Multilingual Agents: Ontologies, Languages and Abstractions

Steven Willmott*
Laboratoire d'Intelligence Artificielle, Ecole Polytechnique Federal de Lausanne,
Lausanne, Switzerland

Ion Constantinescu
Laboratoire d'Intelligence Artificielle, Ecole Polytechnique Federal de Lausanne,
Lausanne, Switzerland

Monique Calisti
Laboratoire d'Intelligence Artificielle, Ecole Polytechnique Federal de Lausanne,
Lausanne, Switzerland

ABSTRACT

Agent Environments are becoming increasingly open, interconnected and heterogeneous. This suggests that future agents will need to be able to deal with multiple agent communication languages, multiple ways of expressing content and multiple ontology representations.

One way to deal with this heterogeneity is by identifying an agent's internal knowledge representation with an abstract ontology representation (AOR). This AOR then can be used to capture abstract models of communication related knowledge (domain models, agent communication languages, content languages and models of how these interact) and make it possible for the agent to manipulate all elements of messages in a uniform way - as instances of its ontological knowledge.

The paper outlines the approach, highlights interesting issues and describes a prototype implementation.

Keywords

Ontologies, Agent Languages, Content Expressions, Representations, Abstraction, Agent toolkits

1. INTRODUCTION

Recent years have seen significant effort invested in the study of communication mechanisms for agents. Particular attention has been paid to Agent Communication Languages (such as FIPA-ACL and KQML), Content Languages (such as KIF and FIPA-SL) and Ontology representations (DAML, OIL and others). These frameworks can be used to describe both the structure and meaning of the messages agents might exchange. For an agent to make effective use of these frameworks it must be able to construct and manipulate messages combining aspects from all three areas (ontology, content language, agent communication language). Often constraints (structural and semantic) imposed by the frameworks being used must be respected to ensure that the message has meaning.

Whilst this task is difficult enough if Agent Communication Language (ACL), Content Language (CL) and Ontology Representation (OR) are fixed in advance, agents in heterogeneous environments are increasingly likely to be faced with:

- Multiple ACLs, each with multiple possible encodings.
- Multiple CLs, each with multiple possible encodings.
- Multiple ORs, each with multiple possible encodings.[1]

How will agents deal with this heterogeneity? How can agent toolkits support developers in exploiting the different language frameworks? How can we ensure that code is reusable across many agent languages? What is the best way to bridge between the ACL, CL and OR levels? In answer to some of these questions, the central thesis of this paper is that:

1. An agent's internal knowledge representation can be seen as an abstract ontology representation (AOR).
2. This AOR can be used to capture abstract models of communication related knowledge (domain models, agent communication languages, content languages and models of how these interact).
3. This makes it possible for the agent to manipulate elements from all levels of messages (ACL, CL and domain) in a uniform way - as instances of its ontological knowledge.

The main idea is therefore to give agents explicit representations of languages and domains to manipulate at runtime. The approach is described in three parts: abstract ontology representation (Section 3.1), the definition of languages structures as ontologies (Section 3.2) and how these can be used to develop communicating agents (Section 3.3). Section 4 then details a prototype implementation of the architecture supporting all of the main ideas presented and, in particular:

- The abstract ontology representation described in Section 3.1.
- An interface for a restricted version of DAML ontology representation as a reification of the AOR.

*Author for correspondence. Email: Steven.Willmott@epfl.ch

[1]Here an *Ontology Representation* is taken to be defined by the information which can be represented (entities, relations and constraints allowed) and its *encoding* is the physical representation. An Object Oriented OR may have two representations for example: one in UML and another in XMI/XML.

- Conceptual models of the agent languages FIPA-SL, FIPA-KIF, First order logic (content languages) and FIPA-ACL (agent communication language) expressed as DAML ontologies.
- Codecs for the reification of instances of messages in three of the languages listed above to their respective syntaxes (FIPA-SL S-Expression syntax, FIPA-KIF Standard syntax and FIPA-ACL S-Expression syntax).

This work described here is in many ways a logical progression of previous work by others in the following areas: 1) work on modelling agent languages as ontologies such as [1], 2) implementations of Agent toolkits such as Jade [9], FIPA-OS [7] and Jat-Lite [10] which provide ACL, Content Language or Ontology access for specific languages and 3) Ontology frameworks such as DAML, OIL, DAML+OIL and UML approaches which form the basis of the example AOR given in Section 3.1.

