplicate itself.*
hasEnvironment. *One instance from the Environment class.*

Ontology class 2 (Percept). *A Percept is that which an Agent may sense through its sensors. The Percept class is abstract, and concrete subclasses must be defined. In the cell phone example, the percepts are the magnetic waves that reach the device through its antenna.*

Ontology class 3 (Message). *Message class is a concrete subclass of the Percept class. A Message is a piece of information that can be transmitted among agents. In the cell phone, for instance, if the user's voice is transmitted using discrete data packets, we could say that such packets belong to Message class. Notice, though, that if there are no discrete packets, if data flows continuously, then Message class could not be used to classify this transmission. Another Percept subclass would have to be created to account for that.*

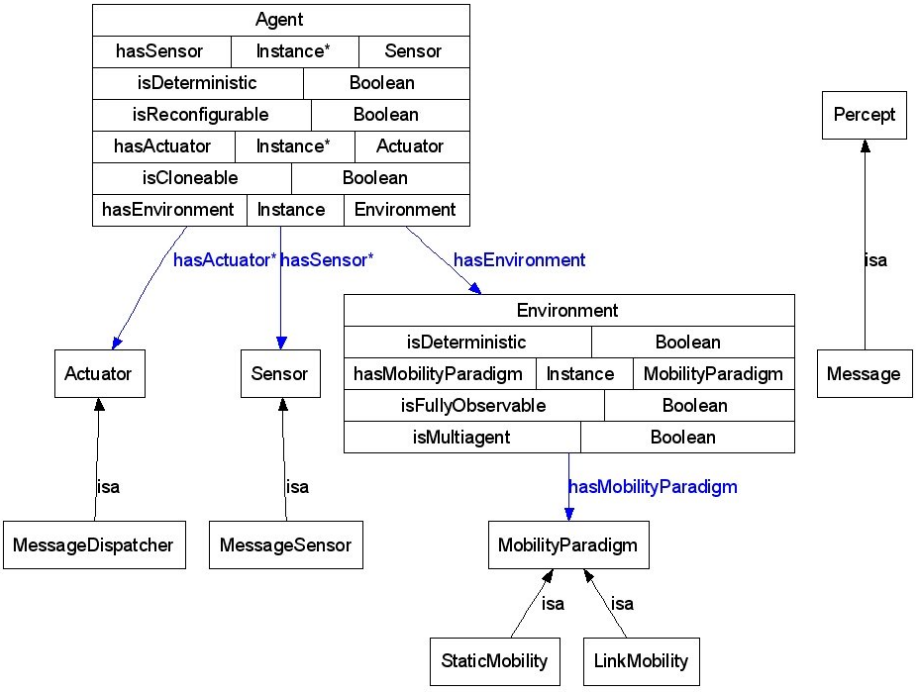

Fig. 3. Mobile agents domain ontology

Ontology class 4 (Sensor). *A Sensor is that which may receive Percepts. Sensor is an abstract class and concrete subclasses must be defined for each concrete subclass of Percept that are supposed to be perceived.*

Ontology class 5 (MessageSensor). *MessageSensor is a concrete subclass of the Sensor class. A MessageSensor is a Sensor that receives Messages. For instance, in the cell phone, the subsystem responsible for receiving and decoding the data packets could belong to MessageSensor class.*

Ontology class 6 (Actuator). *An Actuator is that which can perform actions over the Environment or other Agents. Actuator class is abstract and concrete subclasses must be defined for each action that the Agent may perform.*

Ontology class 7 (MessageDispatcher). *MessageDispatcher is a concrete subclass of the Actuator class. A MessageDispatcher is an Actuator that sends messages. In the cell phone example, the subsystem responsible for packing and sending data could belong to MessageDispatcher class.*

Ontology class 8 (Environment). *An Environment is where Agents exist. Cell phones, for example, exist in an environment of 'cells' (thus, the device's name). That is, physical space is divided into cells, and in each cell one or more*

antennas provide a connection point to the phones. From the cell phone stand point, the environment is nothing more than the antennas that it can connect to.

