Second International Conference on Autonomous Agents and Multi-Agent Systems

3rd Workshop on Ontologies in Agent Systems

Sofitel Melbourne

15th July 2003

Stephen Cranefield, Tim Finin, Valentina Tamma and Steven Willmott (editors)

Preface

There is a growing interest in the use of ontologies in agent systems as a means to facilitate interoperability among diverse software components, in particular, where interoperability is achieved through the explicit modelling of the intended meaning of the concepts used in the interaction between diverse information sources, software components and/or service-providing software. The problems arising from the creation, maintenance, use and sharing of such semantic descriptions are perceived as critical to future commercial and non-commercial information networks, and are being highlighted by a number of recent large-scale initiatives to create open environments that support the interaction of many diverse systems (e.g. Agentcities, Grid computing, the Semantic Web and Web Services). A common thread across these initiatives is the need to support the synergy between ontology and agent technology, and increasingly, the multi-agent systems and ontology research communities are seeking to work together to solve common problems.

This workshop is the third in a series of workshops on Ontologies and Agent Systems (the previous workshops were held at the International Conference on Autonomous Agents 2001 in Montreal, Canada and the International Conference on Autonomous Agents and Multi-Agent Systems 2002 in Bologna, Italy). It aims to provide a forum to foster discussion on the issues involved in using ontologies to support interactions between software agents. Emphasis will be on the discussion of ontologies with respect to the practical impact they have on agent architecture and application design.

A new initiative this year was the introduction of a workshop challenge problem to discuss the design and (preferably) an implementation of a multi-agent system in the domain of travel booking information and services, based on a domain description previously developed for an ontology tool assessment exercise organised by the EU OntoWeb project's SIG on enterprise-standard ontology environments. The details of the challenge problem can be found on the OAS'03 workshop Web page at http://oas.otago.ac.nz/OAS2003/. Although there were only two papers that responded to the challenge problem, we hope that the challenge will provide a focus for wider discussion throughout the workshop of the issues of building agent systems that use ontologies.

Programme Committee

The Programme Committee was chaired by Stephen Cranefield, Tim Finin, Valentina Tamma and Steven Willmott and comprised:

Richard Benjamins

Federico Bergenti

Luis Botelho

Monique Calisti

Ulises Cortes

Ian Dickinson

Noriaki Izumi

Yannis Labrou

Frank McCabe

Marian Nodine

Natalya Noy

James Odell

Martin Purvis

Leon Sterling

Heiner Stuckenschmidt

Mike Uschold

We are grateful to the above for their help in reviewing the papers and for their support for the workshop.

July 2003 Stephen Cranefield, Tim Finin, Valentina Tamma and Steven Willmott

Table of Contents

Regular papers

Challenge papers

Short papers

Dealing with Mathematical Relations in Web-Ontologies

Muthukkaruppan Annamalai
Department of Computer Science & Software Engineering
The University of Melbourne
Victoria 3010, Australia
mkppan@cs.mu.oz.au

Leon Sterling
Department of Computer Science & Software Engineering
The University of Melbourne
Victoria 3010, Australia
leon@cs.mu.oz.au

ABSTRACT

The growing use of agent systems and the widespread penetration of the Internet has opened up new possibilities for scientific collaboration. We have been investigating the role for agent systems to aid with collaboration among Experimental High-Energy Physics (EHEP) physicists. A necessary component is an agreed ontology, which must include complex mathematical relations involving such quantities as the energy and momentum of elementary physics particles. We claim that the current web-ontology specification languages are not sufficiently expressive to be useful for explicit representation of mathematical expressions. We adapt some previous work on representing mathematical expressions to produce a set of mathematical representational primitives and supporting definitions that will allow knowledge sharing in agent systems. The paper sketches out a scheme for dealing with mathematical relations in scientific domain web-ontologies, illustrated with examples arising from our interactions with the EHEP physicists.

1. INTRODUCTION

The growing use of agent systems and the widespread penetration of the Internet has opened up new possibilities and created new challenges for scientific collaboration. On one hand, it is possible to make large amounts of scientific analyses of Experimental High-Energy Physics (EHEP) experiments available to scientists around the world [3, 4, 7]. On the other hand different scientific groups, even within a single collaboration, utilise different calculation methods, and it is sometimes difficult to know how to interpret particular analyses. It is assumed that practitioners in this domain possess the necessary background knowledge to interpret the intended meaning of the appropriated jargon in the domain of discourse. Unfortunately, application developers, newcomers to this field, and software agents lacking in relevant expertise are not capable of making a similar kind of interpretation. Knowledge models, or ontologies built to express specific facts about a domain can serve as the basis for understanding the discourse in that domain [5].

The notion of ontology as specification of a partial account of shared conceptualisation [13, 16] is adopted in this paper, that is an ontology defines a set of representational vocabulary for specific classes of objects and the describable relationships that exist among them in the modelled world of a shared domain.

In 2002, we were involved in a project [1, 8] supported by the Victorian Partnership for Advanced Computing [21] to investigate whether ontologies would be useful for EHEP collaboration, in particular in the Belle [4] collaboration. The research is founded on the idea that suitable web-ontologies be developed and reused to facilitate this scientific community to produce and share information effectively on the semantic web. While the final verdict has not been reached, it is clear that our project extends existing capabilities of web-ontology specification languages and tools.

One concerning issue is as to how to define 'function-concepts' in the EHEP ontology, that is concepts equated with mathematical expressions involving such quantities as the energy and momentum of elementary physics particles. A function-concept is a manifestation of an n-ary mathematical relation or function binding a set of quantifiable terms defined in the ontology. The current web-ontology specification languages, such as DAML [17] and OIL [10] are founded on Description Logic [2] and can only express unary and binary relations. They offer no representational features for expressing functions. We ask, "How do we facilitate the semantic recognition of function-concepts in web-ontology?" The answer to this question led us to cognise the extensions required for additional expressivity in web-ontology. This paper shows the approach we have taken in order to ensure that we could describe mathematical relation underlying a function-concept in the ontology.

This paper is organised as follows. In Section 2, we provide a glimpse of the function-concepts in the EHEP domain. In Section 3, we describe the principles of dealing with mathematical relations in web-ontology and our approach to the problem is elaborated in Section 4. Finally, Section 5 concludes the contribution of this paper.

2. EVENT-VARIABLES IN THE EHEP ONTOLOGY

Mathematics plays a significant role in the EHEP experimental analysis – from the time sensor data is captured up to statistical analysis and systematic error calculation. In this paper however, we limit our scope of discussion to the treatment of mathematics in the vital event selection phase.

The superfluous event data captured by detectors is systematically filtered to suppress much of the background events, while preserving the vital signal events. Parametric restrictions on the event selection variables or 'cuts' are utilised to sift the signal events from background events. Loose cuts (or

skimming), followed by more decisive topological, kinematic and geometric cuts is aimed to produce a set of desired event data, fit for justification of the empirical findings.

A category of event selection variables is defined in the EHEP ontology. These event-variables are identified based on their use and need to specify the event selection or background suppression cuts. A set of competency questions [15] drawn-up while the ontology being built guides the conceptualisation of these required event-variables. A typical competency question is: "What are the kinematic selection criteria applied in an analysis?" The ensuing answer would list the selection criteria enforced, such as: "Beam constrained mass $M_{bc} > 5.2\ GeV/c^2$, Track transverse momentum $P_T > 100\ MeV/c$, Energy difference $\Delta E < 0.2\ GeV$, Likelihood of electron over kaon $L_{e/K} > 0.95$". The ontology must define the necessary vocabulary to represent the competency questions that arose and the answers that were generated.

Some event-variables are constants, while others are functions. In the above illustrative example, $L_{e/K}$ is a constant; whereas, M_{bc} is a function event-variable. $M_{bc} = \sqrt{E^2_{beam} - P^2_{3_B}}$, where E_{beam} is the energy of the beam and P_{3_B} is the 3-Momentum of B particle. E_{beam} is a constant. The 3-Momentum $P_3 = \sqrt{P_x^2 + P_y^2 + P_z^2}$ is a function. In here, P_x, P_y and P_z are constant event-variables that denote the individualised track momentum along the x, y and z axes, respectively.

Note that a function event-variable is expressed algebraically in terms of constant event-variables. A simple function event-variable accepts only constants as parameters. Examples are 3-Momentum, P_3 and Transverse Momentum, P_T that describe the momentum of a track. The function P_3 is given above, while $P_T = \sqrt{P_x^2 + P_y^2}$.

On the other hand, a higher-order function event-variable also admits other functions as parameters. The M_{bc} is a higher-order function. One of its parameter is P_3, a function event-variable. Another example is the Fox-Wolfram Moment-0 event-variable, $H_0 = \sum_j \sum_k (P_{3_j} \times P_{3_k})$, whose parameter is a set of 3-Momentum of tracks. The indices j and k enumerate the tracks in an event.

Sometimes, a function requires weighted parameters. An example is the Fisher Discriminant event-variable $F = \sum_i (\alpha_i \times R_i)$, which combines a set of correlated event-variables R_1, R_2, etceteras to form a single variable. As in the previous case, the index i enumerates the considered event-variables. The coefficient α_i denotes the weight for the parameter R_i.

3. THE MAIN PRINCIPLES OF OUR APPROACH

We are focusing on explicating mathematical relations that bind a set of the scientific domain concepts in web-ontology. We seek to find a formal way to represent the mathematical relationships among domain concepts. The idea is to make clear this factual knowledge to humans and software agents during event analyses.[1] Our development drew heavily from the EngMath ontology [14], an extensive past attempt to capture the semantics associated with mathematical expressions in engineering models. EngMath is a declarative first order KIF [11] axiomatisation, which is supported by sets of theories for describing physical quantities, mathematical object such as scalar, vector, tensor and, functions and operations associated with them.

To a lesser degree, general-purpose ontologies like CYC [9] and SUMO [20] also attempt to declaratively capture the semantics of 'evaluatable' function. However, such function is categorised as unary, binary, ternary, quaternary and continuous types; thereby placing a limit on the number of its argument. Perhaps, this restrain is imposed in order to refer to a function's argument based on its order in the 'argument list'. In our case, a function argument must be able to denote a collection of class instances, which SUMO does not allow. In EngMath, such collection is represented as tensor or vector.

Although, the EngMath approach of providing rigorous descriptions appears to faithfully represent mathematical expressions in instantiated models, it is difficult to understand and apply to working systems. The theories represented in declarative style, as axioms (KIF sentences) are hard to read and understand, more so by physicists who are not well versed with ontology. Furthermore, it is not feasible to port this ontology on the web because web-ontology languages do not provide for definitions of arbitrary n-ary relations and functions, and axioms that make up EngMath.

Since we aim to build suitable EHEP ontologies for the semantic web, we parted from the EngMath approach from the outset. The contrast between EngMath and our modelling principles are described below.

I. Web-ontologies are serialised in XML [22], a mark up language intended to encode metadata concerning web document.

II. The domain concepts are assembled in a hierarchy and their distinguishing properties and relationships among them are specified, using a frame-like syntax. In light of this fact, we have conceived the concepts using notions like class, subclass, range-relation and cardinality. (See the examples in next section)

III. A mathematical relation is closed on a function-concept in the ontology. It explicates the algebraic expression used to derive the function-concept. The knowledge associated with this derivation is representationally attached to the function-concept.

IV. The quantifiable terms in the ontology are modelled upon mathematical data types such as scalar, boolean, record (cartesian product), set and function. The types specify how to interpret the values of these terms and restrict the set of operations that can be applied on them.

[1]Note that our development is quite different from the work being pursued by the mathematical-knowledge representation community. Their work is concerned about content theories of mathematics and experiment with different formalisms for representing theories, definitions, axioms, proofs, etceteras in the domain of mathematics.

V. We also recognise the need to extend the algebra of primitive data type operators to constant quantities.

VI. We have introduced the notion of a parameter, which is associated with function quantity. We have classified the parameters of a function quantity as *individual*, *collection* and *weighted* parameters to distinguish ones role from the other.

VII. The arithmetic-logical expression denoting the intension of a function quantity will be encoded in a suitable format (not KIF expression). A possible candidate is OpenMath [19], an XML standard for exchanging mathematical objects on the web.

