ABSTRACT
This paper describes an object-oriented framework for ontology representation, which was derived from the analysis of a set of use cases for ontology services provided by ontology agents in open agent networks. The described use cases are converted into a set of requirements for the Information System supporting the ontology service.

Each ontology on the server has classes, properties and methods. Classes have attributes, which are properties, methods and axioms. Attributes have facets. Methods have arguments, return value and method definition through axioms. Classes and properties may be arranged in hierarchies. The proposed framework accommodates the expression of arbitrary axioms about the entities in the ontology. We propose a situation calculus approach for describing the action methods of the ontology. Action methods are described by state change and state constraint axioms. The frame problem is handled assuming that nothing changes unless explicitly stated.

We present the definition of several relations between entities of different ontologies that allow inferring global relations between those ontologies. Finally, we propose Extended-SL as the Representation Language of O₃F.

Categories and Subject Descriptors
I.2.4 [Artificial Intelligence]: Knowledge Representation Formalisms and Methods – representation languages.
H.3.5 [Information Storage And Retrieval]: Online Information Services - data sharing, web-based services

General Terms
Standardization, Languages, Theory

Keywords
Ontology, ontology representation framework, action methods, arbitrary axioms, situation calculus, FIPA SL.

1. INTRODUCTION
Ontologies are very useful in a wide range of situations [14][10]. In the less complex extreme of the range, ontologies may be used by agent designers to build their agents for specific domains. Using a widely accepted and disseminated ontology, agent designers can implement agents that may understand a large set of message contents in their application domain. In the more complex extreme of the range, ontologies would be dynamically and incrementally developed, acquired and used by the agents – one of the main uses would be text mining. In this paper we focus our attention in an intermediate complexity problem. We focus on the problem of creating domain independent agents that use the ontologies as a means of finding other agents that provide desired services and information; and as a means for tailoring domain independent information processing mechanisms to specific domains.

In Agentcities [16], most likely the largest agent network ever implemented, there are several value-added types of agents, which must dynamically discover and integrate information and services for their clients [2]. For instance, a Personal Assistant Agent (PAA) may want to find an agent that can book a table in a restaurant. In this scenario the PAA must look for ontologies describing services whose execution results in a table being booked. Possibly, there are different such services. A restaurant representative agent may have the service BookTable that results in a table being booked. The Evening Organizer [4] is another agent providing a more complex service that might also result in a table being booked. Therefore, the procedural description of the service would not help the PAA find the service provider. The best way to find it is by searching for ontologies describing the world states that result of the execution of services. Since services, in the object-oriented framework, are provided by action methods, it is necessary that ontologies provide declarative method description.

This paper makes some contributions to the state of the art in ontologies. First, it presents an explicitly assumed object oriented framework for ontology representation, while similar approaches use a more frame-oriented dressing. We show that our framework supports the representation of DAML+OIL [6] and Ontolingua [7] ontologies. One of the most relevant contributions is the proposal to include method definitions in the ontology, especially action methods (methods whose execution changes the state of the world). Other frameworks, especially Ontolingua, allow the definition of functions and relations but not actions. The other major contribution is the formal definition of relations among
entities in different ontologies from which it is possible to infer global relations between ontologies. Another contribution is the use of FIPA SL [9] as the surface representation language of the framework.

The remaining sections of the paper are organized as follows. Section 2 presents a set of use cases for an ontology service provided by an ontology agent in an open agent network. In Section 3, the presented use cases are converted into a set of requirements for the ontology information system. The information system is described through a UML class diagram. FIPA SL is proposed as a surface language to represent the ontology. Section 4 argues that the information system described in section 3 is not enough from the semantic point of view. It adds provision for hierarchic information representation; it includes arbitrary axioms in the proposal, especially axioms for the declarative description of action methods. Section 5 shows that the presented framework supports inference about relations between ontologies. Section 6 discusses the potential of the presented approach, its advantages and disadvantages, comparing it with related work. Finally, section 7 presents conclusions.

2. USE-CASES

Practical cases in the Agentcities Project revealed several shortages of commonly used ontology representation frameworks and the need to create an improved framework. This section describes a set of use-cases originated in the Project.

How will an agent ask for ontologies and agents that provide/use those ontologies? This section shows some use-cases of a broker agent as an example.