Environment class has the following properties.

isFullyObservable. *Boolean value. Can the Agent always sense the whole Environment? Cell phones cannot, since they are aware only of the antennas in the current cell.*

isDeterministic. *Boolean value. Given the Environment's current state and the Agents actions, is the next Environment's state determined? For cell phones, since hardware may fail, it is best to assume that the Environment is not deterministic.*

isMultiagent. *Boolean value. Can the Environment support more than one Agent? That's clearly true for cell phones.*

hasMobilityParadigm. *An instance from the MobilityParadigm class. How is mobility defined in this Environment?*

Ontology class 9 (MobilityParadigm). *A MobilithParadigm defines how mobility is handled by an Environment. MobilityParadigm class is abstract and concrete subclasses must be defined.*

Ontology class 10 (StaticMobility). *StaticMobility is a concrete subclass of MobilityParadigm. It defines the state of affairs in which no mobility at all takes place.*

Ontology class 11 (LinkMobility). *LinkMobility is a concrete subclass of MobilityParadigm. It defines the state of affairs in which the spatial position of an agent is given by its links to other agents, without any notion of distance. That's precisely the notion of mobility that a cell phone agent has, since links to antennas are all that such agents know about their position.*

2.2 Subontology 2: Formalisms

This subontology aims at describing the actual formalism structures that the verification tools employ. The description is given considering two different levels of abstractions. The first one defines abstract elements of any process calculus (e.g., operators, actions), while the second one specifies the elements of a particular agent calculus, the π-calculus. Graphical representations are provided.

Though we focus on the π-calculus, we believe that the ontologies can be extended to other mobility formalisms. In particular, because many of these formalisms, such as Join-Calculus [20, 19] and Ambient Calculus [17], can be proved equivalent to the π-calculus.

Ontology Classes. The main classes of this subontology are the following (see Fig. 4).

Ontology class 12 (Formalism). *Formalism is an abstract class. Actual formalisms must be represented by concrete subclasses.*

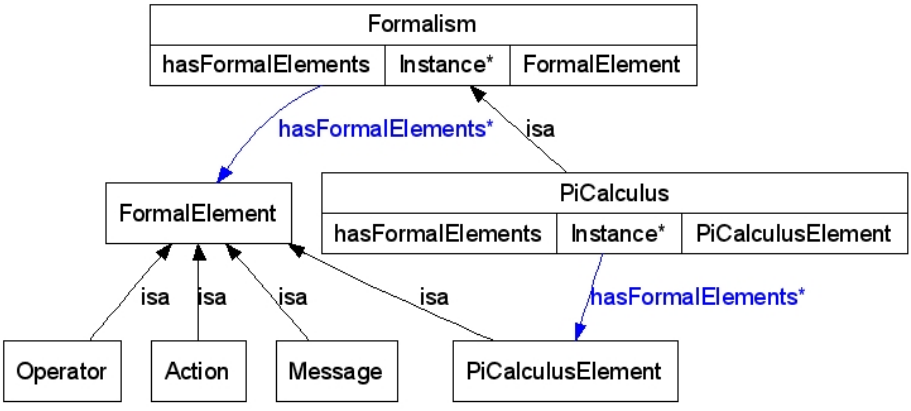

Fig. 4. Main formalism ontology classes

hasFormalElements. *Zero or more instances from the FormalElement class.*

Ontology class 13 (PiCalculus). *PiCalculus is a concrete subclass of Formalism class. It represents the π-calculus theory itself.*

hasFormalElements. *Zero or more instances from the PiCalculusElement class.*

Ontology class 14 (FormalElement). *FormalElement is an abstract class. It represents the syntactic building blocks of formalisms. This class must have concrete subclasses for each Formalism subclass.*

Ontology class 15 (Action). *Action is an abstract class. It represents the elements over which operations can be performed. That is, they are the atomic units that the process calculus works with.*

Ontology class 16 (Message). *Message is an abstract class. It defines what is sent through OutputActions and received by InputActions.*

Ontology class 17 (Operator). *Operator is an abstract class. Operators perform transformations over Actions.*

A class to group together all elements from π-calculus is also provided.