4. EXPLICATING MATHEMATICAL RELATIONS IN ONTOLOGY

The constant and function event-variables mentioned in Section 2 are physical quantities. We need to first define suitable concepts for representing these quantities in the ontology. Then, we will propound a scheme for providing an abstract description of mathematical relations involving these quantities.

4.1 Representation of Quantity

The need to deal with physical quantities in scientific ontologies is obvious. Our conceptualisation of physical quantity is different from that in EngMath. The definition of physical quantity is based upon our viewpoint on how quantities, dimension and units are treated in the EHEP domain.

For this, we defined a metaclass called *PhysicalQuantity*, having *magnitude* and *dimension* as its attributes (or properties). The concept-oriented definition shown in Figure 1 is an abstract idea of a physical quantity described in a Protege [12] -like representation.[2] A physical quantity has a scalar (integer or real) magnitude and its unit of measure is associated with a particular physical dimension specified in the ontology.

class PhysicalQuantity		
Property	**Range Relation**	**Cardinality**
magnitude	Scalar	= 1
dimension	DimensionUnit	= 1

Figure 1: Definition of PhysicalQuantity

A constant quantity is a representation of a quantifiable object in the domain, which holds a single constant value of measure. The metaclass *ConstantQuantity* is defined as a 'concrete' subclass of *PhysicalQuantity* and a constant quantity can be directly instantiated from it. The constant event-variables defined in the ontology are constant quantities. For example, the fundamental track momentum, *Momentum-X* defined in Figure 2, is a constant quantity. A *Momentum-X* object's magnitude is a real value and its unit of measure is given in terms of units of momentum dimension. Note however that *magnitude* and *dimension* properties are inherited from *PhysicalQuantity*.

[2]The semi-formal ontology developed using this frame-based ontology modelling tool, will be eventually formalised as EHEP web-ontology.

class Momentum-X : ConstantQuantity	
subClassOf	Momentum
Property	**Range Value**
magnitude	Real
dimension	MomentumUnit

Figure 2: Definition of Momentum-X

The function event-variables defined in the ontology are function quantities. A function quantity is distinguished from constant quantity by the fact that the value of measure of the denoted quantifiable object is derived from other constant quantities. In other words, a function quantity can be seen as a function that maps one or more constant quantities to a constant quantity. Consequently, a grounded function quantity can be casted into constant quantity by anchoring it in the domain ontology. The function quantity is elaborated in Section 4.3.

Each quantity belongs to exactly one dimension. A unit is a measure of quantity in some dimension. There is a range of units associated with each dimension in the EHEP ontology. For example, the units for Energy dimension are eV, KeV, MeV, GeV and TeV.[3] Relativistic physics asserts that $Energy = Momentum \times SpeedOfLight$. Accordingly, the Momentum units such as eV/c, KeV/c, MeV/c etceteras are always derived from Energy units in this domain. Likewise, $Momentum = Mass \times SpeedOfLight$. So, Mass units such as eV/c^2, KeV/c^2, MeV/c^2 etceteras are always derived from Momentum units.[4]

Note that in this system of units, the symbol **c**, which denotes the speed of light is featured as part of the unit. This rather unconventional way of describing the unit of a physical quantity in terms of abstract dimension symbol allows the physicists to work with a limited set of dimension unit symbols. It also facilitates the dimensional analysis of algebraic expressions involving related quantities. It would not be necessary, in our case to compose the dimension of a derived quantity from a larger set of fundamental dimensions as prescribed in EngMath and SUMO sub-model on 'Unit of Measure'. The onus of dimensional consistency in algebra over the quantities rests with the modeller. However, the ontology is needed to support the unit analysis involving those quantities.

Based on the above standpoint, we have assigned a set of canonical units to each of the dimensions that appears in the ontology. In order to facilitate unit analysis over a dimension, a base unit is selected for each dimension. The other units of the dimension are then specified as conversion factor from a unit to the base unit. For example, the base unit for Energy dimension is eV. Its succeeding units, KeV, MeV, GeV and TeV are 10^3, 10^6, 10^9 and 10^{12} multiples of the base unit, respectively. Quantities to be manipulated

[3]eV stands for electronvolt, is a non-standard unit for energy. The SI unit for energy is J (joule).

[4]The SI units for mass is kg (kilogram), and momentum is Ns (newton second) or $kg.m/s$, depending on the dimensions that are employed in its derivation. EHEP physicists are parsimonious in the use of dimensions so as to avoid such indeterminate situations.

must be same dimension and unit. For instance, the difference in their conversion factor can be used to reconcile the quantities before they are operated upon.

4.2 Quantity and Data Types

Data type identifies the type of values that may be assumed by quantifiable objects and expressions in the ontology. We will make use of mathematical data types [6] to model these sets of values. The primitive data types are scalar and boolean, while the composite data types are record and set.

4.2.1 Primitive Data Type

The simplest data types are scalar and boolean types. The scalar types are Real and Integer. The arithmetic operators in Figure 3 are required to operate on scalar data types.[5]

plus	minus	times	divide	pow	mod	max	min
uminus	abs	log	ln	cos	sin	tan	acos
asin	atan						

Figure 3: Arithmetic Operators

Binary operators such as *plus*, *minus* and *times* have signature: $Scalar \times Scalar \rightarrow Scalar$, while a unary operators like *uminus*, *abs* and *log* have signature: $Scalar \rightarrow Scalar$. Arithmetic expressions are constructed using these operators.

A boolean type can have *False* or *True* constant values. The boolean operators associated with this type are *and*, *or* and *not*. The *and* and *or* operators have signature: $Boolean \times Boolean \rightarrow Boolean$. The *not* operator's signature is: $Boolean \rightarrow Boolean$.

and	or	not	less	more	equal

Figure 4: Logical Operators

We also need the relational operators *less*, *more* and *equal* that maps a pair of Scalars to Boolean. Logical expressions constructed using the boolean and relational operators listed in Figure 4 will be used to specify conditions and constraints involving the quantifiable terms in the ontology.

4.2.2 Composite Data Type

The composite data types are built upon other data types. We have identified the need for two kinds of composite data types in the ontology, namely record and set data types. A record is a cartesian product of its constructed element types, while a set is a mathematical abstraction of a collection of elements.

Record

A record type is composed of a fixed number of data types. The set of values represented by a record is a cartesian product of its data types. A record is analogous to a class construct defined in ontology specification language.

[5]Other scalar types such as Rational and Complex would be included, when their need arise. Likewise, the existing set of operators for will be expanded accordingly.

class Track : RecordQuantity	
Property	**Range Value**
mom-x	Momentum-X
mom-y	Momentum-Y
mom-z	Momentum-Z
energy	Energy
class Momentum-X : ConstantQuantity subClassOf Momentum	
class Momentum-Y : ConstantQuantity subClassOf Momentum	
class Momentum-Z : ConstantQuantity subClassOf Momentum	
class Energy : ConstantQuantity subClassOf EventVariable	

Figure 5: Partial Definition of Track and its Range Relations

We utilise record data type to model a structured set of values associated with an aggregate of disparate quantities such as *Track* (Figure 5). Track information is typically denoted by an n-tuple of distinct quantities, partially described as $< P_x, P_y, P_z, E >$. We choose to refer to such grouped quantities in ontology as record quantity (actually a record of quantities) because we are able to provide explicit names for the individual quantities in the structure, rather than simply rely upon the ordinal of the quantities in an n-tuple.

An access operator called *select* is required to select individual elements of a record. The property names serve as the identifier of a record element. For example, *select(T, mom-x)* accesses the *mom-x* component of track *T*.

Set

A set is a data type representing a collection of things. In the context of our work, we restrict the use of set to represent a collection of individuals with common characteristics, that is class instances of same type. Set operators are specifically defined to be applied on an entire collection of individuals. The operators named in Figure 6 are required for constructing a set, determining its size, checking set membership, telling the maximum and minimum individual, and summing up all the individuals in a set.

setOf	union	intersect	oproduct	filter
size	member	maximum	minimum	summation

Figure 6: Set Operators

The operators on the first row are set constructors. Notably, the *oproduct* operator constructs the outer product of a pair of sets. For example, the Fox-Wolfram Moment-0 event-variable $H_0 = \sum_j \sum_k (P_{3_j} \times P_{3_k})$ can be expressed as follows: *summation(oproduct(setOf(P_3), setOf(P_3)))*. The summation operator repeatedly applies the plus operation on all the individuals in the constructed set of outer product of 3-Momentum.

The filter operator applies a condition (logical expression) on a set to filter out a subset of desired individuals, that is individuals that fulfil the specified condition. As an example, the following expression sums up the transverse mo-

class FunctionQuantity		
subClassOf	PhysicalQuantity	
Property	**Range Relation**	**Cardinality**
parameter	Parameter	≥ 1
intension	ArithmeticLogicalExpression	
magnitude	Scalar	$= 1$
dimension	DimensionUnit	$= 1$

Figure 7: Definition of FunctionQuantity

class Parameter		
Property	**Range Relation**	**Cardinality**
argument	ConstantQuantity	≥ 1
coefficient	Scalar	≤ 1

Figure 8: Definition of Parameter

mentum of tracks (P_T) that makes an angle of more than 45° with the Thrust axis ($\hat{T}$): *summation(filter(setOf(P_T), and(more(angleBetween(x, $\hat{T}$), $\pi/4$), member(x, P_T)))*. The function *angleBetween* is a geometric event-variable defined in the ontology. It represents the plane angle between any two axes in a particular frame of reference.

4.3 Mathematical Relations as Function Quantities

Mathematical relations in the ontology can be described abstractly in the form of grounded function quantities. A function quantity represents a physical quantity that is arithmetically derived from previously defined quantities. It is invoked with an explicit set of constant quantity parameters, which map to a constant quantity. In other words, a function quantity describes the numerical dependencies between its parameters, and the resulting quantity. Consequently, the intension of a function quantity is specified by an arithmetic-logical expression involving its parameters.

We give the definition of function quantity in Figure 7. It is conceptualised as a subclass of *PhysicalQuantity* with a set of properties, namely *parameter*, *intension*, *magnitude* and *dimension*. A function quantity inherits the *magnitude* and *dimension* properties from *PhysicalQuantity*, which together denote the characteristics of the resulting constant quantity.

4.3.1 Parameter of Function Quantity

Parameters identify the types of objects that are involved in a mathematical relation described by a function quantity. These parameters are constant quantities, which includes existing grounded function quantities and record quantities. Recall that a grounded function quantity delivers a constant quantity as result. A record quantity constitutes a structured set of disparate constant quantities.

Each parameter of a function quantity is either an individual quantity or a collection of quantities. Alternatively, a weight is attached to a parameter that indicates its degree of influence on the resulting value. The metaclass definition in Figure 8 is an abstraction of the various types of parameters of a function quantity, whose *argument* and *coefficient* are constrained to *ConstantQuantity* and *Scalar* type, respectively.

class TransverseMomentum : FunctionQuantity	
subClassOf	Momentum
Property	**Range Value**
parameter	P_x, P_y
intension	*abs(pow(sum(pow(P_x,2),pow(P_y,2))),0.5)*
magnitude	Real
dimension	MomentumUnit

class P_x : IndividualParameter	
Property	**Range Value**
argument	Momentum-X

class P_y : IndividualParameter	
Property	**Range Value**
argument	Momentum-Y

Figure 9: Partial Definition of Transverse Momentum and its Range Relations

Individual Parameter

An individual parameter is a subclass of *Parameter* whose *argument* cardinality is equal to 1. It denotes a specific quantity argument of a function quantity. Scalar and relational operators can be applied on the magnitude of this argument.

Figure 9 illustrates the definition of Transverse Momentum function quantity with two individual parameters. An instantiated *TransverseMomentum* maps instances of *Momentum-X* and *Momentum-Y* to an instance of momentum quantity, according to the specified intension.

Collection Parameter

A collection parameter is a subclass of *Parameter* whose *argument* cardinality is greater than 1. This type of argument is viewed as a collection of homogenous quantities and is typically applied to set operators as a whole. A case in point is the collection parameter (a set of 3-Momentum) of the Fox-Wolfram Moment-0 function quantity.

Weighted Parameter

A weighted parameter is a subclass of *Parameter* whose *coefficient* has a definite value, as in the case of Fisher Discriminant function quantity. This value will be dealt within the arithmetic-logical expression that specifies the intension of this function quantity.