Consider an Agent that works as a broker of all kinds of Restaurant Services. We will call this agent Fredo. In order to provide/use information about restaurants Fredo will look for agents that work with any Restaurant Ontologies. First, Fredo will look for the domain ontologies by asking the Ontology Agent (OA) the following question: “What are the ontologies that define/use class ‘Restaurant’?”. In a generic and more complete way, the question is: “What are the ontologies that define class C with a set of attributes including attributes \{A_1, A_2, ..., A_n\}?”. With this information, Fredo will be able to look for agents providing services based on the ontologies returned by the OA.

Consider further that Fredo wanted to know what kind of information is delivered in those ontologies (related to information services) or the key attributes of the class. It would have to ask OA something like: “What are the attributes of the class Restaurant?” or “What are the sets of attributes that univocally identify the instances of class Restaurant defined in ontology Restaurant-Information-Service-Ontology?”. The same applies if Fredo wishes to know something about a class’s methods. For instance, Fredo is looking for ontologies that provide a Restaurant Booking Service. It would have to ask the OA: “What are the ontologies that define class Restaurant with method BookTable, which takes an String (Table Name) and a String (Booker Name)?”.

More generally, what are the ontologies that define class C with a set of methods including the methods \{<M_i, \langle P_{i,1}, C_{i,1} \rangle, P_{i,2}, C_{i,2}>, \ldots, \langle P_{i,m}, C_{i,m} \rangle, C_i\}\, and a set of attributes including attributes \{\langle A_j, C_j \rangle\}, in which \(P_{i,1}\) is the name of argument 1 of method \(M_i\), and \(C_{i,1}\) is the class of the argument 1 of method \(M_i\), \(A_j\) is the name of attribute \(j\) and \(C_j\) is the class of attribute \(j\), and \(C_i\) is the class of the value returned by the method \(M_i\)?

3. MINIMUM REQUIREMENTS

In order to be able of handling the interactions described in section 2, the OA must have the following information: the set of properties of the ontology, the set of methods of the ontology and the set of classes of the ontology. Each property is described by property name, property type (i.e., class or a primitive data type) and unique identifier. Each method is described by method name, a set of parameters, i.e., parameter name and parameter type, by the type of the method return value if there is one or void if none exists, and by a unique identifier. Each class is described by the class name, a set of attributes and a set of methods. Each class must also specify the set of key attributes that univocally identify its instances.

Figure 1 shows the UML class diagram of the information system necessary to represent ontologies, following the presented analysis. The presented class diagram states that properties can be of two kinds: either their range is a datatype like String, Integer or Float; or a Class. An attribute is a property that is associated to a class. It is also possible to declare alternative key attribute sets of a class. Attributes have facets that enumerate a set of characteristics like cardinality, numeric maximum and others. The Information System can capture relations among classes (e.g., inheritance and disjoint composition) and among properties (e.g., inheritance and inversion). Arbitrary axioms, functional and relational definitions, and state change axioms cannot easily be captured in a traditional information system – some knowledge representation language is also needed.

Ontolingua is the best-known ontology representation framework allowing the representation of arbitrary axioms. Here we present an alternative approach using FIPA SL language.

The information system represented by the class diagram can be manipulated through the general-purpose predicate and function instance/2 and value/2 respectively. Ontologies may be added to the Information System using the predicate instance/2 [1].

The following example states that the ontology RestaurantOntology created by the Agentcities Consortium is an instance of the class Ontology.

(instance (Ontology :name RestaurantOntology :author Agentcities) Ontology)

Particular classes may also be inserted in the class ‘Class’ using the instance/2. For example, (instance (Class :name Restaurant) Class) means that the class named ‘Restaurant’ is an instance of class ‘Class’.

Using the predicate instance/2, it is also possible to create properties. (instance (Property :name maxPrice) Property) means that ‘maxPrice’ is an instance of class ‘Property’.
The associations between the classes are represented using the special-purpose class *Association* which specifies the relation between two or more classes. For example, \(\text{instance Association :name ClassOfOntology :arguments (set (AArg :name Class :key name :object Restaurant) (AArg :class Ontology :key name :object RestaurantOntology) :attributes (set ) Association)\) means that Restaurant (of class Class) is associated with RestaurantOntology (of class Ontology).

The option of representing associations by the special-purpose class *Association*, which specifies the classes of the associated entities, allows to reason about the way classes are associated with each other. Using the proposed approach, agents can dynamically add new entities and relationships between them to the ontology, and they can also query the ontology service about existing entities and the relationships between them.

Through the message in Figure 2, Fredo asks the Ontology Agent to tell it the classes of ontology RestaurantOntology.