Ontology class 18 (PiCalculusElement). *PiCalculusElement is an abstract subclass of FormalElement. It represents the syntactic elements found in the π-calculus.*

Action, Message and Operator classes are the main abstract concepts that we use to describe a general process calculus. Each of these classes, in turn, have their own subclasses and, finally, these subclasses have actual concrete classes that represent an actual process calculus, the π-calculus.

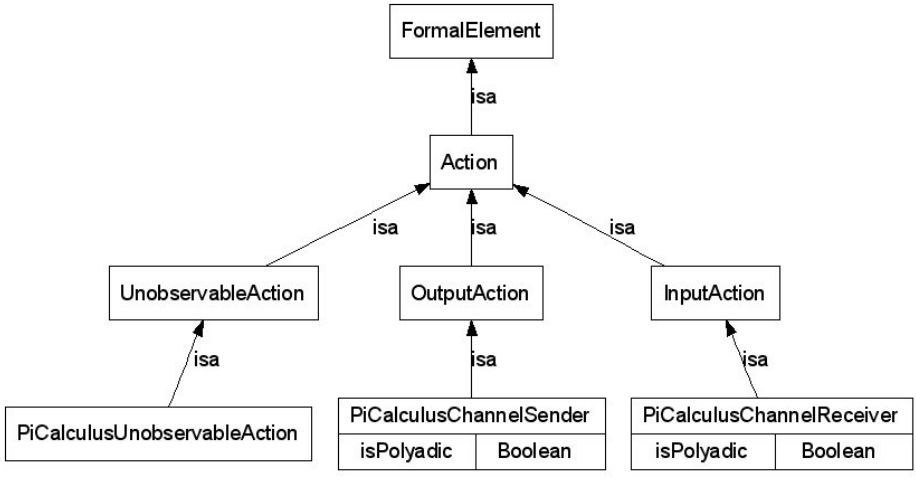

Fig. 5. Action subclasses

Ontology class 19 (InputAction). *InputAction is an abstract subclass of Action. It represents Actions that take some input.*

Ontology class 20 (OutputAction). *OutputAction is an abstract subclass of Action. It represents Actions that send some output.*

Ontology class 21 (UnobservableAction). *UnobservableAction is an abstract class. It represents Actions that do not affect other Actions.*

Ontology class 22 (ChoiceOperator). *ChoiceOperator is an abstract class. ChoiceOperators are Operators that given two expressions, output one of them in a nondeterministic manner.*

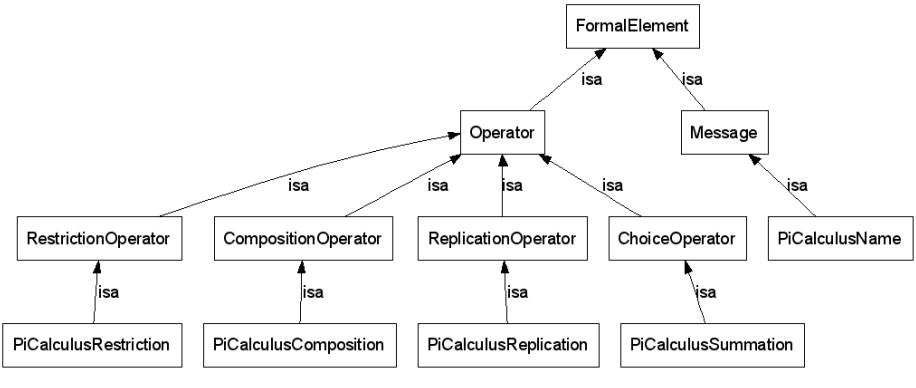

Fig. 6. Operator and Message subclasses

Ontology class 23 (CompositionOperator). *CompositionOperator is an abstract class. CompositionOperators are Operators that allow two expressions interact.*