4.3.2 Intension of Function Quantity

An arithmetic-logical expression (a constrained sequence of symbols) conveys the intension of a function quantity. The semantics is determined largely by the arithmetic, logical and set operations applied on the parameters of the function quantity. One possible way to encode this expression is using OpenMath [19].[6] Figure 10 shows the OpenMath encoding of the arithmetic-logical expression that describes the intension of TransverseMomentum (see definition in Figure 9).

[6]Another W3C data exchange standard is MathML [18]. While OpenMath focuses on the semantics in mathematical expression, MathML gives importance to the presentation (rendering) of the mathematical expression.

```
<OMOBJ>
  <OMA>
    <OMS cd="arith1" name="abs"/>
    <OMA>
      <OMS cd="arith1" name="pow"/>
      <OMA>
        <OMS cd="arith1" name="plus"/>
        <OMA>
          <OMS cd="arith1" name="pow"/>
          <OMV name="P_x"/>
          <OMI> 2 </OMI>
        < /OMA>
        <OMA>
          <OMS cd="arith1" name="pow"/>
          <OMV name="P_y"/>
          <OMI> 2 </OMI>
        < /OMA>
        < /OMF dec="0.5"/>
      < /OMA>
    < /OMA>
  < /OMA>
< /OMOBJ>
```

Figure 10: Intension of TransverseMomentum Encoded in OpenMath

Expressions encoded in OpenMath are constrained by XML-syntax with implied semantics. An OpenMath object (OMOBJ) is a sequence of one or more application objects (OMA). OpenMath maintains reportative definitions of the mathematical-oriented metadata in a set of content dictionaries. In this example, the content dictionary (cd) named *arith1* holds the definition of the following symbols (OMS): *abs*, *pow* and *plus*. These symbols coincide with the data type operators identified in Section 4.2. The parameter or variable (OMV) is associated with constant quantity, in our case.

We appeal to an interpreter for manipulating this encoded expression. OpenMath only recognises scalar values such as integer (OMI) and real (OMF). The design decision will also have to consider on how to extend the scalar and boolean algebra to cover the constant quantities in the ontology. For instance, the interpreter could apply the operators on the scalar magnitude of dimensionally compatible quantities whose units have been reconciled. We do not wish to dwell further into this implementation issue at this stage.

4.3.3 Result of Function Quantity

The result of a grounded function quantity is a constant quantity, whose *magnitude* and *dimension* are same as that of the function quantity. Even a pure scalar result can be viewed as a constant quantity whose magnitude is scalar and its physical dimension is termed as 'dimensionless'. The *dimension* of a function quantity is only defined generically. An instance of a function quantity will however use one of the prescribed units associated with its dimension type.

4.4 Dynamic Mathematical Relations in Ontologies

We want to deal with both static and dynamic mathematical relations in the ontology. A static relation is a permanent relationship that exist between conceptual terms in all instantiated model (read 'experimental analysis description') of the EHEP domain. An example is the relation describing Transverse Momentum function quantity, which is always composed from Momentum-X and Momentum-Y constant quantities. In other words, the parameters of function quantity denoting a static relation are fixed.

On the other hand, a dynamic relationship between concepts is coerced specifically for a particular use or purpose. An example is the relation described by Fisher Discriminant function quantity, which combines a set of correlated event-variables. Different event selection analysis entails different sets of correlated event-variables. Therefore, the set of event-variables combined by Fisher Discriminant varies from one instantiated model to another. Although the intension of this function quantity is fixed, its parameters are not.

Function quantities to find the difference or the sum of any two quantities also fall under the category of dynamic mathematical relations. In here, the number of parameters is fixed, but their types can vary. For instance, the quantity sum of two mass quantities is not same as the quantity sum of two momentum quantities, as they have dissimilar dimensions.

This paper mainly discusses on the representation of the static relations. We think it is possible to extend the existing framework to handle dynamic relations in the ontology. An alternative is to define a dynamic mathematical relation as an 'abstract' function quantity, still ungrounded in the ontology. The intension of this function quantity will be described in terms of physical quantities (that is, the type of the argument is *PhysicalQuantity*, instead of *ConstantQuantity*). Subsequently, a specialised form of this function quantity can be derived from its abstract definition, to suit a specific need. The specialised function quantity will assign appropriate interpretation of the mathematical relation by specifying the exact parameters and characteristics of the resulting quantity. This idea is still under consolidation.

5. CONCLUSION

This EHEP ontology should provide the necessary representational vocabulary to facilitate the scientific community to collaborate effectively on the semantic web. One concern however is due to the limitations of web-ontology languages, which disallow the direct representation of mathematical relations in ontologies. The existing web-ontology languages falter when the need to specify metaclasses, n-ary relations and functions arises. They also lack the necessary epistemological and arithmetic primitives to explicitly represent algebraic expressions involving domain terms.

This paper highlights the additional vocabulary and language features required to deal with mathematical relations in web-ontology. Future web-ontology languages may offer a richer set of primitives to express complex relationships among entities. Until such time, our approach recommends the use of implicitly defined metadata to partly describe the meaning of the mathematical relations in web-ontology.

6. ACKNOWLEDGMENTS

We are grateful to Glenn Moloney and Lyle Winton of the Physics department for providing insights into EHEP exper-

imental analysis. We also thank the members of the Intelligent Agent Laboratory at the University of Melbourne for their helpful comments.

7. REFERENCES

[1] M. Annamalai, L. Sterling, and G. Moloney. A collaborative framework for distributed scientific groups. In S. Cranefield, S. Willmott, and T. Finin, editors, *Proceedings of AAMAS'02 Workshop on Ontologies in Agent Systems*, Bologna, Italy, 2002.

[2] F. Baadar and W. Nutt. Basic description logics. In F. Baadar, D. McGuiness, D. Nardi, and P. Patel-Schneider, editors, *Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2002.

[3] The Babar Physics Collaboration. http://www.slac.stanford.edu/BFROOT/.

[4] The Belle Physics Collaboration. http://belle.kek.jp/belle/.

[5] B. Chandrasekaran and J. R. Josephson. What are ontologies, and why do we need them? *IEEE Intelligent Systems*, pages 20–26, January/February 1999.

[6] J. C. Cleaveland. *An Introduction to Data Types*. Addison-Wesley, 1986.

[7] The Cleo Physics Collaboration. http://www.lns.cornell.edu/public/CLEO/.

[8] L. Cruz, M. Annamalai, and L. Sterling. Analysing high-energy physics experiments. In B. Burg, J. Dale, T. Finin, H. Nakashima, L. Padgham, C. Sierra, and S. Willmott, editors, *Proceedings of AAMAS'02 Workshop on AgentCities*, Bologna, Italy, 2002.

[9] Cycorp. http://www.cyc.com/.

[10] D. Fensel, I. Horrocks, F. vanHarmelen, D. McGuiness, and P. Patel-Schneider. Oil: Ontology infrastructure to enable the semantic web. *IEEE Intelligent Systems*, pages 38–45, March/April 2001.

[11] M. Genesereth and R. Fikes. Knowledge interchange format. Technical Report Logic-92-1, Computer Science Department, Stanford University, 1992.

[12] W. E. Grosso, H. Eriksson, R. W. Fergerson, J. H. Gennari, S. W. Tu, and M. A. Musen. Knowledge modeling at the millennium (the design and evolution of Protege-2000). In *Proceedings of KAW'99 Workshop on Knowledge Acquisition, Modelling and Management*, Banff, Alberta, 1999.

[13] T. Gruber. A translation approach to portable ontologies. *Knowledge Acquisition*, 5(2):199–220, 1993.

[14] T. R. Gruber and G. R. Olsen. An ontology for engineering mathematics. In J. Doyle, P. Torasso, and E. Sandewall, editors, *Proceedings of International Conference on Principles of Knowledge Representation and Reasoning*. Morgan Kaufmann, 1994.

[15] M. Gruninger and M. S. Fox. Methodology for the design and evaluation of ontologies. In *Proceedings of IJCAI'95 Workshop Basic Ontological Issues in Knowledge Sharing*. Montreal, Canada, 1995.

[16] N. Guarino. Ontologies and knowledge base: Towards a terminological clarification. In N. Mars, editor, *Towards Very Large Knowledge: Knowledge Building and Knowledge Sharing*, pages 25–32. IOS Press, Amsterdam, 1995.

[17] J. Hendler and D. McGuiness. The Darpa Agent Mark up Language. *IEEE Intelligent Systems*, pages 67–73, November/December 2000.

[18] Mathematics Mark up Language. http://www.w3.org/TR/MathML2/.

[19] Openmath Mark up Language. http://monet.nag.co.uk/cocoon/openmath/index.html.

[20] Suggested Upper Merged Ontology. http://ontology.teknowledge.com/.

[21] Victorian Partnership for Advanced Computing. http://www.vpac.org/.

[22] EXtensible Mark up Language. http://www.w3.org/XML/.

Using OWL in a Pervasive Computing Broker*

Harry Chen
University of Maryland
Baltimore County, USA
hchen4@cs.umbc.edu

Tim Finin
University of Maryland
Baltimore County, USA
finin@cs.umbc.edu

Anupam Joshi
University of Maryland
Baltimore County, USA
joshi@cs.umbc.edu

ABSTRACT

Computing is moving toward a pervasive context-aware environment in which agents with limited resources will require external support to help them become context-aware. In this paper, we describe an agent based architecture called Context Broker Architecture (CoBrA) to help these agents to acquire, reason about and share context knowledge. A key component in our architecture is an explicit context ontology defined using the Web Ontology Language (OWL). This ontology models the basic concepts of people, agents, places, and presentation events. We also describe a use case scenario and prototype design of CoBrA in an intelligent meeting room environment.

1. INTRODUCTION

Pervasive computing is a vision for the near term future in which devices, services, and software agents will seamlessly integrate and cooperate in support of human objectives – anticipating our needs, negotiating for services, acting on our behalf, and delivering services in an anywhere, any-time fashion. To realize this vision, an important next step is the development of an infrastructure that can help ubiquitous agents, services, and devices become aware of their contexts (including the ability to reason about and to share this knowledge). We are developing a new pervasive context-aware computing infrastructure called Context Broker Architecture (CoBrA) [3] that will help agents behave in an intelligent, context-aware manner.

By context, we mean an understanding of a location, its environmental attributes (e.g., room temperature, noise level, and light intensity), and the people, devices, objects, and software agents it contains. This understanding necessarily extends to modeling the activities and tasks that are taking place in a location as well as the beliefs, desires, commitments, and intentions of the human and software agents involved.

In the past, a number of distributed systems have been developed with a common goal to support pervasive computing (e.g., Intelligent Room [5], Context Toolkit [16], and Cooltown [12]). Although the research of these systems have made tremendous progress in advancing pervasive computing, the resulting implementations, however, have weaknesses in supporting knowledge sharing and context reasoning due to lacking an explicit representation of context ontologies [3, 4].

An explicit representation of ontology consists of a formal model of vocabularies (i.e., classes and properties) and associated semantics (i.e., the relationships between different classes and properties). Without a well-defined ontology, knowledge sharing and context reasoning is often difficult in the previous systems [3].

In the previous systems, ontologies are often defined based on *ad hoc* representation schemes such as a set of programming language objects or data structures. There are two problems with this approach: (i) the use of *ad hoc* representation schemes lacking shared vocabularies can hinder the ability of independently developed agents to interoperate (i.e., to share context knowledge), and (ii) the use of objects and data structures of low expressive power provides inadequate support for context reasoning.

In order to help agents to discover, reason about and communicate contextual information, we must define explicit ontologies for context concepts and knowledge. In this paper, we present a set of ontologies that we have developed to support ubiquitous agents in the CoBrA system. Our ontology (CoBrA ontology) is defined using the Web Ontology Language (OWL) [18], an Semantic Web language being specified by the W3C. The current version of the CoBrA ontology models the basic concepts of people, agents, places and presentation events. It also describes the properties and relationships between these basic concepts including (i) relationships between places, (ii) roles associated with people in presentation events, and (iii) typical intentions and desires of speakers and audience members.