The expression \((\text{value ?c name})\) represents the value of attribute ‘name’ of the object ?c, which in this case is an instance of class Class. The name of a class is its unique identifier. The detailed reading of the message in Figure 2 is “Send me the instances of class Class that are associated to the Ontology RestaurantOntology, through the association ClassOfOntology”.

The presented framework, O3F, allows the representation of ontologies originally described in other frameworks, such as Ontolingua and DAML-OIL. Furthermore, the use of Extended-SL as the ontology representation language facilitates ontology sharing between agents.
4. ARBITRARY AXIOMS

The information system that supports the minimum requirements identified for the Ontology Service provides only a very shallow semantics (see section 3).

Some useful relations between the entities of the ontology cannot be represented by the above system. Some provision must be made in order to represent arbitrary axioms in the ontology. For instance, we might be interested in expressing some constraint between two attributes of a given class through a set of axioms, such as A1 must take a value between the value of A2 and its double.

Finally, in many circumstances, it is necessary to clearly define the methods of the ontology. Sometimes, the name, the return type and the arguments of the method are not enough – agents must reason about what the method is. Functional and relational methods may be defined declaratively through a set of axioms. Methods that change the state of the world can also be declaratively defined using situation calculus, through the specification of the relation between the input state and the output state.

In the next subsection we show that FIPA SL can be used to define relational and functional methods through sets of axioms. Sub section 4.2 presents one of the main contributions of the paper: a proposal to represent action methods through axioms about the way the method changes the state of the world when it is executed.

4.1 Relational and Functional Methods

Relational and functional methods may be defined by sets of statements in the language of first order logic. For instance, if it is necessary to define a method PriceDiff of the class Restaurant that computes the difference between the maximum and minimum prices, we may use the following axiom

∀r Instance(r, Restaurant) ∧ IsNumber(r.maxPrice) ∧ IsNumber(r.minPrice) ⇒ r.PriceDiff() = r.maxPrice – r.minPrice

This axiom can easily be represented in Extended SL [1] as follows:

(forall ?r (implies (and (instance ?r Restaurant) (isNumber (value ?r maxPrice)) (isNumber (value ?r minPrice))) (= (value ?r PriceDiff (sequence)) (- (value ?r maxPrice) (value ?r minPrice)))))

Relational methods and arbitrary constraints may also be defined in SL. This way, agents can dynamically add constraints, relational and functional definitions to the ontologies. If those definitions are kept in the ontology service information system, they may also be consulted to be further reasoned upon.

4.2 Representing Action Methods

Action Methods are methods whose execution changes relevant aspects of the state of the world. Therefore, if we want to declaratively represent action methods in the ontology, we need some provision to talk about world states and changes in them. In this paper we propose to describe action methods in ontologies through situation calculus, which was introduced by McCarthy and Hayes in 1969 [13] for describing how actions and other events affect the world.

In this proposal, state-dependent properties are described by methods that map objects into sets of world states in which the objects have a specific value for an attribute. For instance, Table1.BookedBy(A) represents the set of world states in which the Table1 (of a certain restaurant) has the value A for the attribute BookedBy. Such methods are called fluents.

Since a fluent returns the set of world states in which a given condition holds, in order to say that a given condition, represented by the fluent F() of object obj, holds in a given state of the world (s), we write s ∈ obj.F(). For instance, S0 ∈ Table1.BookedBy(A) means that, in state S0, Table1 is booked by A.

In order to emphasize the fact that fluents represent conditions of the world, it is common practice to use logical notation to represent set operators that will be applied to fluents. ¬ denotes complement, ∧ and ∨ denote intersection and union, and (A ⇒ B) means that A is a subset of B.

In this approach, concrete invocations of action methods are represented by action designators. For instance, the action method BookTable may be invoked on an object of type Table with the person on behalf of whom the table is booked as argument: Table1.BookTable(A).

Finally, the function Do maps pairs of method invocation and world state into world states. It represents the state that results of the invocation of a given method with a given sequence of arguments in a given world state.

Methods are described by a set of state change axioms that represent the way their invocation changes the world, as in the following example:

∀t,x,s s ∈ t.Free() ⇒ Do(t.BookTable( x), s) ∈ t.BookedBy(x)

If a table is free in a certain state, then it will become booked by x in the state resulting of x booking that table.

To properly define what happens in the world when an action method is executed, we also need to state several axioms that represent general constraints that hold in any world state. These are called state constraints.

∀t,x,s s ∈ t.BookedBy(x) ⇒ ¬∃y (s ∈ t.BookedBy(y) ∧ y ≠ x)

If a table is booked by someone, it can’t be booked by someone else in the same world state.