It should be noted that the objective of this paper is to provide a global view of the feasibility and potential utility of this approach rather than provide definitive results in any one area.

2. DEFINITIONS

Before delving into the main body of the paper this section defines important terms. The following example of an agent message using FIPA-ACL (S-expression syntax [4]), FIPA-SL [6] and an ontology about cars illustrates one way of seeing the relationship between different levels of agent communication:

```
(inform
 :sender (agent-identifier :name i)
 :receiver (agent-identifier :name j)
 :ontology car
 :language FIPA-SL
 :content
   "((= (any ?x (is-car ?x))
     (car
       :colour lightgrey
       :registration VD 3651
       :make VW
       :type Golf
     )
   )"
)
```

Elements in the world are defined in the *domain ontology* (the predicate "is-car" and the "car" object would be defined in the ontology "car"). A *content language* expression (the argument of the ":content" parameter in the example) is then used to represent and bind instances of these entities together into a statement about the world. Finally a speech act defined in an *agent communication language* expressing an agent's opinion about this state of affairs is wrapped around the content expression. A similar structure exists for messages using, for example, KQML with KIF as a content language. These three levels are also linked by constraints such as:

- In FIPA-ACL, the ACL expresses which content language, encoding and ontologies are to be used in the content field.
- Often the speech act used at the ACL level constrains the type of content allowed - a FIPA "request" should be for an action and not a proposition for example.
- Content Languages usually interface with ontologies via certain constructs able to express entities defined in an ontology (the car construct in the message above is represented by an SL functional term for example).
- Ontologies often express constraints on composition of concepts in the ontology, e.g. that the value of a car's colour attribute must be a "colour" of some sort.

Some of these constraints are expressed in language grammars, others in ontology definitions and others as free text descriptions.

This paper also makes use of the following terms:

- **Agent Language**: a language which is either an Agent Communication Language or a Content Language.
- **Concept (or Class)**: A notion defining a named class of entities.
- **Conceptual Model**: a model (or meta model) containing definitions of a number of concepts and relationships between these concepts.
- **Public**: stable definition (of a language, encoding or other) or structure which is available to all agents in a given community and something which can be relied upon to enable interoperability.
- **Internal**: something which is not necessarily public.
- **Instance Knowledge**: statements about particular instances of concepts (classes). (E.g. "Harry the cat is blue".)
- **Ontological Knowledge (or Meta Knowledge)**: statements about classes and relationships - instances of conceptual models. Meta knowledge w.r.t. instance knowledge (E.g. "Cats are animals", "Blue is a colour".)

3. ABSTRACTIONS AND ARCHITECTURE

The presentation of the approach is divided into three parts: abstract ontology representations (Section 3.1), modelling languages as ontologies (Section 3.2) and usage in agent systems (Section 3.3).

3.1 Abstract Ontology Representations

As shown in Figure 1, most agent architectures store and manipulate knowledge in several areas: conceptual models, knowledge bases and messages involved in communication with other agents. The underlying model the agent uses to represent knowledge of conceptual models of the world is called its "knowledge representation".

Since we are primarily concerned with communicating agents, it is important that this knowledge representation is able to represent ontological knowledge and in particular ontologies defined in one or more public ORs. Given that there are already a number of different public OR frameworks (DAML, OIL, UML based approaches) and that there may be more in the future, choices on what an agent's internal knowledge representation is able to capture is of great importance. Choices here impact both the effectiveness of the agent and

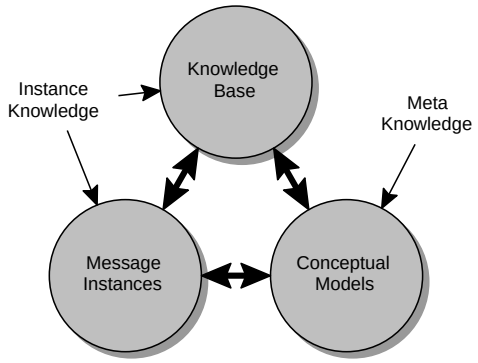

Figure 1: An agent's knowledge is usually divided into instance knowledge (knowledge of individual facts) and meta knowledge (knowledge about classes of entities).

the future re-usability of its code. The problem has two levels as illustrated in Figure 2:

1. **Encoding**: abstracting from multiple encodings of one OR to extract a conceptual model for that OR.