The rest of this document is structured into nine sections. In the next section, we briefly review the Semantic Web and the OWL language. In Section 3, we describe the key features of CoBrA and its design rationale. In Section 4, we present a typical use case scenario of the CoBrA system and point out how CoBrA can help agents to reason about contexts and share knowledge. After our discussion, in Section 5, we describe the key ontology concepts in the latest version of our CoBrA ontology. In Section 6, a prototype system design of the Context Broker Architecture is presented. A brief discussion of related work and some concluding comments are given in Section 7 and Section 8, respectively.

2. THE SEMANTIC WEB AND OWL

Semantic Web is a vision of the next generation World Wide Web in which information is given well-defined meaning, better enabling computers and people to work in cooperation [2]. Research on the Semantic Web is driven by the

*This work was partially supported by DARPA contract F30602-97-1-0215, Hewlett Packard, NSF award 9875433, and NSF award 0209001.

need for a new knowledge representation framework to cope with the explosion of unstructured digital information on the existing Web. Current Semantic Web research focuses on the development of ontology languages and tools for constructing digital information that can be "understood" by computers [2].

The origin of the Semantic Web research goes deep in the roots of Artificial Intelligent research (e.g., knowledge representation and ontology). However, the recent publicly known Semantic Web research begins with the DAML (DARPA Agent Markup Language) effort in the US[1] and the OIL (Ontology Inference Layer) effort in Europe[2]. In late 2000, the original DAML language is combined with many of the ontology modeling features from the OIL language, and the result is the DAML+OIL language.

In late 2001, the World Wide Web Consortium (W3C) established the Web Ontology Working Group with the goal of introducing Semantic Web technologies to the main stream web community. The group has specified a language OWL that is based on DAML+OIL and shares many of its features (e.g., using RDF as the modeling language to define ontological vocabularies and using XML as the surface syntax for representing information [18]).

We have chosen to define our ontology for contexts for three reasons. First, it is much more expressive than RDF or RDF-S allowing us to build more knowledge into the ontology. Second, we chose to use OWL over DAML+OIL because OWL has been designed as a standard and has the backing of a well known and regarded standards organization. Third, from a system implementation point of view, the emergence of ontology inference engines in support for the OWL language (e.g., FaCT [9], RACER [19] and Bubo [20]) leads us to believe adopting OWL will create new opportunities for building more advanced intelligent systems.

3. CONTEXT BROKER ARCHITECTURE

CoBrA is an agent based architecture for supporting context-aware computing in intelligent spaces. Intelligent spaces are physical spaces (e.g., living rooms, vehicles, corporate offices and meeting rooms) that are populated with intelligent systems that provide pervasive computing services to users [11].

Central to our architecture is the presence of an intelligent *context broker* (or broker for short) that maintains and manages a shared contextual model on the behalf of a community of agents. These agents can be applications hosted by mobile devices that a user carries or wears (e.g., cell phones, PDAs and headphones), services that are provided by devices in a room (e.g., projector service, light controller and room temperature controller) and web services that provide a web presence for people, places and things in the physical world (e.g., services keeping track of people's and objects' whereabouts [12]).

In a large-scale intelligent space (e.g., a campus or a building), multiple brokers can form a *broker federation*. Individual broker in a federation is responsible for managing parts of the intelligent space (e.g., a room in a particular building). Brokers are related to each other in some organizational structure (e.g., peer-to-peer or hierarchical) in a federation, and they can periodically exchange and synchronize context knowledge, enabling fault-tolerance (similar to the *persistent broker team* described in [13]).

[1]DAML web site:`http://www.daml.org`

[2]OIL web site: `http://www.ontoknowledge.org/oil/`

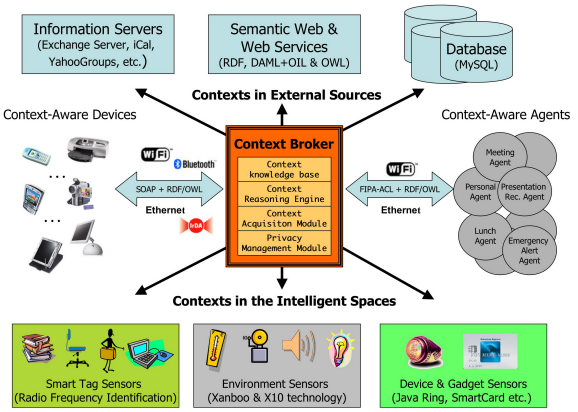

Figure 1: An intelligent context broker acquires context information from devices, agents and sensors in its environment and fuses it into a coherent model that is then shared with the devices and their agents.

In an intelligent space, the primary responsibilities of a broker are to (i) acquire context information from heterogeneous sources and reason about this information to maintain a consistent context model, (ii) help distributed agents to share context knowledge through the use of common ontologies, agent communication languages and protocols, and (iii) protect the privacy of users by establishing and enforcing user-defined policies while sharing sensitive personal information with agents in the community.

A context broker has four main functional components:

1. **Context Knowledge Base**: a persistent store for context knowledge in an intelligent space. This knowledge base provides a set of API's for other components to assert, delete, modify, and query stored knowledge.

2. **Context Reasoning Engine**: a reactive inference engine that reasons over the knowledge base. Its main function is to deduce additional knowledge from information acquired from external sources and to maintain the consistency of the knowledge base.

3. **Context Acquisition Module**: a collection of predefined procedures for acquiring information from the external sources. It serves as a middleware abstraction for acquiring contexts from heterogeneous sources (e.g., physical sensors, web services, databases, devices and agents).

4. **Privacy Management Module**: a set of communication protocols and behavior rules that the broker follows when performing privacy management tasks (i.e., negotiate privacy policies with new users and enforcing these policies when sharing information with agents in the community).

4. ONTOLOGIES IN COBRA

In a pervasive computing environment, individual agents may have limited resources to acquire, reason about and share contexts. In CoBrA, the role of a broker is to help

these resource-limited agents – e.g., to reason about contexts and share context knowledge.

In the next two sections, we will describe a typical multi-agent scenario of CoBrA (Section 4.1) and point out how explicitly represented ontologies enable knowledge sharing and context reasoning (Section 4.2).

4.1 An Intelligent Meeting Room Scenario

R210 is an intelligent meeting room with RFID sensors[3] embedded in the walls and furniture for detecting the presence of the users' devices and clothing. As Alice enters the room, these sensors inform the R210 broker that a cell phone belonging to her is present and the broker adds this fact in its knowledge base.

As she sits, the agent on Alice's Bluetooth enabled cell phone discovers R210's broker and engages in a "hand shake" protocol (e.g. authenticates agent identities and establishes trust [10]) after which it informs the broker of Alice's privacy policy. This policy represents Alice's desires about what the broker should do and includes (i) the context information about Alice that the broker is permitted or prohibited from storing and using (e.g., yes to her location and roles, no to the phone numbers she calls), (ii) other agents that the broker should inform about changes in her contextual information (e.g., keeping Alice's personal agent at home informed about her location contexts), and (iii) the permissions for other agents to access Alice's context information (e.g., all agents in the meeting room can access Alice's contexts while she is in the room).

After receiving Alice's privacy policy, the broker creates a profile for Alice that defines rules and constraints the broker will follow when handling any context knowledge related to Alice. For example, given the above policy, the profile for Alice would direct the broker (i) to acquire and reason about Alice's location and activity contexts, (ii) to inform Alice's personal agent at home when Alice's contexts change, and (iii) to share her contexts with agents in the meeting room.

Knowing Alice's cell phone is currently in R210 and having no evidence to the contrary, the broker concludes Alice is also there. Additionally, because R210 is a part of the Engineering building, which in turn is a part of the Campus, the broker concludes Alice is located in Engineering building and on campus. These conclusions are asserted into broker's knowledge base.

Following the profile, the broker informs Alice's personal agent of her whereabouts. On receiving this information about Alice, her personal agent attempts to determine why Alice is there. Her Outlook calendar has an entry indicating that she is to give a presentation on Campus about now, so the personal agent concludes that Alice is in R210 to give her talk and informs the R210 broker of it's belief.

On receiving information about Alice's intention, the R210 broker shares this information with the projector agent and the lighting control agent in the ECS 210. Few minutes later, the projector agent downloads the slides from Alice's personal agent and sets up the projector, the lighting control agent dims the room lights.

4.2 Knowledge Sharing & Context Reasoning

An explicit representation of ontologies can be used to enable knowledge sharing and provide a means for reasoning. In the scenario above, three distinct but related types of context information are used: (i) location contexts ("Where is Alice?"), (ii) activity context ("What is she doing?"), and (iii) agent contexts ("What might she want to do in this context?"). Note that an understanding of these contexts is only possible because of all different types of agents (including sensors and devices) share information with each other using common ontologies, and the broker is able derive additional information about Alice's location because it has a model of the rooms and buildings involved and has a rudimentary model of spatial relationships.

Context reasoning may also take place in detecting inconsistent beliefs about certain contexts. For example, in a pervasive computing environment, information sensed from the physical world can be both noisy and ambiguous (e.g., sensors may report that the same person is present in two different room at the same time). With ontologies, it is possible to guide the context reasoning process to detect and resolve this inconsistent knowledge (e.g., in our CoBrA ontology, containment relationships between different classes of locations can be used to detect this type of knowledge inconsistency).

5. THE COBRA ONTOLOGY

This section describes key ontology concepts in the current version of the CoBrA ontology (v0.2)[4]. This ontology defines a vocabulary for describing people, agents, places and presentation events for supporting an intelligent meeting room system on a university campus. It also defines a set of properties and relationships that are associated with these basic concepts.

Figure 2 shows a complete list of the names of the classes and properties in the CoBrA ontology. In v0.2, there are 41 classes (i.e., RDF resources that are type of `owl:class`) and 36 properties (i.e., RDF resources that are type of either `owl:ObjectProperty` or `owl:DatatypeProperty`). These ontologies are expressed using the OWL/XML syntax [17].

Our ontology is categorized into four distinctive but related themes: (i) concepts that define physical places and their associated spatial relations (e.g., containment, social and organizational properties)[5], (ii) concepts that define agents (i.e., both human agents and software agents) and their associated attributes, (iii) concepts that describe the location contexts of an agent on a university campus, and (iv) concepts that describe the activity contexts of an agent, including the roles of speakers and audiences and their associated desires and intentions in a presentation event. In the rest of this section, we discuss each of these four themes.

5.1 Places

The notion of a place in CoBrA is presently restricted to a set of physical locations that are typically found on a university campus. These locations include *campus*, *building*, *room*, *hallway*, *stairway*, *restroom*, and *parking lot*. These physical locations are all assumed have well-defined spatial boundaries (e.g., all locations can be uniquely identified by geographical coordinates – longitude and latitude). In addition, all locations on a university campus have identifiable

[3]RFID stands for Radio Frequency Identification (see also `http://www.rfid.org`)

[4]A complete version of the ontology is available at `http://daml.umbc.edu/ontologies/cobra/0.2/cobra-ont`

[5]In v0.2, only containment relations are defined, additional properties will be included in the next version of the ontology.

CoBrA Ontology Classes		CoBrA Ontology Properties	
"Place" Related	Agents' Location Context	"Place" Related	Agent's Location Context
Place AtomicPlace CompoundPlace Campus Building AtomicPlaceInBuilding AtomicPlaceNotInBuilding Room Hallway Stairway OtherPlaceInBuilding Restroom Gender LadiesRoom MensRoom ParkingLot	ThingInBuilding SoftwareAgentInBuilding PersonInBuilding ThingNotInBuilding SoftwareAgentNotInBuilding PersonNotInBuilding ThingInRoom SoftwareAgentInRoom PersonInRoom	latitude longitude hasPrettyName isSpatiallySubsumedBy spatiallySubsumes accessRestricted-ToGender lotNumber	locatedIn locatedInAtomicPlace locatedInRoom locatedInRestroom locatedInParkingLot locatedinCompoundPlace locatedInBuilding locatedInCampus
"Agent" Related	Agent's Activity Context	"Agent" Related	Agent's Activity Context
Agent Person SoftwareAgent Role SpeakerRole AudienceRole IntentionalAction ActionFoundInPresntation	PresentationSchedule EventHappeningNow PresentationHappeningNow RoomHasPresentationHappeningNow ParticipantOfPresentation-HappeningNow SpeakerOfPresentationHappeningNow AudienceOfPresentationHappeningNow PersonFillsRoleInPresentation PersonFillsSpeakerRole PersonFillsAudienceRole	hasContactInformation hasFullName hasEmail hasHomePage hasAgentAddress fillsRole isFilledBy intendsToPerform desiresSomeone-ToAchieve	participatesIn startTime endTime location hasEventHappeningNow invitedSpeaker expectedAudience presentationTitle presentationAbstract presentation eventDescription eventSchedule

Figure 2: A complete list of the names of the classes and properties in the CoBrA ontology (v0.2).

string names that are assigned to them by some official bodies (e.g., by the university administration).