Using the state change axioms that describe the way the world changes by invoking methods, we may know what becomes true when the method is invoked. Using state constraints we may conclude what becomes false when methods are invoked. In the case of table booking, we know that after a table has been booked, it becomes booked by someone, and we also know that it is not booked by anybody else.

State change axioms and state constraint axioms describe what becomes true and what becomes false when methods are invoked, but they don’t describe what is preserved after methods are invoked. This is the so-called frame problem.

The frame problem may be solved by writing sets of frame axioms that explicitly state what does not change when methods are invoked. However this is not practical in realistic situations because for each method, there are so many things that stay the
same that we would need a tremendous quantity of frame axioms. The alternative that we chose is a more practical way of handling the frame problem. We assume that, if nothing is stated otherwise, things do not change.

The above axioms can also be written using FIPA SL syntax.

\[
\text{(for all } ?t \text{ (for all } ?x \text{ (for all } ?s \text{ (imply (member } ?s \text{ (value } ?t \text{ Free (sequence) } ?x) \text{ (value } ?t \text{ BookedBy } ?s)))})
\]

\[
\text{(for all } ?t \text{ (for all } ?x \text{ (for all } ?s \text{ (value } ?t \text{ BookedBy } ?x) \text{ (not (exists } ?y \text{ (and (member } ?s \text{ (value } ?t \text{ bookedBy } ?y) \text{ (<- } ?x ?y)))))})
\]

Using the proposed approach, action methods can be declaratively described in terms of what changes in the world. Contrarily to what happens with DAML-S Service Descriptions [5], the presented approach allows agents to access to and to reason about methods described in the ontology in terms of the changes they can bring about and not in terms of their internal control structure. This is a clear advantage since there are several ways a method can be internally engineered in order to effect the same changes. This approach is also amenable of being expressed in FIPA SL, which is an enormous advantage since SL is being used both in FIPA Platform Management Specifications and in the Agentcities project, and has extensive computational support available.

5. RELATIONS BETWEEN ONTOLOGIES

The FIPA Ontology Service Specification [8] presents a framework, explicitly based on OKBC [3], which specifies the behavior of a FIPA compliant Ontology Agent. This specification includes two different ontologies: the Meta Ontology, which deals with the objects specific to an ontology (classes, slots, etc), and the Ontology Service Ontology (OSO), which deals with the operations on ontologies and the relations between them.

The OSO defines six possible kinds of relations between ontologies: extension, identical, equivalent, strongly-translatable, weakly-translatable and approx-translatable. This classification presents a significant shortage: it applies to the whole ontology, ignoring the relation between individual classes. This implies that if we have two ontologies that share the same n classes but that have each one additional class that is not shared, they will simply be considered weakly-translatable.

To overcome this lack of expressiveness, we present an approach that defines relationships between the entities of two ontologies (e.g., classes and attributes). Specific relations between the lower level entities determine the level of the relation between ontologies. This approach allows inferring not only global relations between ontologies but also relations about the entities in the ontology.

Ontologies define complex entities from structural and functional relationships between simpler entities. In the end, every complex concept is represented in terms of a pre-defined and limited initial vocabulary. In the object-oriented ontology representation framework described in sections 3 and 4, the fundamental building blocks of the ontology, i.e., the elements in its vocabulary, are basic data types (class DataType in Figure 1), facet names, property names, method names and class names. Hereafter, the set of non-decomposable elements of the ontology is termed its basic vocabulary. The set of all concepts of the ontology, either atomic or compound, will be called its extended vocabulary. Therefore the extended vocabulary of a given ontology includes its basic vocabulary.

The degree to which an ontology is translatable to another ontology depends on the mapping between elements in the basic vocabulary of the first ontology into (atomic or complex) elements of the extended vocabulary of the other ontology. Ontology A is completely translatable to ontology B only if there is a total mapping from the elements in the basic vocabulary of A to elements of the extended vocabulary of B. Certainly, this is not a sufficient condition, but it is necessary.

In certain circumstances, it would be possible to infer mappings from the basic vocabulary of one ontology into another ontology from global relationships between the ontologies. The definition of such mappings is out of the scope of the current work. In the remaining of this section, without lack of generality, the mapping from the basic vocabulary of one ontology to the other ontology is known a priori.

Being L(A) the basic vocabulary of ontology A, and V(A) the extended vocabulary of ontology A, we define I_{AB}:L(A)→V(B) as the basic vocabulary mapping from A to B I_{AB} is a partial function that maps elements of the basic vocabulary of ontology A into elements of the extended vocabulary of ontology B.