2. **Conceptual Model**: abstracting from the (usually different) conceptual models of multiple ORs to extract a common concept model.

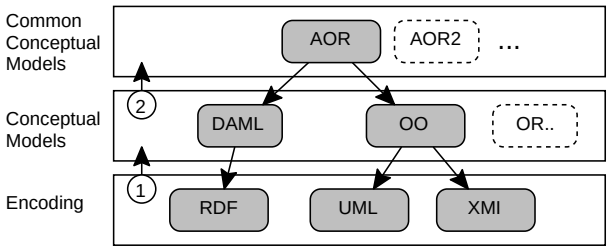

Figure 2: Abstraction from encoding is the first step, followed by abstraction from conceptual models to find common features. The final abstract ontology representation matches the agent's internal knowledge representation.

The product of the second step is then a potential candidate for use as the agent's internal knowledge representation.[2] In the context of this paper it is referred to as an Abstract Ontology Representation (AOR). In general there may be several useful abstractions and more than simply two abstraction steps. We focus on these two major transitions (encoding - conceptual model, conceptual models - unified conceptual model) to simply presentation however.

3.1.1 Abstraction from Encodings

There is clear value in separating internal representations of knowledge from any one particular public encoding since supporting new encodings can then normally be supported by simply adding new codecs.[3] This type of abstraction is generally relatively straightforward since most encodings of a single OR will attempt to express the same elements of the ORs underlying conceptual model.

3.1.2 Abstraction from Conceptual Models

Whilst dealing with multiple encodings is normally relatively simple, dealing with multiple conceptual models of ORs is much more problematic. It is highly likely that some types of knowledge (e.g. certain types of relations or constraints) are allowed by some frameworks and not by others. Java class hierarchies, for example, forbid multiple inheritance but this is allowed by DAML and OIL. There are two straightforward approaches to generating abstractions of a set of *target ORs* (the ORs a designer wishes to consider):

- *Feature intersection*: Supporting representations of only the features which fall into the intersection of features of all target ORs. This approach ensures that all of an agent's ontological knowledge could be represented in any of the target ORs but means that it may not be able to represent/reason with all the information in ontologies it shares with others.

- *Feature union*: Supporting representations of all features of all the target ORs. This extreme ensures that the agent is able to represent all aspects of the knowledge expressed in each of the target ORs. The negative consequence is that there may be subsets of the agent's knowledge which cannot be expressed in any single OR (particularly if the agent is able to derive new ontology information).

There are clearly complex tradeoffs involved here and the choices made are directly linked to an agent's reasoning capabilities and potential functionality. In reality, most agent systems will likely fall somewhere between these extremes and pick and choose OR features to represent.

3.1.3 Example Abstract Ontology

The above are rather general statements, this section defines an abstract ontology representation which is used throughout the rest of the paper. The intention is not to argue that the model chosen is in anyway "the best" for the combination of target ORs but to draw out common aspects which appeared to be necessary for a basic systems. Target ORs are: DAML, OIL, UML, Frames (such as those currently used in FIPA specifications), simple language grammars expressed in BNF/EBNF. Although a detailed study of the similarities between these frameworks is beyond the scope of this paper, a useful intersection appears to include the elements below. The following is the example AOR which will be used from now on:

1. **Class**: corresponding to a concept describing a named class of entities.

2. **subClassOf relations**: indicating that one class is a more specific version of another (can also be read as "can substitute", as in "VW" can substitute "car").

[2]Note that an agent may well use several knowledge representations internally, or that the AOR described here is simply mapped to an internal KR when necessary. One KR assumed here for clarity.

[3]Codec is used to refer to software modules which translate a single public representation to and from an internal data structure.

3. **sameConceptAs relations**: indicating that two classes are identical, includes implict equivalences between properties of the class.

4. **Properties**: corresponds to attribute value pairs which express something about one or more classes. The following constraints can be applied to properties:

   - Domain: the classes the property applies to.
   - Range: the range of values the property may take.
   - Cardinality: the number of times it may or must occur for a given class.

The terminology used here is taken from the DAML+OIL specification [2] but equivalent formulations could be made using (for example) frames and slots.[4] The resulting structure is a directed graph corresponding to a class diagram but which allows multiple inheritance and equivalence.