When modeling physical locations, we define a class called `Place` which generalizes all type of locations on a campus. This abstract class defines a set of properties that are common to all concrete physical location classes, which consists of `longitude`, `latitude` and `hasPrettyName`.

`Place` classes (including subclasses) participate in containment relations. These relationships are defined by two related object properties[6] called `spatiallySubsumes` and `isSpatiallySubsumedBy`. The former describes the subject of this property spatially subsumes the object of this property (e.g., a building spatially subsumes a room in the building), and the latter describes the subject of this property is spatially subsumed by the object of this property (e.g., a room in the building is spatially subsumed by the building). In the context of the OWL language, these two properties are defined as an inverse property of each other.

Note that in the current version of the ontology, the domain and the range of both `spatiallySubsumes` and `isSpatiallySubsumedBy` properties are of the class type `Place`. In other word, these two properties cannot be used to make statements about the containment of a person, agent or object in a physical place. In Section 5.2, we will describe alternative constructs for expressing this type of statements.

In addition to containment relationships, physical places may also have events and activities associated with them (e.g., a meeting may be taken place in a room, or an annual festival may be taken place on a university campus).

[6]This refers to the `owl:ObjectProperty` property

In order to make statements about some events that are currently associated with a particular place, we introduce an additional object property called `hasEventHappening-Now`. The domain and range of this property are of the class `Place` and the class `EventHappeningNow`, respectively. The `EventHappeningNow` class represents a set of all events that are currently taking place (details of this class is discussed in Section 5.4).

```
<owl:Class rdf:ID="Place">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#AtomicPlace"/>
    <owl:Class rdf:about="#CompoundPlace"/>
  </owl:unionOf>
</owl:Class>

<owl:Class rdf:ID="AtomicPlace">
  <rdfs:subClassOf rdf:resource="#Place"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty
          rdf:resource="#isSpatiallySubsumedBy"/>
      <owl:allValuesFrom
          rdf:resource="#CompoundPlace"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty
          rdf:resource="#spatiallySubsumes"/>
      <owl:cardinality>0</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#AtomicPlaceInBuilding"/>
    <owl:Class
        rdf:about="#AtomicPlaceNotInBuilding"/>
  </owl:unionOf>
</owl:Class>

<owl:Class rdf:ID="CompoundPlace">
  <rdfs:subClassOf rdf:resource="#Place"/>
  <owl:disjointWith rdf:resource="#AtomicPlace"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty
          rdf:resource="#isSpatiallySubsumedBy"/>
      <owl:allValuesFrom
          rdf:resource="#CompoundPlace"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty
          rdf:resource="#spatiallySubsumes"/>
      <owl:allValuesFrom
          rdf:resource="#Place"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Campus"/>
    <owl:Class rdf:about="#Building"/>
  </owl:unionOf>
</owl:Class>
```

Figure 3: A partial ontology definition of the `AtomicPlace` & `CompoundPlace` classes in OWL/XML syntax

5.1.1 *AtomicPlace*

From the list of concrete physical locations that we have

mentioned (i.e., campus, building, room, hallway, stairway, etc.), some of these locations usually do not contain (spatially subsume) other physical locations. For example, hallways, stairways and rooms in a building usually are not usually considered to be a type of physical place that contains other places.

For this reason, we introduce an abstract class called `AtomicPlace` to represent a set of all physical places that do not contain other physical places. This class inherits all properties from its superclass `Place`. However, it puts restrictions on the range of the two properties `spatiallySubsumes` and `isSpatiallySubsumedBy`. In this `AtomicPlace` class, the cardinality of the property `spatiallySubsumes` is 0, indicating all instances of this class do not contain any other physical places. On the other hand, the range of the property `isSpatiallySubsumedBy` is restricted to the class `CompoundPlace`, which is a subclass of `Place`. This `CompoundPlace` class represents all physical places that may contain other physical places. Figure 3 shows partial representation of these classes in OWL/XML syntax.

Some subclasses of the `AtomicPlace` class include `Room`, `Hallway`, `Stairway`, `Restroom`, `LadiesRoom`, `MensRoom` and `ParkingLot`.

5.1.2 CompoundPlace

The `AtomicPlace` class represents a set of places that contains zero number of `Place` instances, the `CompoundPlace` class represent places that contain at least one `Place` instances. This class is also a subclass of `Place`.

Being a subclass of the `Place` class, `CompoundPlace` inherits all properties from its parent class. In order to express all instances of the `CompoundPlace` class should only be spatially subsumed by instances of other `CompoundPlace`, the range of this class's property `isSpatiallySubsumedBy` is restricted to have class type `CompoundPlace`. This restriction excludes all instances of the `CompoundPlace` class to be spatially subsumed by instances of the `AtomicPlace`.

5.2 Agents

The agent class in CoBrA represents both humans agents and software agents. Human agents are users in an intelligent space. Software agents, on the other hand, are autonomous computing entities that provide services to users (either directly or indirectly) in an associated space.

All agents have associated properties that describes their contact information, which includes uniquely identifiable names, URLs for their home pages, and email addresses. In addition, agents are assumed to have certain roles in different events and activities (e.g., a person can have the speaker role in a presentation event, and device agents in the close vicinity may take on the presentation assistant role during the presentation session). Different roles may give rise to different desires and intentions of an agent.

In the CoBrA ontology, the notions of desire and intention are both associated with actions[7]. Specifically, the notion of desire is defined as an agent's desire for some actions to be achieved by other agents (e.g., a person with the speaker role may desire some service agents to dim the lights when his presentation starts), and the notion of intention is defined as an agent's commitment to perform some particular actions (e.g., a person with the audience role may intend to download a copy of the slides after attending a presentation event).

To begin our ontology modeling for agents, we introduce a general class called `Agent`, which is a set of all human agents and computational agents. We define the class `Person` to represent human agents and the class `SoftwareAgent` to represent computational agents (both of which are subclasses of the `Agent` class and disjoints with each other). All agents in our ontology are associated with properties that describe their contact information. To generalize properties that serve as descriptions of contact information, we define an object property called `hasContactInformation`. From this property, we further define sub-properties of contact information, which consist of `hasFullName`, `hasEmail`, `hasHomePage` and `hasAgentAddress`.

5.2.1 Role

In our ontology, the class `Role` represents a set of all roles that can be associated with an agent. In other words, it is an abstract class that generalizes all possible types of agent roles in the CoBrA ontology. In v0.2 of the ontology, pre-defined subclasses of `Role` consist of `SpeakerRole` and `AudienceRole`.

To associate roles with an agent, the object properties `fillsRole` and `isFilledBy` are defined. In the context of the OWL language, these two properties are inverse property of each other – `fillsRole` has domain `Agent` and range `Role`, and `isFilledBy` has domain `Role` and range `Agent`.

```
<owl:Class rdf:ID="Role"/>

<owl:ObjectProperty rdf:ID="fillsRole">
  <rdfs:domain rdf:resource="#Agent"/>
  <rdfs:range rdf:resource="#Role"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="isFilledBy">
  <owl:inverseOf rdf:resource="#fillsRole"/>
</owl:ObjectProperty>

<owl:Class rdf:ID="SpeakerRole">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#RoleInPresentation"/>
    <owl:Restriction>
      <owl:onProperty
rdf:resource="#intendsToPerform"/>
      <owl:hasValue
rdf:resource="#GivePPTPresentation"/>
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>

<owl:Class rdf:ID="AudienceRole">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#RoleInPresentation"/>

    <owl:Restriction>
      <owl:onProperty
        rdf:resource="#intendsToPerform"/>
      <owl:hasValue
        rdf:resource="#AttendPresentation"/>
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```

```
<owl:Class rdf:ID="IntentionalAction"/>

<owl:Class rdf:ID="ActionFoundInPresentation">
  <rdfs:subClassOf rdf:resource="#IntentionalAction"/>

  <owl:oneOf rdf:parseType="Collection">
    <owl:Thing rdf:about="#GivePPTPresentation"/>
    <owl:Thing rdf:about="#SetupPPTPresentation"/>
    <owl:Thing rdf:about="#AdjustLightIntensity"/>
    <owl:Thing rdf:about="#DistributeHangouts"/>
    <owl:Thing rdf:about="#AttendPresentation"/>
    <owl:Thing rdf:about="#AcquireHangouts"/>
    <owl:Thing rdf:about="#ExchangeContact"/>
  </owl:oneOf>
</owl:Class>