Having the basic vocabulary mapping I_{AB} it is possible to define relationships between compound concepts of two ontologies with respect to that mapping.

Generally speaking, the relations that will be defined in the remaining of the section are such the relation between two composite concepts is reduced to relations between their parts.

Translation of Properties

Definition 1: Property P1 in ontology O1 is translatable to property P2 in ontology O2, with respect to the basic vocabulary mapping I_{O1,O2} iff the name of P1 is mapped to the name of P2 by I_{O1,O2}, and the type of P1 is translatable to the type of P2, by I_{O1,O2}.

Translation of Facets

Definition 2: Facet F1 of ontology O1 is translatable to facet F2 of ontology O2, with respect to the basic vocabulary mapping I_{O1,O2} iff the name of the facet F1 is mapped to the name of facet F2 by I_{O1,O2}, and the value of F1 is the same as the value of F2.

This definition assumes that the values of facets are not subject to translations, which is somehow restrictive. An alternative approach would be to add the possible values of facets to the basic vocabulary of the ontology. This way, the translation would be handled by the basic vocabulary mapping function IO1.O2.

Translation of attributes

Definition 3: Attribute A1 of class C1 of ontology O1 is translatable to attribute A2 of class C2 of ontology O2 with respect to the basic vocabulary mapping I_{O1,O2}, iff the property associated to C1 by A1 is translatable to the property associated to C2 by A2 with respect to I_{O1,O2}, and for each facet F1.i of A1, there is a facet F2.j of A2 such that F1.i is translatable to F2.j with respect to I_{O1,O2}.

643
**Translation of methods**

Definition 4: Method M1 in ontology O1 is translatable to method M2 in ontology O2, with respect to the basic vocabulary mapping $\text{IO1.O2}$ iff the name of M1 is mapped to the name of M2 through $\text{IO1.O2}$, the return type of M1 is translatable to the return type of M2, and if for each argument A1.i of M1 there is an argument A2.j of M2 such that A2.j is translatable to A2.j with respect to $\text{IO1.O2}$, and if the axiomatizations of M1 and M2, respectively $\Delta_1$ and $\Delta_2$ satisfy the relation $\text{Trans}_{\text{IO1.O2}}(\Delta_1)$ logically implies $\Delta_2$, in which $\text{Trans}_{\text{IO1.O2}}(\alpha)$ is the translation of $\alpha$ of ontology O1 into the correspondent concept in ontology O2 with respect to the basic vocabulary mapping $\text{IO1.O2}$.

The main problem with this approach is to show that $\text{Trans}_{\text{IO1.O2}}(\Delta_1)$ logically implies $\Delta_2$. In general, logical implication is a hard problem. For the first order logic, it is semi decidable. That is, it is possible to create an algorithm that shows that $\text{Trans}_{\text{IO1.O2}}(\Delta_1)$ logically implies $\Delta_2$, if that is actually the case. However, it is not possible to create an algorithm that is guaranteed to stop when $\text{Trans}_{\text{IO1.O2}}(\Delta_1)$ does not imply $\Delta_2$. This solution, although complex, is very elegant and powerful in the sense that it provides an automatic means that allows to determine a solution, although complex, is very elegant and powerful in the current proposal. Those properties are represented by the following statements in which C1 and C2 are classes and M is a basic vocabulary mapping between the ontologies containing C1 and C2:

$$\text{Strongly-Translatable}(C_1, C_2, M) \Rightarrow \text{Weakly-Translatable}(C_1, C_2, M)$$

$$\text{Weakly-Translatable}(C_1, C_2, M) \Rightarrow \text{Approx-Translatable}(C_1, C_2, M)$$

$$\text{Equivalent}(C_1, C_2, M) \Leftrightarrow \text{Strongly-Translatable}(C_1, C_2, M) \land \text{Strongly-Translatable}(C_2, C_1, M)$$

$$\text{Identical}(C_1, C_2, M) \Rightarrow \text{Equivalent}(C_1, C_2, M)$$

With the above definitions of possible relations between classes, properties and methods, it is also possible to improve the definition of the relations between ontologies. These definitions were intended to have a meaning similar to the one expressed in [8], but they are defined using the more fundamental concepts that we have defined (definitions 1 through 9), assuming that all ontologies are expressed using the OntoRF formalism.