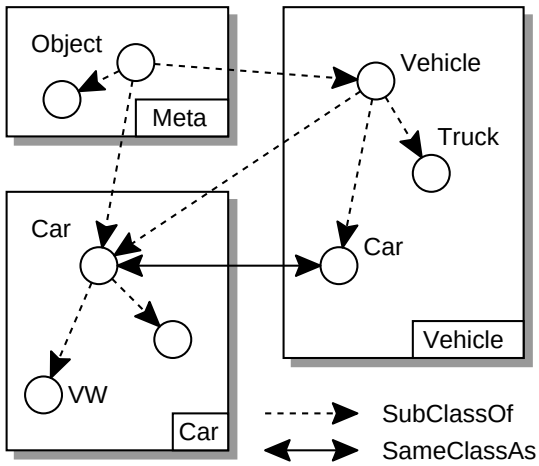

Figure 3: Ontologies are represented as directed graphs.

Figure 3 shows several linked ontology definitions using the model defined. The boxes in the figure each represent a namespace defined by a single ontology definition. In addition to the subClassOf and sameClassAs relations each class may also have properties assigned. There is clearly a lot of richness from the target ORs which is lost is this model but it should be remembered that the main purpose of this representation is to illustrate the approach rather than to propose a specific abstract ontology representation.

3.2 Abstract Agent Languages

The AOR given in the previous section gives the agent a fixed model for its ontology knowledge. The next step is to allow it to do the same with instance knowledge, specifically with instance knowledge expressed in varied agent languages (we are less concerned with knowledge bases). For any given language, this problem is again at two levels: encoding and conceptual model.

3.2.1 Abstraction from Encodings

In principle this can be solved as simply as for ontologies - by presuming a common representation of message elements which are extracted from diverse encodings by encoding specific codecs. The question of how the conceptual model of a language is expressed arises however. This question is more obvious for languages than for ontology representations since:

- Language grammars can be relatively complex and capture a good deal of information about the meaning of expressions in the language
- Agents will often need to manipulate instances of messages and require access to the conceptual model of the language to ensure correctness.
- The interaction between agent languages and between agent languages and entities defined in domain ontologies are particularly important.
- Language syntax definitions come in a variety of forms (XML Schemas, DTD, EBNF grammars etc.) but these tend to be highly dependent upon the individual syntax involved.

As Cranefield et. al. point out in [1] however, ontologies can in fact be seen as abstract grammars for languages. An OR can be used to construct conceptual models of languages. A logical usage of this is to give the agent access to these language models *at runtime* and allow it to manipulate them. This enables the agent to treat knowledge about languages it knows at the same level as domain knowledge. The objective is to model languages in a formalism compatible with the AOR defined in the previous section. For a language based on an EBNF grammar a first pass at generating the model could be done as follows:

- **Disjunctions become Classes**: Each disjunction on the right hand side (RHS) of a production rule in the grammar can be used to generate a new class. The class name can be derived either from the first constant symbol on the RHS of the expansion or (if there is no such constant), from the name of the non-terminal on the LHS of the production.
- **Elements become Properties**: The elements in the expansion of a single RHS disjunction to the new class each generate a new property, s.t. the property's domain = class generated by the disjunction, range = type of the element, cardinality is determined w.r.t. cardinality expressed in EBNF (+, * etc.).
- **Generating the Class Hierarchy**: The concepts so generated can be linked by subClassOf relationships which express which grammatical elements can be substituted for others (e.g. in FIPA-SL, a term may be replaced by a constant, giving rise to a relation defining constant as a sub class of term.)

Similar schemes could be defined for XML Schema and XML DTDs. This method is only a sketch and needs to be applied with interpretation by the designer to deal with ambiguities it might generate. Furthermore, grammars are often optimised or compacted to reduce overhead. It may be necessary to restructure the grammar to generate a clear conceptual model.