<owl:ObjectProperty rdf:ID="intendsToPerform">
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
                <owl:Class rdf:about="#Role"/>
                <owl:Class rdf:about="#Agent"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdfs:range rdf:resource="#IntentionalAction"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="desiresSomeoneToAchieve">
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
                <owl:Class rdf:about="#Role"/>
                <owl:Class rdf:about="#Agent"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdfs:range rdf:resource="#IntentionalAction"/>
</owl:ObjectProperty>
```

Figure 4: This is a partial definition of the concepts related to roles, intentions and desires in an intelligent meeting room system.

5.2.2 Intentional Actions

All actions in CoBrA are defined as instances of the class `IntentionalAction`. Informally, intentional actions are actions that an agent performs intentionally and with certain goals in mind. In our design, we assume domain applications will extend this class to define specialized subclasses and instances. To support the construction of intelligent meeting room systems, we have pre-defined a set of concrete instances of `IntentionalAction` that are common in presentation events (see Figure 4).

All instances of the `IntentionalAction` class (or its sub-

[7]the semantics of an action is not formal defined in the current version of the ontology. In v0.2, all instances of actions are assumed to be atomic.

classes) can be associated with either an instance of the `Role` class or the `Agent` class through object properties `intendsToPerform` or `desiresSomeoneToAchieve`. The domain of these two properties are defined to be a union of the class `Role` and `Agent` (see Figure 4).

5.3 Agent's Location Context

In the last two sections, we have described a set of CoBrA ontology concepts for physical locations and agents. In this section, we will discuss additional concepts for modeling the location context of agents.

By location context, we mean a collection of dynamic knowledge that describes the location of an agent. In the context of the OWL language, this knowledge is a collection of RDF statements that describe the location property of an agent. To model the location property of an agent, we introduce an object property called `locatedIn`, which has range `Place`[8].

Physical locations, as we have discussed previously in Section 5.1, are categorized into two distinctive classes namely `AtomicPlace` (e.g., hallways and rooms) and `CompoundPlace` (e.g., campus and building). Following the semantics of these two classes, *no agent can locate in two different atomic places at the same time, but any agent can locate in two or more compound places at the same time.* In the context of the OWL language, any two instances of the `AtomicPlace` class are different if and only if both instances have distinctive object values[9] for the same class property.

To capture the notion an agent can be in an atomic and a compound place, from the `locatedIn` property we define two sub-properties called `locatedInAtomicPlace` and `locatedInCompoundPlace`. The former restricts its range to the `AtomicPlace` class, and the latter restricts its range to the `CompoundPlace` class. From these two properties, we can define additional properties that further restricts the type of physical place an agent is located in. For example, `locatedInRoom`, `locatedInRestroom` and `locatedInParkingLot` are sub-properties of `locatedInAtomicPlace`; `locatedInCampus` and `locatedInBuiding` are sub-properties of `locatedInCompoundPlace`.

Since all agents in CoBrA are associated with different types of location properties, we can generalize subsets of these agents in according to their location properties. To do so, we define `PersonInBuilding` and `SoftwareAgentInBuilding` to represent a set of people and software agents who are located in a building, respectively. The complement of these classes are `PersonNotInBuilding` and `SoftwareAgentNotInBuilding`.

5.4 Agent's Activity Context

An agent's activity context is similar to its location context – a collection of dynamic knowledge about certain aspects of an agent's situational condition. While location context describes the location at which the agent is situated, activity context describes activities in which the agent participates. In the current version of the ontology, the notion of an activity is restricted to represent a set of all typical group activity events in a meeting room (meeting, presentation and discussion)[10].

Activity events are assumed have schedules. For presentation events, we have defined `PresentationSchedule` class to represent their schedules. Presentation schedules are defined to have `startTime`, `endTime` and `location` properties, and each of which respectively represents the start time of a presentation, the end time of a presentation and the location of a presentation event. Each presentation event has one or more invited speaker and an audience. These two concepts are defined using the `invitedSpeaker` and `expectedAudience` properties. In addition to start time, end time and location, the schedule of a presentation usually includes a title and an abstract of the presentations. To model these two concepts, we introduce `presentationTitle` and `presentationAbstract` properties.

An agent's activity context is usually associated with activity events that are currently happening. For example, the activity context of a speaker includes the presentation event that he/she is giving the presentation at. To model this, we introduce the `PresentationEventHappeningNow` class. This class is a subclass of the `EventHappeningNow` class which models an event with the time predicate "now".

For a presentation that is currently happening, we can specialize the type of rooms at which the event takes place. For example, a room that has an ongoing presentation event is defined as `RoomHasPresentationEventHappeningNow`, which is a subclass of `Room` and restricts the range of its `hasEventHappeningNow` property to the class `PresentationSchedule`. In addition, we can also specialize people who has various roles in an on-going event. For example, a set of all people who have the speaker role of some on-going presentation event is defined as the `SpeakerOfPresentationHappeningNow` class. Similarly, we define the `AudienceOfPresentationHappeningNow` class to represent a set of all people who have the audience role of some on-going presentation event.

6. A PROTOTYPE SYSTEM DESIGN

In the last section, we have describe the key concepts in the CoBrA ontology. In this section, we present a prototype system design for realizing the intelligent meeting room scenario that we have described in Section 4.1.

In our prototype design (see Figure 5), there are five major system components: (i) a context broker, (ii) a personal agent of the user, (iii) a projector agent, (iv) a Bluetooth-enabled cell phone that the user carries, and (v) devices and clothes tagged with RFID tags. The context broker, the personal agent, and the projector agent will be implemented using the JADE agent development framework [1], a FIPA compliant Java agent development library. These agents will communicate with each other using the standard FIPA-ACL. When these agents communicate for the purpose of sharing context knowledge, the OWL language will be used as the content language for encoding context knowledge. Because all agents in CoBrA are assumed to share a common ontology (i.e., CoBrA ontology) for representing context knowledge, both the projector agent and the personal agent can interoperate with the context broker even if their internal implementations are independently designed

[8]The domain of this property is `owl:Thing`, indicating any thing may be located in some physical place.

[9]an object value refers to the object in an N-triple statement (i.e. (<subject>, <predicate>, <object>)

[10]In v0.2 of the ontology, we have only included concepts related to presentation events. In the future version, we will extend the ontology to includes other activity events

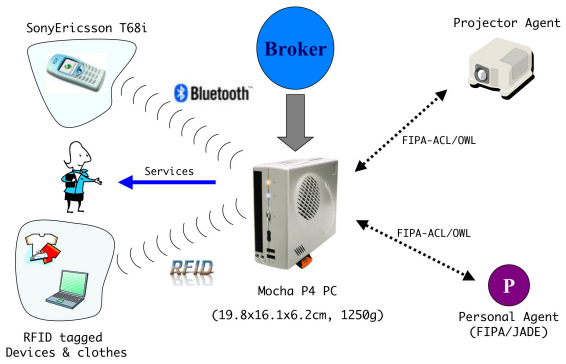

Figure 5: In our prototype system, the context broker will operate on a MOCHA PC, an all-in-one design mini book-size PC. It will be integrated with Bluetooth wireless communication and RFID readers for detecting people presence. The broker will communicate with software agents via FIPA-ACL/OWL.

(i.e., the interoperability of these agents are not completely pre-defined but rather achieved through ontology sharing).

In addition to the agents, our design also includes the use of a SonyEricsson T68i cell phone, which is component (iv). On this Bluetooth-enabled cell phone, users will store their personal policies (e.g., similar to the policy described in Section 4.1). As users enter the meeting room, they will submit their personal policies to the broker through Bluetooth networks. These policies will be expressed using the XML/OWL language syntax following the CoBrA ontology[11].

In order to detect the presence of users, we will label user devices and clothing with RFID tags, similar to the approach used in the CoolAgent RS [4] system. We are planning to deploy a number of RFID readers in our prototype environment, for example, placing readers next to the meeting room's door and underneath the tables in the meeting room.

7. RELATED WORK

Our work is closely related to other pervasive and context-aware computing research such as Intelligent Room, Context Toolkit, Cooltown, One.World [8] and Centaurus [11]. In comparison to the previous systems, our design of the Context Broker Architecture takes a knowledge representation approach to build ontologies of contexts and attempts to use Semantic Web language (i.e., OWL) as the content language in agent communication.

An explicit representation of context ontologies distinguishes CoBrA from other context-aware systems. While the previous systems rely on *ad hoc* representations of contexts to exchange information, CoBrA takes a knowledge representation approach allowing context knowledge to be reasoned and shared through a well-defined ontology model.

As in other systems (e.g., Context Toolkit and Cooltown), the CoBrA ontology models concepts for describing user locations. These concepts are useful for guiding the decision making of context-aware applications [16, 5, 11]. Nevertheless, none of the previous systems have explored the space and spatial relationship aspects of the location contexts (i.e., information that describes the whole physical space that surrounds a particular location and its relationship to other locations). Modeling space and spatial relationships are important in CoBrA. We currently have a simple model of space and spatial relationships (see Section 5.1). In the DAML+OIL community, recent discussions on the daml-spatial mailing list have initiated the work to develop a Semantic Web version of the spatial ontology based the SUO [14] and Cyc [6]. In the future, we plan on using, if possible, or at least mapping to, if feasible, one of these consensus ontologies for space.

In addition to using OWL as an ontology language for modeling contexts, we also attempt to use OWL as the content language in agent communication. Our approach is similar to the use of OWL in the TAGA system (Travel Agent Game in Agentcities) [22]. In TAGA, collections of agent communication primitives (e.g., action, result, query, sender, and receiver) are defined using OWL, forming ontologies for agent communications[12]. Using these ontologies, agents can express their reasons for communicating with other agents (i.e., making propositions and querying for information).

Using OWL as the content language in agent communication will allow the underlying agent implementations to be better integrated with Semantic Web technologies (e.g., ontology inference engines and Semantic Web query languages [7]). In contrast to the use of other content languages such FIPA-SL, KIF and XML, the use of OWL as a content language helps to simplify the underlying implementations for composing communication messages (e.g., avoiding multi-level parsing implementations for translating contents into internal knowledge representation).

8. CONCLUSION AND FUTURE WORK

Computing is moving towards pervasive, context-aware, environments in which resource-limited agents will require external supports to help them to become aware of their contexts. The Context Broker Architecture described in this paper will help these agents to acquire, reason, and share contextual knowledge. A key component of this infrastructure is an explicit representation of context ontologies expressed in the OWL language. Without this ontology, inconsistent and ambiguous context knowledge cannot be easily detected and resolved, and acquired context knowledge cannot be shared with independently developed agents.

We are developing an OWL reasoning engine called F-OWL[13] to support the use case described in Section 4.1. This reasoning engine is implemented using Flora-2 in XSB [15], an object-oriented knowledge base language and application development platform that translates a unified language of F-logic, HiLog, and Transaction Logic into the XSB deductive engine [21].

We plan to prototype an intelligent context broker and integrate this broker with the Centaurus systems (a framework for building pervasive computing services developed

[11]In the next version of the CoBrA ontology (v0.3), we will use concepts from the policy language REI [10]

[12]FIPA OWL content language ontology is available at `http://taga.umbc.edu/taga2/owl/fipaowl.owl`

[13]F-OWL web site: `http://umbc.edu/~hchen4/fowl/`

at UMBC) [11]. The objective is to create a pervasive context-aware meeting room in the newly constructed Information Technology and Engineering Building on UMBC's main campus[14].

9. REFERENCES

[1] F. Bellifemine, A. Poggi, and G. Rimassa. Developing multi agent systems with a fipa-compliant agent framework. *Software - Practice And Experience*, 31(2):103–128, 2001.

[2] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, may 2001.

[3] H. Chen. An intelligent broker architecture for context-aware systems. PhD. dissertation proposal. `http://umbc.edu/~hchen4/`, 2003.

[4] H. Chen, S. Tolia, C. Sayers, T. Finin, and A. Joshi. Creating context-aware software agents. In *Proceedings of the First GSFC/JPL Workshop on Radical Agent Concepts*, 2001.

[5] M. H. Coen. Design principles for intelligent environments. In *AAAI/IAAI*, pages 547–554, 1998.

[6] Cyccorp Inc. *The Upper Cyc Ontology*, 1997. `http://www.cyc.com/cyc-2-1/cover.htm`.

[7] R. Fikes, P. Hayes, and I. Horrocks. *DAML Query Language (DQL)*, 2002.

[8] R. Grimm, T. Anderson, B. Bershad, and D. Wetherall. A system architecture for pervasive computing. In *Proceedings of the 9th ACM SIGOPS European Workshop*, pages 177–182, 2000.

[9] I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In H. Ganzinger, D. McAllester, and A. Voronkov, editors, *Proceedings of the 6th International Conference on Logic for Programming and Automated Reasoning (LPAR'99)*, number 1705 in Lecture Notes in Artificial Intelligence, pages 161–180. Springer-Verlag, 1999.

[10] L. Kagal, T. Finin, and A. Joshi. A policy language for a pervasive computing environment. In *IEEE 4th International Workshop on Policies for Distributed Systems and Networks*, 2003.

[11] L. Kagal, V. Korolev, H. Chen, A. Joshi, and T. Finin. Centaurus : A framework for intelligent services in a mobile environment. In *Proceedings of the International Workshop on Smart Appliances and Wearable Computing (IWSAWC)*, 2001.

[12] T. Kindberg and J. Barton. A Web-based nomadic computing system. *Computer Networks (Amsterdam, Netherlands: 1999)*, 35(4):443–456, 2001.

[13] S. Kumar, P. R. Cohen, and H. J. Levesque. The adaptive agent architecture: Achieving fault-tolerance using persistent broker teams. In *Proceedings of the Fourth International Conference on Multi-Agent Systems*, pages 159–166, 2000.

[14] I. Niles and A. Pease. Towards a standard upper ontology. In *Proceedings of the 2nd International Conference on Formal Ontology in Information Systems (FOIS-2001)*, 2001.

[15] K. Sagonas, T. Swift, D. S. Warren, J. Freire, P. Rao, B. Cui, and E. Johnson. *The XSB Programmers' Manual*, 2002.

[16] D. Salber, A. K. Dey, and G. D. Abowd. The context toolkit: Aiding the development of context-enabled applications. In *CHI*, pages 434–441, 1999.

[17] M. K. Smith, C. Welty, and D. McGuinness. Owl web ontology language guide. `http://www.w3.org/TR/owl-guide/`, 2003.

[18] F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. Owl web ontology language reference. `http://www.w3.org/TR/owl-ref/)`, 2002.

[19] R. M. Volker Haarslev. Description of the racer system and its applications. In *Proceedubgs International Workshop on Description Logics (DL-2001)*, 2001.

[20] R. Volz, S. Decker, and D. Oberle. Bubo - implementing owl in rule-based systems. 2003.

[21] G. Yang and M. Kifer. *Flora-2: User's Manual*. Department of Computer Science, Stony Brook University, Stony Brook, 2002.