Ontology class 24 (ReplicationOperator). *ReplicationOperator is an abstract class. ReplicationOperators are Operators that create copies of an expression.*

Ontology class 25 (RestrictionOperator). *RestrictionOperator is an abstract Class. RestrictionOperators are Operators that bind an identifier to a particular expression.*

The other classes are subclasses of PiCalculusElement and of some FormalElement. Each represents a particular construction from the π-calculus theory. Their names are self-explaining, but we shall list them here for the sake of completeness.

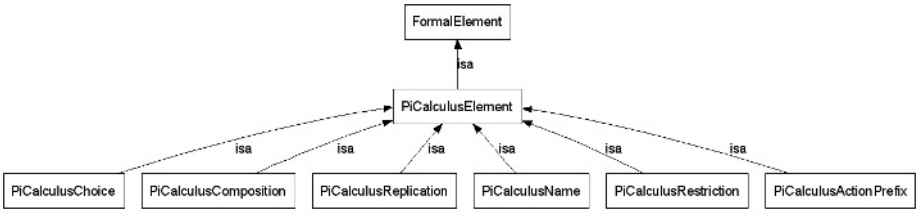

Fig. 7. π-calculus elements

Ontology class 26 (PiCalculusName). *Concrete subclass of Message. Defines π-calculus names.*

Ontology class 27 (PiCalculusActionPrefix). *PiCalculusActionPrefix is an abstract subclass of Action.*

Ontology class 28 (PiCalculusChannelReceiver). *PiCalculusChannelReceiver is a concrete subclass of PiCalculusActionPrefix and InputAction classes. It represents the π-calculus input channels. It has only one property.*

isPolyadic. *Boolean value. Can the channels receive messages consisting of more than one name?*

Ontology class 29 (PiCalculusChannelSender). *PiCalculusChannelSender is the concrete subclass of PiCalculusActionPrefix and OutputAction classes. It represents the π-calculus output channels. It has only one property.*

isPolyadic. *Boolean value. Can the channels send messages consisting of more than one name?*

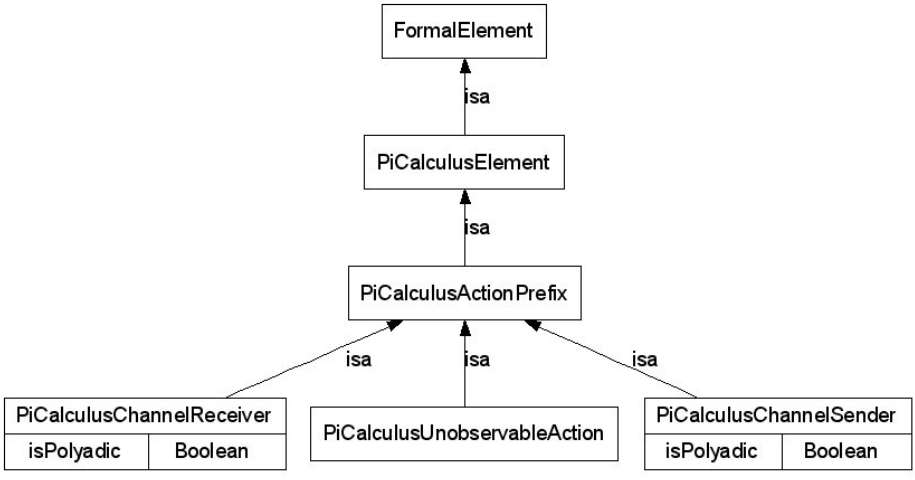

Fig. 8. π-calculus action prefixes

Ontology class 30 (PiCalculusUnobservableAction). *Concrete subclass of UnobservableAction. The τ action prefix.*