It should be clear what is being done here - the conceptual model represents *only* the syntactic/lexical constraints

[4] The terms "class" and "concept" are used interchangeably throughout the paper.

between the concepts in the language. The information so generated does include a important part of the language structure (such as, for example, the fact that an "+" operator may only take "numbers" as arguments) but it does not capture more than an EBNF or Schema grammar - i.e. it does not attempt to capture the semantics. The concepts extracted for a language such as FIPA-SL include classes such as:

```
Class: "BinaryTermOp"
 SubClassOf: AtomicFormula
 Property: argument
  range: Term,
  Cardinality: 2.

Class: "="
 SubClassOf: BinaryTermOp

Class: "Term"
```

Where there are a number of sub classes of "Term". Concepts such as BinaryTermOp and Term are never normally instantiated in messages but are useful to structure the model of the language. In the case of Term for example it is clearly useful since it is used in the definition of the equals operator.

3.2.2 Abstraction from Conceptual Models

A further reason for considering how conceptual models of languages are expressed (and made available to the agent) is that the approach can be re-used for modelling abstract languages and how they relate to each other. An example of a conceptual difference between two languages is the belief modal operator "B" which is available in FIPA-SL but not defined in KIF. This means that:

- "(and X Y)" is defined in both languages.
- "(B fred (and X Y))" is only defined in FIPA-SL.

The reason "defined" is used and not "expressed" is that the second statement can be expressed in KIF. KIF simply attaches no special meaning to the belief operator.

FIPA-SL and KIF are in fact a good example for the potential use of abstraction since they are both extensions of First Oder Logic (FOL - see [8] for example). Due to this common heritage, FOL concepts such as *term, variable, constant, predicate* appear in both FIPA-SL and KIF and account for a significant subset of both languages. As noted above, in many cases specific constructions such as "member" etc. in FIPA-SL and KIF are captured in a more general way by FOL functions.

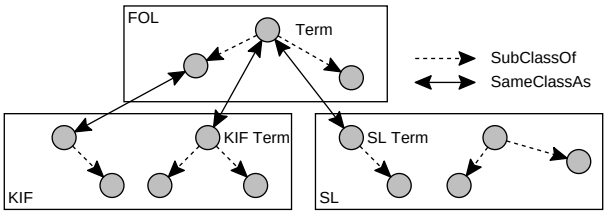

Figure 4: Hierarchical ontology definitions for FOL, FIPA-SL and KIF.

Figure 4 sketches how languages might be defined as three separate but linked ontologies. Although FOL cannot be said to abstract all of KIF and FIPA-SL it is clearly more abstract. Agent code manipulating messages using only FOL concepts would be more *language independent* than code using using concepts which appear only in SL or KIF respectively (and not in FOL). Messages using concepts only from FOL could be represented in both SL and KIF and hence any of their encodings (potentially also in potentially other FOL derivatives such as Prolog). Definitions of languages as hierarchies of ontologies could also be applied to non-logical languages and at the ACL level - there are performatives in KQML and ACL which are roughly equivalent (such as "tell" and "inform") and others which are only found in one of the languages (such as "stream").

Relationships between concepts in different languages become easy to model once a common meta model is used to express conceptual models. In particular the sameClassAs relation can be used to match equivalent concepts in different languages. This is especially important for declaring mappings between concepts such as objects, actions and functions which may be declared in a domain ontology and the concepts in any given language which may represent them.

The most abstract (general) language allowed by the AOR given in Section 3.1 appears to be:

```
Class ::= ( ClassName Class* )
```

This provides a lot of leeway for different levels of abstraction of languages. It should be remembered, however, that languages tend not to be very useful unless specific semantics are attached to the concepts involved and the range of things which can be expressed is well delimited. FOL is also an especially good example because it covers a well defined range of expressions with known computational properties. It seems likely that the languages agents may know will not form a neat hierarchy of abstractions but a patchwork of communications concepts. This is especially true if, as suggested in [1], models of language are exploited to construct ad-hoc domain specific content languages linking conceptual models of languages with domain models. Agents could then potentially construct application specific languages at runtime (to negotiate a particular contract for example).

3.3 Building Multilingual Agents

Up until now there has been little discussion on how these developments impact the building of agents. In principle it is possible to now limit an API for manipulating messages to two areas:

- **Ontological/Meta Knowledge**: accessing and manipulating stored ontology information stored in the AOR representation.
- **Instance Knowledge**: accessing and manipulating instances of concepts defined in the ontology knowledge.

Again, the important point is that both areas (ontology and instance) represent ACLs, CLs and domains - removing the need for Developer interfaces for each new language, OR or domain the agent wishes to deal with. Developer code is isolated from communication details such as syntax and perhaps language if abstract languages are defined and used. A simplified view of the architecture is shown in Figure 5.