[22] Y. Zou, T. Finin, L. Ding, H. Chen, and R. Pan. Taga: Trading agent competition in agentcities. In *IJCAI-2003 Trading Agent Conmpetition Workshop*, 2003.

[14]ITE building construction live feed: `http://www.cs.umbc.edu/ITE/ITE.html`

Experiences with Ontology Development for Value-Added Publishing

Maia Hristozova
Department of Computer Science and Software Engineering,
The University of Melbourne,
Australia
majah@cs.mu.oz.au

Leon Sterling
Department of Computer Science and Software Engineering,
The University of Melbourne,
Australia
leon@cs.mu.oz.au

ABSTRACT

This paper presents our practical experience of developing an ontology using the EXPLODE method for Value-Added Publishing. Value-Added Publishing is a relatively new area of electronic information distribution that extends beyond the established simple modes of online publishing. VAP considers the needs of the user as a primary measure of effectiveness of an online document by evaluating various metrics such as author reputation and number of citations. At implementation level the features of VAP can be achieved by customised information extraction agents. The domain of VAP raises many challenges. VAP is being developed by different groups of researchers, the requirements for the ontology come from many different places, not necessarily consistent with each other. Since VAP is still not a strongly established field, its borders and the issues that it addresses often change. EXPLODE is a method which is suitable for a dynamic environment. In this paper we give an overview of EXPLODE and describe the issues that we encountered and the decisions we had to make regarding the ontology design, structure and content.

1. Introduction

Since 1997 ontology-based information extraction agents have advanced significantly, maturing in width and breadth of capabilities [6, 24, 25]. No longer is the state of the art a lone information agent treading a one-dimensional path through a field of data. Growing interest in multi-agent systems is providing a platform for realizing much more sophisticated outcomes through the interactions of numbers of information agents, as the outputs of individual agents can now be combined and manipulated. The grouping of multiple agents into a single system implies interaction between the individuals, in effect the construction of a community of software programs. It is widely acknowledged [15, 16] that without some shared or common knowledge, the member agents of such group systems have little hope of effective communication. The shared knowledge required could be common experiences, public information or an agreed set of definitions and meanings of basic communicable concepts. The term ontology is borrowed from philosophy to describe the latter, but the usage of ontologies has become common in the AI community. Particularly in the Semantic Web [8] and agent communities, the most common way to enable software agents to communicate is to give each of them the same specified conceptualization, or ontology, of the domains they are expected to work in.

A number of attempts have been made to lay guidelines for the development of ontologies in much the same vein as the traditional approaches to large software application development; these are discussed in Section 3. However, as this paper describes, these methods are generally unsuited to the development of the kinds of ontology generally required by multi-agent systems. Within the Intelligent Agent Lab at The University of Melbourne, we have designed a lighter approach to ontology development for multi-agent systems called EXPLODE, which is described briefly in Section 4. We then detail our experiences from applying EXPLODE to the task of developing an ontology to be used by agents in a multi-agent system that implements the recent theory of Value-Added Publishing.

2. Value-Added Publishing in Theory and Practice

One of the most important and challenging problems in computer science is the development of effective technologies that support access to online information. With the expansion of the Internet useful tools for finding relevant published data are needed. In [7] Vannevar Bush proposed what became the area of information access and information extraction/retrieval. This area was concerned with modelling, designing and implementing systems able to provide fast and effective content-based access to a large amount of information. Later the aims of these systems were re-defined. The need to estimate the relevance of documents to the user's information needs was realised. This is a very difficult and complex task, since it is flavoured by subjectivity and must cope with vagueness and uncertainty. Further, the traditional after-publication activities such as the publication of revisions, corrections, updates and new editions need to be managed within the automated online process.

To face these challenges Hal Berghel has identified a new area of e-publishing called Value-Added Publishing [2] which is addressed in this paper. In similar way to traditional e-publishing Value-Added Publishing (VAP) acknowledges the need for distributing static and dynamic publications via the Internet. Additionally it often includes elements such as sophisticated multimedia and hypermedia technologies, secure transactions and communications and billing and charging systems (though common standards are still lacking). VAP steps beyond that and tries to recognize the specific extensions, techniques and methods that increase the usefulness of a publication and help publishers to meet their specific reader's needs.

To assist their readers in finding what they need, publishers must consider where their publications fit in the overall picture of online documents. By associating publications with subjects and topics, the sea of online documents becomes easier to navigate. Users move from one cluster of documents to another, finding documents that cover similar topics in one virtual place and so greatly reducing the time taken to find what they want. Digital documents can be firmly integrated into a cluster of related documents - a significant benefit that couldn't be achieved by the mechanisms available to traditional publishing. The central issue in such an approach is the capability of publishing to not only provide the documents, but to represent and supply their connections to other data sources, as well as other valuable information. For more information and examples of VAP please refer to [2, 3].

In the following section some of the techniques to achieve the goal of adding value to publications will be briefly introduced.

2.1 VAP Techniques

In the context of VAP, where the information carriers and venues accept from and react to additional factors, the challenges are likely to be found in areas such as content enhancement, meta-data, confidence indicators and some others, all of which are described in the following section. They all contribute to some extent to the value of a publication and when combined together can significantly ease the difficult task of assessing a particular publication or comparing different publications. For brevity in this paper only few of them will be described.

2.1.1 Content enhancement

Content enhancement involves "enrichment of the semantic and syntactic content of a document" [2]. The theory that lies behind this is that the value of the content is dependent on people's ability to read it, view it, use it and reference it. From the information retrieval point of view, data which cannot be found or used is worthless. Enhancement leads to attempts to extract more meaning from the documents, i.e. the semantic content could be summarized, reported or abstracted by intelligent agents, natural language processing machines, translation systems and other mechanisms. When a document is cited, an appropriate reference entry can be automatically generated or looked up.

2.1.2 Meta-data

In the context of value-added publishing effective meta-data facilitates for example resource description and discovery, the management of information resources and their long-term maintenance. In the context of digital resources, there is a wide variety of metadata formats. These range from the basic records used by robot-based Internet search services, through relatively simple formats like the Dublin Core Metadata Element Set (www.dublincore.org) and the more detailed Text Encoding Initiative (www.tei-c.org) header to highly specific formats like the FGDC Content Standard for Digital Geospatial Metadata, the Encoded Archival Description (EAD) and the Data Documentation Initiative (Codebook http://www.icpsr.umich.edu/DDI) that continually increase in complexity.

2.1.3 Confidence Indicators

A practical way to provide useful information about a document or resource is to employ 'confidence indicators'. The purpose of a confidence indicator is to increase the assurance a reader can have that a particular document will be useful for them. The value of a document to a particular person may seem more a subjective issue than a technical one, but a lot of research has been done (especially by the commercial and advertising companies) in this area and it would be remiss not to describe the essential elements.

Confidence indicators work on the principle that if someone with similar interests to the user found a particular publication helpful, then it is likely to be helpful to the user as well. Generally, the size and nature of the audience of a publication is an indicator of the publication's quality or appeal.

An efficient way to guarantee to the reader that the document he/she is reading is a widely recognized one and has a value is to provide them with comments and notes from reviewers and leading authorities on the subject. Among all their other aims a function often performed by special interest groups and communities is to evaluate and recommend publications to their members.

The role of intelligent agents in providing confidence indicators is as an embodiment of the reader's preferences. Personalised agents are capable of and well-suited to providing these services.

Other techniques to achieve VAP include information customisation, perceived quality of the imprimatur, word profiles, digital libraries and others, which for brevity are not described in details in this paper (see [2] for more details).

The domain of VAP raises many challenges. One such arises from the fact that VAP is being developed by different groups of researchers, not necessarily consistent with each other. Some of its characteristics have already been implemented, for example automatic summary and word profile creation (Figure 1).

Other features of VAP are still struggling with the complexity of the process. Despite this the majority of the VAP techniques are ideally suited to implementation using intelligent agents, typically personalized, mobile or information extraction. As explained in the introduction of this paper, for a group of agents to communicate effectively, an ontology is needed. It is an established practice in agent-systems research that agents need an explicit shared specification of the concepts that can be communicated within the system. For example, one agent will extract the keywords from a particular document and send them to a second agent that will perform a search on CiteSeer

(http://citeseer.nj.nec.com) for other papers that match the same keywords. In order to communicate both agents need to know what a 'keyword' is and additionally its relation to 'topic' for example (since some digital libraries categorize documents by topics).

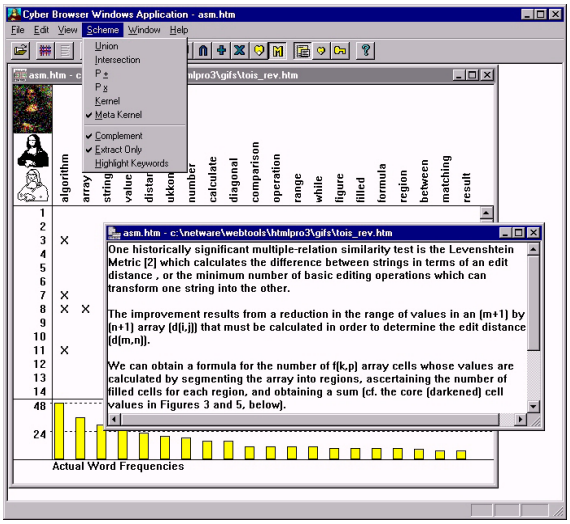

Figure 1:Keyword-oriented document extraction in Cyberbrowser [3]

VAP is a slightly different domain than publishing and it brings some requirements for ontological commitments that are not covered in the existing ontologies for publishing, such as 'approved by' or 'rated', which significantly influence the value of a particular document.

3. Choosing a methodology for ontology development

VAP is still not a strongly established field; its borders and the issues that it addresses often change. Thus the requirements for the ontology come from many different places, not necessarily consistent with each other - they are dynamic. This brings a need for a flexible methodology to develop the ontology, that allows change in the requirements after the initial elicitation and ensures that consistency between the requirements is maintained (through often testing) after each change. Additionally VAP is being developed by different groups of researchers and covers many different areas, sometimes not closely related to each other. The methodology must cater for the need to work on small fragments of the ontology independently as individual characteristics of VAP take shape. In these cases in order to preserve the unity of the domain frequent integration is important and so the chosen methodology must support this feature. In such a dynamic multi-agent environment as VAP, the final form of the ontology must be open for extension and change, thus the methodology chosen for its development should facilitate continuous modification of the ontology structure.

Some of the currently existing methodologies for ontology development that we have reviewed are described here.

3.1 Enterprise Model Approach

In 1995 Uschold and King formulated a methodology for building ontologies by recording their experiences from developing the Enterprise Ontology [23]. The Enterprise Ontology is a collection of terms and definitions relevant to business enterprises and includes knowledge about activities and processes, organizations, strategies, marketing and more [9]. According to this methodology the phases in ontology development are:

1. Identify purpose and scope - on the basis of the available knowledge the level of formality is described.

2. Ontology capture and identification of the scope - identify the key concepts and relationships that the ontology must characterize; produce unambiguous text definitions and identify terms to refer to such concepts and relations.

3. Ontology coding - commit to a meta-ontology; choose a representation language and write the formal ontology code; integrating existing ontologies.

4. Evaluation

5. Documentation

The techniques applied during stage 2 produce a list of potential relevant concepts and erase the irrelevant afterwards. The problem that arises when creating ontologies in this way is finding the balance between extracting a large number of entries and then deleting redundant terms on the one hand, and on the other initially proposing an insufficient number of concepts and subsequently needing to extend the ontology.

3.2 TOVE (Toronto Virtual Enterprise)

Developed in 1994 by [10] as a result of the experience of building an enterprise modeling ontology, the TOVE methodology involves constructing a logical model of the knowledge that is to be specified in the ontology, although the model is not built directly. Instead, competency questions are written in a formal language based on first-order predicate logic, and this formalization becomes the basis for a specification of the problem.

3.3 Bernaras et alia

This methodology is described in details in [19], here only the three key steps are summarized:

1. Specification of the application.

2. Preliminary design, based on relevant top-level ontological categories.

3. Ontology refinements and structuring.

3.4 KBSI IDEF5

Created by Knowledge Based Systems Inc. in 1994 the main purpose of this approach is to facilitate the design, development, modification and maintenance of ontologies [14]. This is one of the few methodologies that take into account the whole system and protocols and the main phases are:

1. Organizing and scoping: establishes the purpose, framework and the viewpoint of the ontology creation.

2. Data collection

3. Data analysis.

4. Initial ontology development

5. Ontology refinement and validation.

The validation tests according to this method are performed as a last stage and an essential part of the cycle is the iterative refinement and redevelopment of the structure.

3.5 Methontology

The METHONTOLOGY framework, as described in [4], aims to facilitate the construction of ontologies at the knowledge level. By considering three separate activities, namely the ontology development process, a proposed lifecycle and the methodology itself, the framework identifies which tasks should be performed when building ontologies, the steps to be taken to perform each task, and finally the products to be output and the means by which they are to be evaluated.

While suitable for the systems they have originally been developed for, we have identified these methodologies as inappropriate for our domain. As mentioned above VAP requires certain features of the methodology to be used, such as dynamism, the ability to change requirements during the development, frequent integration and validation. On the other hand, although specific features of VAP may evolve over time, a number of core characteristics are fundamental for the domain. Thus the methodology used to develop an ontology for VAP must facilitate establishing a constant baseline.