Ontology class 31 (PiCalculusComposition). *Concrete subclass of CompostionOperator. The π-calculus composition ($|$) operator.*

Ontology class 32 (PiCalculusChoice). *Concrete subclass of ChoiceOperator. The π-calculus choice ($+$) operator.*

Ontology class 33 (PiCalculusRestriction). *Concrete subclass of RestrictionOperator. The π-calculus restriction (new) operator.*

Ontology class 34 (PiCalculusReplication). *Concrete subclass of ReplicationOperator. The π-calculus replication ($!$) operator.*

2.3 Mappings of Agents onto Formalisms

At last, it is necessary to connect the first subontology to the second. We provide this connection using some rules which assert that the support for some structures in one subontology exists if, and only if, the support for some other structures from the second subontology also exists.

These rules, together with the profile[3] for a verification tool, make it possible to find out if the verification tool matches a search criteria. The search criteria is given using the mobile agent ontology, but the verification tool had been originally programmed and documented having the particular formalism (e.g., the π-calculus) in mind.

[3] The profile for a verification tool is the set of elements from both ontologies that have been registered to be supported.

Mapping Rules. We shall now state the rules, in a formalism-centric fashion.

Mapping rule 1 (PiCalculus). *PiCalculus theory is supported if, and only if, the Environment's MobilityParadigm is a LinkMobility.*

Mapping rule 2 (PiCalculusName).
PiCalculusName class is supported if, and only if, Message, MessageSensor or MessageDispatcher classes are supported.

Mapping rule 3 (PiCalculusChannelReceiver). *PiCalculusChannelReceiver is supported if, and only if, MessageSensor class is supported.*

Mapping rule 4 (PiCalculusChannelSender). *PiCalculusChannelSender is supported if, and only if, MessageDispatcher class is supported.*

Mapping rule 5 (PiCalculusUnobservableAction). *PiCalculusUnobservableAction is supported if, and only if, MessageDispatcher, MessageSensor and PiCalculusComposition classes are supported.*

Mapping rule 6 (PiCalculusChoice). *PiCalculusChoice is supported if, and only if, the supported Agents have the property Deterministic set to false.*

Mapping rule 7 (PiCalculusRestriction). *PiCalculusRestriction is supported if, and only if, the supported Environment is not fully observable.*

Mapping rule 8 (PiCalculusComposition). *PiCalculusComposition is supported if, and only if, the supported Environment has its properties set according to the following rules.*

isDeterministic. *Must be set to true.*
isMultiagent. *Must be set to true.*

Mapping rule 9 (PiCalculusReplication). *PiCalculusReplication is supported if, and only if, the supported Agent has its Cloneable property set to true.*

3 Usage Example

Let's now consider an example of how the ontology may be used. For simplicity, let's assume we have only two verification tools for the π-calculus, M_1 and M_2, which differ in notations employed as well as in the supported fragments of π-calculus.

We would like to integrate M_1 and M_2 in a system so that we could:

- query the system to find out if either M_1 or M_2 give support to a certain subset of π-calculus that we are interested in;
- write such a query using a language which does not depend on the verification tools' language syntax. For instance, M_1 might denote the composition operator as '—' and M_2 as 'comp', but we wish the query to be independent of these syntatic particularities.

To achieve these goals, the system could use our ontology. In order to integrate verification tools, the following steps could be carried out for each tool:

1. find out which π-calculus elements the verification tool supports. This could be done, for example, surveying the verification tool's documentation or source code. In the example, let's say that M_1 supports all the basic[4] π-calculus and that M_2 supports the same elements, except for the choice operator.
2. register the supported elements using the formalisms subontology. In the example, all we need to do is to tell the system that M_1 supports the PiCalculus formal theory with PiCalculusName, PiCalculusChannelReceiver, PiCalculusChannelSender, PiCalculusUnobservableAction, PiCalculusChoice, PiCalculusRestriction, PiCalculusComposition and PiCalculusReplication and tell that M_2 supports all of those too, except for PiCalculusChoice.