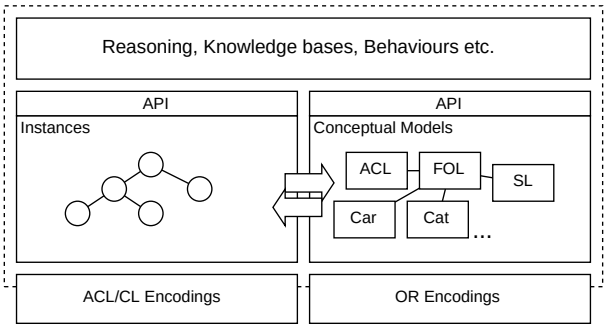

Figure 5: Development intensive aspects of the agent implementation such as behaviours and reasoning are built on APIs accessing conceptual models and instances of concepts (classes). These in turn are abstractions and independent of any particular language or encoding.

As the figure shows, there would usually be interaction between the instances of messages / concepts and conceptual models. The most obvious uses of the models applied to instances are:

- **Validation**: Checking that a particular combination of instances of concepts respects all the constraints imposed by the applicable models. This is the most basic usage of the conceptual models and very important for languages modelled in particular since it allows agents to recognise whether a particular message (incoming or outgoing) respects the constraints of a particular language or not.
- **Translation**: Equivalences expressed in conceptual models can be used to map instances from one model (ontology) to another, in terms of language for example - mapping from SL to FOL could render a incoming message suitable for an FOL theorem prover the agent has built in.
- **Generation**: When constructing messages it can be useful to be able to generate the concepts which could be inserted at a particular point in a message.
- **Learning**: If a message instance arrives which references the concept "orange" in ontology "fruit" (unknown to the agent) the agent may be able to read in the referenced ontology and find a link between "orange" and the "food" concept in some more general ontology it already knows. This allows it to infer that the construction "(eat sally orange)" makes some sort of sense even if it has only a superficial understanding of "orange".

There is also no reason why additional information about domain ontologies or languages could not also be encoded in message instances. This information could then be assimilated into existing conceptual models, mixing the sources of the two types of knowledge. Further processing of instances could include variable scope management, unification, simple evaluations (e.g. executing associated functions in arithmetic etc.). Add on features for the conceptual models could include summarisation of knowledge, forgetting, macros to extract well known schemas etc.

4. IMPLEMENTATION

This section describes ATOMIK[5] which is a simple prototype implementation of the architecture presented in Section 3. ATOMIK is intended to:

- Provide a proof of concept for the ideas presented in this paper.
- Act as an additional illustration of the approach (the source code is available - see Section 4.3).
- (if it proves useful) be evolved into a library which could be plugged into existing agent toolkits.

4.1 Overview

ATOMIK has a modular architecture and implements the following:

- A kernel providing: an implementation of the AOR described in Section 3.1 as a compact directed graph (ontological knowledge) as well as an API for creating instances of concepts and composing them (instance knowledge).
- A validator module which is able to the check all the relationships defined in known ontologies are respected by a particular combination of concept instances.
- A codec for a subset of the DAML+OIL Ontology Representation in RDF which corresponds to the AOR (so ontologies can be loaded and saved from DAML files)
- Conceptual models (coded as DAML ontologies) and codec modules for the following languages:
  - FIPA-SL [6].
  - A subset of FIPA-KIF [5], corresponding to SKIF (a limited form of KIF covering only KIF sentences).
  - FIPA-ACL [3], S-expression syntax [4].
  - FOL ([8] conceptual model only - no codec since it is used internally only).
- A simple agent able to perform tests of the above functionalities.

The implementation is in Java[6] (version 1.2) and uses the SIRPAC RDF package[7] to handle parsing for DAML RDF. All other codecs were implemented using JavaCC[8].