Definition 10: An ontology O1 is an extension of ontology O2, with respect to the basic vocabulary mapping $\text{IO1.O2}$ iff all classes in O2 have equivalent classes in O1, with respect to $\text{IO1.O2}$.

Definition 11: An ontology O1 is equivalent to ontology O2, with respect to the basic vocabulary mapping $\text{IO1.O2}$ iff all classes in O2 have equivalent classes in O1 and all classes in O1 have equivalent classes in O2, with respect to $\text{IO1.O2}$.

Definition 12: An ontology O1 is identical to ontology O2, with respect to the basic vocabulary mapping $\text{IO1.O2}$ iff all classes in O2 have identical classes in O1 and all classes in O1 have identical classes in O2, with respect to $\text{IO1.O2}$.

Definition 13: An ontology O1 is strongly-translatable to ontology O2, with respect to the basic vocabulary mapping $\text{IO1.O2}$ iff all classes in O1 are strongly-translatable to classes in O2, with respect to $\text{IO1.O2}$.

Definition 14: An ontology O1 is weakly-translatable to ontology O2, with respect to the basic vocabulary mapping $\text{IO1.O2}$ iff some classes in O1 are weakly-translatable to classes in O2, with respect to $\text{IO1.O2}$.

Definition 15: An ontology O1 is approximately-translatable to ontology O2, with respect to the basic vocabulary mapping $\text{IO1.O2}$ iff some classes in O1 are approximately-translatable to classes in O2, with respect to $\text{IO1.O2}$ and O1 is not weakly-translatable to O2.
Readers acquainted with [8] will notice that our definition of equivalent and identical ontologies do not demand the two ontologies to be expressed using the same language. The reason for this option is twofold: first, we assume that all ontologies are expressed using (or have been converted to) O3F. Second, we consider this to be too restrictive a constraint: in our view, it suffices that two ontologies (initially expressed using different frameworks) map to equivalent or identical ontologies in O3F.

6. EVALUATION OF O3F

This section shows some ontology examples in several well-known ontology representation frameworks, namely Ontolingua and DAML-OIL. The section also summarizes the limitations of these frameworks and presents O3F as an alternative ontology representation framework.

The example in section 2 (partial representation of the Restaurant Ontology) is used here.

Figure 3 presents part of the DAML-OIL Representation of the Restaurant Ontology.

Figure 3 presents part of the Restaurant Ontology represented in DAML-OIL. The Restaurant class of the RestaurantOntology ontology includes a property named maxPrice. Figure 4 presents the same example in Ontolingua.

(1) Relation Restaurant
(Subclass Of Restaurant Individual-Thing)
(Class Restaurant)
(Arity Restaurant 1)
(Range Maxprice Real-Number)
(Domain Maxprice Restaurant 1)

Figure 4 - Part of the Ontolingua Representation of the Restaurant Ontology

The representation of the method BookTable in these ontology representation frameworks is impossible, since Ontolingua can only represent functions, general predicates and axioms, while DAML+OIL is only able to represent classes and properties. To deal with this shortage, we have defined an extension to DAML+OIL that allows the definition of methods and their association with classes [15]. This method would be defined as:

\[
\text{method BookTable}
\]

\[
\text{arguments (set}
\text{ (AArg :class Argument :key name}
\text{ :object table)}
\text{ (AArg :class Argument :key name}
\text{ :object string))) Association)
\]

The description of the argument booker is omitted in Figure 6 but it would be defined similarly to argument table.

7. CONCLUSIONS AND FUTURE WORK

The paper describes an object-oriented ontology representation framework allowing the representation of the most common OO concepts, including classes, properties, methods and their relationships. The main contribution of the paper is the declarative description of action methods, that is, methods whose execution changes the state of the world. The paper shows that the presented proposal improves the autonomy of agents in agent networks because it provides a stronger semantic content than related ontology representation frameworks (e.g., DAML, Ontolingua, and UML). We have also shown that the proposed approach can be used for representing ontologies originally written in DAML+OIL, Ontolingua and UML.

Future steps along this work are the development of software tools that allow the automatic mapping of different surface representation methods into the proposed framework, and using the action method description for automatically generate agent programs in the Pagoda of Creation [12][11].

8. ACKNOWLEDGMENTS

The research described in this paper is partly by UNIDE/ISCTE and partly supported by the EC project Agentcities.RTD, reference IST-2000-28385. The opinions expressed in this paper are those of the authors and are not necessarily those of the Agentcities.RTD partners. The authors are also indebted to all other members of the Agentcities ADETTI team.
9. REFERENCES