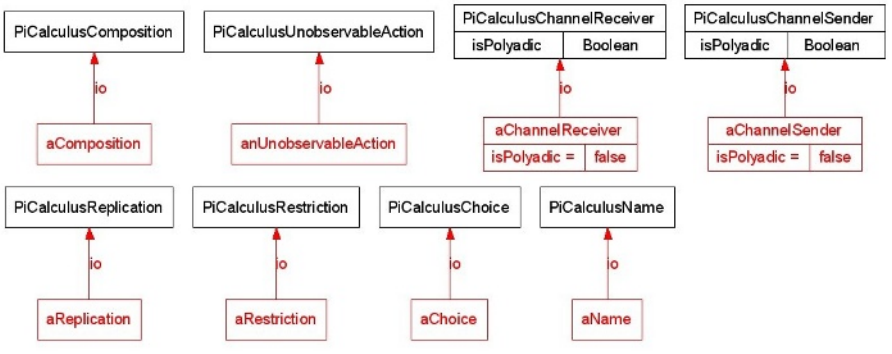

Fig. 9. Instances of all π-calculus elements we need

3. use the mapping rules to discover which elements from the mobile agent ontology are supported by the verification tool. In the example, we would find out that the only difference between M_1 and M_2 is that M_1 will support choice of agents, while M_2 won't.
4. make these elements available to a search engine. Notice that now the verification tool's particularities do not trouble the search, since they have been mapped into the ontology.

Now, suppose a user wants a verification tool that supports some elements from the π-calculus, in particular the choice operator. To locate the desired verification tools, the following steps could be performed:

1. using the ontology, the user tells the system which elements from the π-calculus theory are required.

[4] The subset of π-calculus which contains action prefixes, choice, composition, replication and restriction

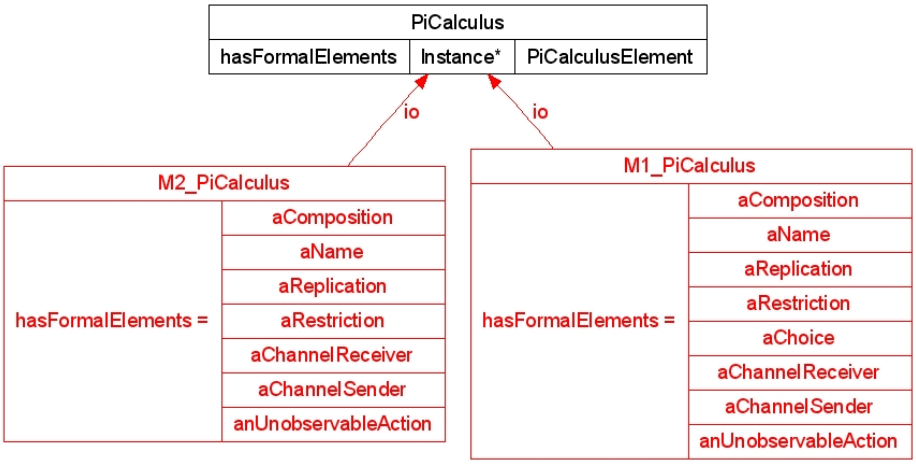

Fig. 10. Instances of the π-calculus theory that each verification tool support

2. using the mapping rules, the system transforms these elements into mobile agent's elements.
3. the system checks in its data base if there are any registered verification tools that gives support to the mobile agent elements. We have seen that only M_1 supports PiCalculusChoice. Thus, only M_1 will be returned to the user as a search result.

The point here is that the search is done using a common language, which does not depend on the underlying formal theory. Although so far we have been working only with the π-calculus, we believe that other formalisms, such as Ambient Calculus [7] or Join-Calculus [6], could be used as well.