4.2 Examples of Operation

The following examples illustrate the current status of ATOMIK's capabilities based on the languages/representations currently available (FIPA-SL, KIF, ACL, DAML):

[5]Obligatory hastily chosen acronym: "AgenT Ontology ManipulatIon Kernel".
[6]The system could perhaps have been even more easily written in Prolog, Lisp, Scheme etc. but Java was chosen simply because most current agent toolkits are Java based.
[7]See: http://www.w3.org/RDF/Implementations/SIRPAC
[8]See: http://www.metamata.com/JavaCC/

- Validation of messages involving concepts from several ontologies. Given a definition for a car ontology specifying that the colour property of a car must contain a colour concept found in a second ontology and has defined colours red, yellow, blue. ATOMIK is able to tell that:
  - "(for-sale (car :colour black))" must be bogus but that:
  - "(for-sale (car :colour blue))" is potentially a legitimate statement.
- Detecting messages which are invalid according to a particular language grammar. The following are detected as illegal in FIPA-SL for example:
  - "(forall days (make-tea :milk true :sugar false))". Fails because "days" is not a variable.
  - "(or A B C)". Disjunction in FIPA-SL is a binary operator.
- Checking validity of complete FIPA-ACL messages such as:

```
(request
 :sender (agent-identifier :name j)
 :receiver (set (agent-identifier :name i))
 :ontology car
 :language FIPA-SL
 :content
   "((action (agent-identifier :name i)
     (lend-to
           (agent-identifier :name j)
           (car
             :colour lightgrey
             :registration VD 3651
             :make VW
             :type Golf
            ))
     ))"
)
```

  Including checks on the ACL level concepts, content lanaguage expressions and ontology aspects.
- Reifying expressions in FOL into FIPA-KIF and FIPA-SL: the expression "((not X) => Y)" can be constructed using concepts only from FOL and mapped into both FIPA-KIF and FIPA-SL:
  - in FIPA-KIF this becomes: "(=> (not X) Y)".
  - in FIPA-SL this becomes: "(implies (not X) Y)".

  In fact the syntaxes of FIPA-SL and KIF are very similar for the subset of elements which correspond to FOL hence this flexibility does not appear very useful. For other languages with other syntaxes however (or other syntaxes) this capability is likely to be of great importance.

In summary, the prototype is able to check correctness of constructions in the defined languages based on their conceptual models, link language concepts to domain concepts, check constraints imposed by domain ontologies. translate between languages in a limited way and make use of abstract languages.

The most important things the prototype cannot do are related to the AOR model applied, it cannot:

- Cope with variable scoping - i.e. it is not possible to check in FIPA-SL that variables used in a message instance have been declared or not.
- Enforce semantic constraints between (e.g.) performatives and content.

These arise because they are constraints which cross more than one node in the graph (i.e. they do not apply from a concept to one of its properties). In principle those named are not difficult to add but a more general solution would be preferable.

4.3 Resources

Since this paper can onl give an overview of the implementation work done, the following can be found on-line at http://liawww.epfl.ch/ATOMIK/:

- Full source code to the ATOMIK implementation (LPGL license)
- DAML ontology definitions for: FIPA-SL, FIPA-ACL, FOL, KIF and several example domain ontologies

We hope these will provide a useful support to the issues discussed in this paper and might be re-used by others implementing agent systems.

5. INTERESTING ISSUES

The work presented here raises a number of interesting issues:

1. *What should the AOR include/exclude?*: As noted in Section 3.1 the choice of internal representation has a profound effect on the agent system. Although it is "internal" and not shared with the outside world choices by agent toolkit developers will have a effect on how large numbers of agents may use of ontologies defined.
2. *Do Agents need to share AORs?*: In principle AORs are internal representations and as such do not need to be public. Agents working in mission critical areas however are likely to need to find ways of establishing a lowest common denominator of understanding for a given set of ontologies to ensure both can model the others perception of the situation.
3. *What types of Agent Languages can be represented?*: Since the AOR as described can in principle model the main features of a wide range of languages (logical, functional and object oriented). How does this relate to other meta-modelling work (e.g. [1]) and what aspects of languages which could be regarded as part of the conceptual model cannot be represented (variable scoping for example).
4. *How can we cope with equivalences between groups of concepts?*: Currently the AOR chosen allows class equivalence, more generally however a combination of concepts in one ontology may be equivalent to a combination of concepts in another.

5. *How could semantic constraints be represented in the OR?*: How could constraints such as the fact that a FIPA-ACL "inform" speech act may only contain a proposition as content be generically represented in the ontology definition?