The example π-calculus tools used in this section are fictitious and were proposed to illustrate the problem of having tools for a single formalism that are concerned with different fragments of it. In practice, we also have verification tools for π-calculus which actually support different subsets of the theory. The verification tool VTubaina [8], for example, does not support the choice operator, while another tool, HAL [9, 10, 11], does support choice but not replication. In fact, these tools can share services although they contemplate different fragments of π-calculus and their input languages differ.

4 Discussion

This paper presented two main ontologies: one for mobile agents and one on formalisms of mobile agents. Besides that, the relation between elements of both ontologies has been provided in order to show how agents can be modeled from abstract concepts to appropriate mobile agent formalisms. π-calculus has been

used as instance of the formalism ontology, and an example involving two proposed tools for π-calculus was presented to give insights on the usage of ontologies as a framework to register tools and use them in a collaborative way.

The ontologies defined here have a double purpose. From a computational stand point, it provides a common language for information exchange. This allows tools using different notation to integrate, albeit in a limited fashion. On the other hand, from the user's point of view, the ontology defines an *understandable* language, from which higher level concepts of the mobile agent domain may be easily handled.

We have given the cell phone as an example of mobile agent that can be modeled using our ontologies. Other examples can be easily found among modern wireless devices (e.g., wireless networks, where computers may connect as they move into a certain region), which share many of the cell phone's characteristics. Furthermore, mobile agency is also present in software systems. An example of software mobility is the Aglets project [16].

Regarding those features, verification tools can be registered as instances of the formalism ontology and we can further provide an environment for sharing services from various tools. For that, a software system must actually be implemented and make use of our ontologies to register verification tools in order to build a cooperative environment. We believe that Protégé-2000 can be used as a framework for such implementation, since it is designed as an *extensible*[5] tool for ontologies.

In this particular work, we have instanced the formalism ontology for π-calculus. This formalism ontology, however, is not limited to represent π-calculus. Instances of this ontology for other formalisms, such as Join-Calculus and Ambient Calculus, can be produced in the same way. With such new instances of formalisms, we could register tools based on these other calculus and, better than this, provide an environment that comprises services from different formalisms. The ontological relation among all instanced formalisms can be established via the main formalism ontology and, for certain classes of problems, tools services from different formalisms can be shared.

The ontology developed in the present work refers only to the representation of agents. To provide an environment for sharing services from verification tools for mobile agent systems, we also need, as stated in the introduction, an ontology to capture the *capabilities* of the verification tools. This study is in progress and is first devoted to capabilities of model-checkers and equivalence verification tools.

We don't have the expectation of solving the whole problem of services-collaboration among verification tools for mobile agent systems with such ontologies. We already known that ontologies are not enough for knowledge sharing [18] because a semantic mapping is necessary for certain pairs of formalisms, and this problem remains in services-sharing among verification tools for mobile agents. However, many of the formalisms for mobile agents are comparable and a great part of them can be mapped without a semantic conversion. The present work is a step-forward to provide services-sharing among tools for such classes of

[5] Through the use of plug-ins.

problems. For a complete services-sharing, a study on pairwise verification tools must be conducted and there is no guarantee that a semantic mapping can be found for each pair of formalisms.

Finally, we note that the ontologies described in this paper have not yet been implemented in a working software system.

Acknowledgments

This project has been granted by CNPq (*Conselho Nacional de Desenvolvimento Científico e Tecnológico - Brazil*) and FAPESP (*Fundação de Amparo à Pesquisa do Estado de São Paulo -Brazil*), Project 03/00312-0.

References

1. Stuart Russel and Peter Norvig. *Artificial Intelligence:* A Modern Approach. Prentice Hall, 2003.
2. Robin Milner. *Communicating and Mobile Systems:* the π-Calculus. Cambridge University Press, 1999.
3. Nicola Guarino and Pierdaniele Giaretta. *Ontologies and Knowledge Bases:* Towards a Terminological Clarification. In *Towards Very Large Knowledge Bases:* Knowledge Building and Knowledge Sharing, N. Mars (ed.), pp 25-32. IOS Press, Amsterdam, 1995.
4. Thomas R. Gruber. A translation approach to portable ontologies. In *Knowledge Acquisition*, vol. 5, pp. 199-220, 1993.
5. Mark D'Inverno and Michael Luck. Understanding Agent Systems. 2nd ed. Springer, 2004.
6. *The Join-Calculus language*. Resources for the Join-calculus. Available at: <http://join.inria.fr/>. Accessed in: May 23rd, 2005.
7. Luca Cardelli. *Mobile Computational Ambients*. Introduction and other resources to the Ambient Calculus. Available at: <http://www.luca.demon.co.uk/Ambit/Ambit.html>. Accessed in: May 23rd, 2005.
8. Marcelo M. Amorim. VTUBAINA Tool. A Verification Tool for Up-to Bisimulation and Automata Integration Automatization. Available at: <http://www.lcpd.ime.usp.br/~mamorim/vtubaina/>. Accessed in: May, 23rd, 2005.
9. Ferrari, G., Gnesi, S., Montanari, U., Pistore, M. and Ristori, G., Verifying Mobile Processes in the HAL Environment, in: Alan J. Hu and Moshe Y. Vardi, Eds., *CAV'98*, *Springer LNCS* 1427, pp.511-515.
10. Montanari, U. and Pistore, M., An Introduction to History Dependent Automata, in: Andrew Gordon, Andrew Pitts and Carolyn Talcott, Eds, *Second Workshop on Higher-Order Operational Techniques in Semantics (HOOTS II)*, ENTCS, Vol. 10, 1998
11. History Dependant Automata Laboratory (HAL). Available at: <http://rep1.iei.pi.cnr.it/projects/JACK/HAL/hal.html>. Accessed in: May, 23rd, 2005.

12. *Web Ontology Language (OWL)*. Specifications, articles, tools and other resources for OWL. Available at: <http://www.w3.org/2004/OWL/>. Accessed in: August 19th, 2004.
13. *Description logics*. Description logics resources. Available at: <http://dl.kr.org/>. Accessed in: February 21st, 2005.
14. The Protégé Ontology Editor and Knowledge Acquisition System. Available at: <http://protege.stanford.edu/>. Accessed in: August 19th, 2004.
15. OntoViz. Information and downloads regarding the OntoViz Protégé plugin. Available at: <http://protege.stanford.edu/plugins/ontoviz/ontoviz.html>. Accessed in: February 21st, 2005.
16. Aglets. Information and downloads regarding the Aglets framework, a system for mobile software agents. Available at: <http://aglets.sourceforge.net>. Accessed in: August 15th, 2005.
17. L. Cardelli and A. D. Gordon. Mobile ambients. In Maurice Nivat, editor, *Foundations of Software Science and Computational Structures*, volume 1378 of *LNCS*, Springer-Verlag, 1998.
18. Flávio S. C. da Silva, Ana C. V. de Melo, Jaume Agusti, Wamberto Vasconcelos, David Robertson, Marcelo Finger, and Vigínia Brilhante. On the insufficiency of ontologies: Problems on knowledge sharing and alternative solutions. *Knowledge-Based Systems*, 15(3):147–167, 2002.
19. C. Fournet and G. Gonthier. The reflexive chemical abstract machine and the join-calculus. In *23rd ACM Symposium on Principles of Programming Languages*. ACM Press, 1996.
20. C. Fournet, G. Gonthier, JJ Levy, L. Maranget, and D. Remy. A calculus of mobile agents. In *CONCUR'96*, volume 1119 of *LNCS*. Springer–Verlag, 1996.
21. D. Sangiorgi and D. Walker. *The Pi-Calculus: A Theory of Mobile Processes*. Cambridge University Press, October 2003.