6. *What happened to the semantics anyway?*: As with current language descriptions the conceptual models only capture the basic concepts of a language and their "syntactic" relationships but say nothing about how semantics might be managed or enforced. Although beyond the immediate scope of the paper it would be interesting to see if the abstractions described would support effective semantic checking tools.

7. *How easy/valid is it to generate abstractions from CLs?*: Extracting FOL concepts as common to SL and KIF is clearly a special case. Although there may be others (such as predicate logic from FOL) it is not clear how easy it is to perform this abstraction in general. It is also not clear that there will always be neat 1-1 mappings between language concepts.

8. *How valid is it to construct CLs on the fly?*: As discussed in Section 3.2, defining languages as ontologies makes it easy in principle for agents to put together arbitrary combinations of language concepts at runtime. Although this may be useful it could clearly lead to the construction of intractable languages - could this process be guided to allow agents to reason about the power of the languages they create?

9. *How should ontology definitions be linked?*: Like their counterparts in DAML, subClassOf and sameClassAs relations allow linking of entities in different ontologies - this is important to (for example) infer that objects from a domain ontology can be referenced in a content language statement such as "(= (car :colour red) fashionable)". It appears to be useful to identify a set of common concepts which act as bridging points between ontologies. What should these concepts be? How many should there be and at what granularity?

10. *How can concept definitions be linked to functions and Actions?*: two important classes of ontological entities are likely to be functions (e.g. "(+ 1 1)") and actions (e.g. "paint", "eval"). These are things which may have computation or activities to carry out associated with them. It appears to be useful therefore, to have a general mechanism for linking concept definitions with function and action definitions (and/or code).

6. CONCLUSIONS

There is no doubt that agents will need to deal with considerably heterogeneity at all levels of communication and that they will need to effectively compose agent communication language, content expressions and domain knowledge. This paper proposed strategies for equipping agents with flexible communications interfaces which:

- Isolate code intensive areas such as reasoners, theorem provers and agent behaviour from languages and encodings.
- Give the agent access to explicit representations of conceptual models of all levels of communication.
- Allow manipulation of message instances at a single uniform level.

The paper describes the approach and a prototype implementation. Future work includes:

- Theoretical: more detailed investigation of the link between AI knowledge representation and ontology frameworks, requirements for representing conceptual models of languages, mechanisms for linking language and domain ontologies.
- Development: potential integration with existing agent toolkits, developing a stable API which could be used to interface with existing AI/Agent tools such as theorem provers, planners etc.
- Application: testing the resulting system on a significant project - one potential example being a multi-way translation agent able to act as a gateway between groups of agents using different languages and/or ontology representations.

7. ACKNOWLEDGEMENTS

Many thanks go to Fabio Bellifemine, Federico Bergenti, Giovanni Caire, Giovanni Rimassa and Tiziana Trucco for interesting discussions on the issue of agent language support in FIPA compliant agent platforms. Thanks also to the reviewers for their useful comments.

8. REFERENCES

[1] S. Cranefield, M. Purvis, and M. Nowostawski. Is it an Ontology or an Abstract Syntax? - Modelling Objects, Knowledge and Agent Messages. In *Proceedings of the Workshop on Applications of Ontologies and Problem-Solving Methods, 14th European Conference on Artificial Intelligence (2000) 16.1-16.4.* 2000.

[2] DAML. Darpa Agent Markup Language: DAML+OIL specification v 1.7. Technical report, DAML Project, 2001.

[3] FIPA. FIPA ACL Message Structure Specification (00037). Technical report, Foundation for Intelligence Physical Agents, 19.

[4] FIPA. FIPA ACL Message Representation in String Specification (00070). Technical report, Foundation for Intelligence Physical Agents, 2000.

[5] FIPA. FIPA KIF Content Language Specification (00010). Technical report, Foundation for Intelligence Physical Agents, 2000.

[6] FIPA. FIPA SL Content Language Specification (00008). Technical report, Foundation for Intelligence Physical Agents, 2000.

[7] FIPA-OS. FIPA OS v1.3.3. Technical report, FIPA-OS Open Source Team, 2000.

[8] M. Genesereth and N. J. Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann, 1988.

[9] Jade. Java Agent Development Environment (JADE) v2.0. Technical report, Jade Open Source Team, 2000.

[10] C. Petrie. Agent-based engineering, the web, and intelligence. *IEEE Expert*, 11(6):24–29, Dec. 1996.