In view of the requirements, it is obvious that none of the above-described methodologies fully addresses these issues. Instead, we have decided to use the EXPLODE method [11] for the development of the ontology. In the following section a brief description of the EXPLODE method is presented.

3.6 EXPLODE

Lately much attention has been given to Agent Oriented Software Engineering [12]. Even with the emphasis on internal state and level of intelligence, agents are software modules that work autonomously in a given environment. They are pieces of software and applying principles from Software Engineering significantly contributes to the process of development, maintenance and management of multi-agent systems (MAS).

While developing ontologies, it is important to keep in mind that the final product will be a software component, a complex data structure that is to interact with the other components in the software system. Ontologies in this context they are software artifacts and they need to be treated accordingly. They are prone to the same fragility and maintenance needs that complicate software engineering, and it is entirely appropriate to apply software engineering approaches to the development of ontologies.

With all this in mind, EXPLODE was created as a method for ontology development by transferring key ideas from the eXtreme Programming methodology [1]. It is particularly suitable for dynamic and open environments thanks to its focus on immediate feedback and evaluation. Additionally the approach not only allows but favors change in the requirements at any stage of the development lifecycle. An overview of the EXPLODE method is presented in the next paragraphs.

Requirements

In the EXPLODE method requirements are determined as follows: the requirements for ontology development are extracted from both the competency questions and the system constraints that match the specific use and application of the ontology. Competency questions are the requirements that the users of the ontology specify. They indicate the scope and content of the ontology [13].

Planning

After the competency questions and the requirements of the system are specified the process of planning is initiated. The purpose of planning is to lay out the overall ontology lifecycle including development, integration and usage. At this stage important tasks are to identify the scope and problem, to identify the concepts and relations and to consider functional as well as quality requirements. The competency questions are prioritized and decisions are made such as which question to implement at a particular iteration and what happens if two or more questions contain the same concepts.

Baseline

The baseline is a simple ontology that focuses on architectural and usage requirements. At this stage answers to difficult technical or design problems are considered.

Iterations

Iteration is the process of repeating the same development activities multiple times, generally at increasing levels of detail or accuracy. Each iteration consists of three steps – testing the competency question, iteration planning and implementation.

Development

Development is part of each iteration and consists of actual implementation of a concept or relationship and refactoring. Refactoring is an important step in the implementation process the main purpose of which is to simplify the structure of the ontology, remove redundancy, eliminate unused functionality and increase quality.

Iteration Tests

The purpose of this phase is to test if the product satisfies the requirements, in the intended environment.

Acceptance tests

Continuous integration ensures that the ontology is integrated smoothly in the system and that there are no discrepancies between the ontology structure and the agents. The purpose of the acceptance tests is to minimize these discrepancies.

Maintenance

The maintenance concept in EXPLODE is primary and the important rules to follow are the same as in eXtreme Programming, i.e. to release 'early and often'. In effect, after the first test case the rest of the process can be classified as maintenance.

4. Developing the Ontology

The EXPLODE method, presented in the previous section, was deployed in order to develop the ontology for the domain of VAP. The procedure that was followed and some of the major decisions made are described in the following section.

Requirements

The first step according to the EXPLODE method is to fetch the requirements both from the customers in the form of

competency questions and from the system. In our particular case the majority of the questions have been extracted from an interview with Hal Berghel during his visit at The University of Melbourne in 2002. A partial list of the competency questions is given below:

1. How many other papers on a similar topic have cited this paper in their introduction?

2. What is the ranking of the author of this publication in the most-published people website?

3. Which of these two papers is more closely related to a particular topic?

4. What is the overall confidence indicators level of this paper?

5. What is the percentage difference between two published versions written by the same set of authors?

It is important to mention that during the process of elicitation of the competency questions, the ontological engineer did not play the 'standard' role of interface between requirements and development (as in software engineering for example), but acted rather as facilitator to the customer. In this sense only partial freedom was given to the user - they provided the initial set of questions, but after some planning and discussions had occurred the user had to chose between a number of options provided by the ontological engineer.

After the competency questions had been collected and preliminary analysis was performed the requirements of the system were determined. These were extracted from the VAP features, some of which were presented earlier in this paper. A list had to be generated with all the module's input and output, corresponding to each agent achieving a particular VAP goal. The process of system requirements extraction was performed manually, and in this case the result had to be expressed in a primitive enough and simple way. In the case of a well organized MAS the system requirements can be extracted by using middle agents that serve to collect the agent's capabilities. Middle agents have been used mainly to interact between end agents [22] and they suit very well the XP principle to extract the requirements from some form of system by one module and further process them by other module. In the case of VAP the agents are mainly personalized, mobile or information retrieval. Their capabilities are specified, for example finding the awards of a paper, rating of the author, creating a paper summary, words profiles or other. The existing agents can also be providers or requesters of information (where one agent could be a provider and a requester at different times). The middle agent needs to classify the other existing agents in regards of whether they are information providers or information requesters. The task becomes even more complex if the agent receives the parameters from another agent, not from a standard input.

Planning

For brevity, here we merge the details of the overall ontology planning and iteration planning. Two types of competency questions exist: core and non-core. For the domain of VAP the list of core competency questions was provided earlier. During the process of planning it was analysed in the perspective of their contribution to the ontology structure and the potential candidates for elements of the ontology were underlined. Non-core questions were provided later during the second and third iterations of the development and thus dynamism and change of the environment were introduced and handled. An abstract of the planning process is presented below:

1. How many other papers on the similar topic have cited this paper in their introduction? - this question considers that citing a paper in the introduction brings more value to the cited paper, than if it is cited in the main body text. To answer this question, firstly the topics of the collection of papers need to be identified and then filtered to only those on the similar or the same topic.

2. What is the ranking of the author of this publication in the most-published *people* website? - to answer this question the author of the paper needs to be extracted and then searched on the website for the most-published people in that particular area. An extension of the question would be to ask about specific topic are, instead of searching all the published papers in all the different fields. In this case the first step is to open the website and filter the authors by category. This question almost straightforward lead to the definition of a relation or property (which one is it was identified later): author has rank.

3. Which of these two papers is more tightly related to a particular topic? - this can easily identified by extracting the word frequencies and comparing the occurrence of the main topic words. This question also leads to identifying a statement 'paper has topic'.

4. What is the overall confidence indicators level of this paper? - to answer this question the following information must be collected: awards from special interest groups and communities; awards received from professional organisations and bodies. It is clear that a 'paper has a confidence indicator' but since so far these feature of VAP has not yet been implemented the type of the confidence indicator is unknown. For the purpose of the ontology development it will be assigned as text.

5. How much is the percentage difference between two published versions written by the same set of authors? - this question identifies the difference between two papers published by the same authors but in different journals or conferences for example. The significant contribution of this question is that it identifies the level of novelty in each paper - i.e. if the same people have published very similar or identical papers at different conference, this reduces the contribution of the overall collection of publications by the same authors. For example two published papers by the same authors that have higher than 60% similarities bring less credibility than two papers published by the same authors in a completely different topics (and thus the percentage of similarity is lower than 60%). Even though this agent has been deployed in other fields (for example The University of Melbourne has a system to compare the similarities between student assignments) it has not yet been widely used by publishers. For this reason the description of the algorithm for comparison will be skipped here. The only relevant information from this question is that the expression 'set of authors' implies that the cardinality of 'author' has to be larger than 1. It also can be assumed that a 'document has version' and this version can be 'published'. Additional types of versions are extracted from the description of the capabilities of agent 3.

System requirements

Some of the system requirements were mentioned in the analysis of the competency questions. Here some additional notes are included.

The use of the concept ‘introduction’ automatically lead to the question - what other parts can a document have? From the same question it was identified that each document contains a section where the citations are, i.e. a ‘reference’ section. During the first iteration the remaining parts of a document were not addressed.

The number of citations of an author was identified as an integer, which brought light how to encode ‘author has number of citations’- i.e. a property of the concept ‘author’ rather than binary relationship between two classes.

Additionally to the third question from the information about the keyword extraction agent's capabilities it was identified that the keywords are in the form of a list, there are exactly 5, and they are strings. Also the topic of a document is one single string, usually between 1 and 5 words.

During the process of iterations it was possible to modify the existing requirements or discover and additionally include new ones.

Baseline

The baseline ontology for the domain of VAP is presented in Figure 2.

- Content *(of a publication)*
  - Abstract *(automatic summary up to 200 words)*
  - Introduction *(cites other papers)*
  - References *(cites other papers)*
- Publication *(has version in a string format) (=document)) (has topic) (has author at least 1 [>=1 to many]) (has confidence indicator-text)*
  - Paper
- Person *(has name which is a string; can have a title: Prof., PhD.)*
  - Author *(writes papers) (has number of citations)*

Figure 2: Baseline of the VAP ontology

So far all the existing methodologies suggest that the ontology engineers create the baseline ontology manually. Our baseline ontology was also created manually. In a MAS though by deploying middle agents this process can be performed semi-automatically - the middle agent provides the concepts and the attributes, but the engineers define the structure. For example, based on the description of agent’s capabilities, a middle agent will suggest concepts such as ‘author’ and ‘paper’, the type of the output, i.e. ‘list of keywords’ and maybe other information, depending on what is provided by the information extraction agent, but the relationship between them - ‘writes’ or ‘has’ - will be defined manually.

Validation Tests

The validation tests ensure the gradual development of the ontology in a step-wise refinement fashion. In the case of VAP a competency question might be: "What is the topic of this paper?". For this example, the ontology developed so far is shown in Figure 2. Clearly, at this iteration the ontology already contains sufficient concepts to fully answer the competency question. This can be determined by searching the existing ontology for the words in the competency question. At this stage the comparison was mainly syntactic, based on pattern matching, but further it could be extended to semantic mapping [18, 21] (for example by calculating the distance between the concepts according to Wordnet). As the question can be answered using the current ontology the ontological commitments are valid and the next question is tested. Pattern matching for adding concepts to ontologies has already been used by a number of other researchers and we do not consider this as an obstacle when applying it to our case.

If some words in the competency question are not found in the current ontology, it is not yet safe to assume that the entire question must be implemented. The ontology may still contain some words that are already in the ontology, and to reimplement them would cause an inconsistency. During the development of the ontology for VAP, there has not been a single core question that was found to be already answerable by the existing ontology. The only concepts that appeared constantly were 'paper', and in two of the questions, 'author' and 'topic'. For 'paper' and 'author' the plurals were identified, i.e. when the next question asked referred to 'papers', it was identified that 'paper' already exists; similarly for 'author'.

Development and refactoring

During the development of the ontology there has not been a need of significant modifications of the already existing structure. During the implementation of the second question the concept ‘person’ was included and after that ‘people’.

In practice, there are two different approaches to ontology modification. The first one is to allow the user to identify inconsistencies in the model and change them. Modifying ontologies is covered in [17]. This sounds very easy, however from our experience we have discovered that removing classes or changing the structure (relationships) is a complex process that even intelligent agents lack capabilities to perform perfectly. In [5] is proposed a semi-automated approach for modifying ontologies, using management assistants. For the different modifications of the ontology corresponding assistants analyze the model to identify the consequences of the planned actions. The assistant then works in cooperation with the user to select and perform operations without violating the consistency of the knowledge base in order to achieve user's goal. An intelligent agent can predict the consequences of an ontology structure, thus the evaluation will be performed before the actual change. In this scenario the development of the knowledge-based agents is based on ontology reuse and development.

Iteration Tests

EXPLODE covers some of the major types of tests and following the guidelines was a straightforward process. During the development of the ontology for VAP, because of the simplicity of the ontology structure, the iteration tests were

mainly performed manually. Additionally the current version of Protégé (protege.stanford.edu) supports some redundancy tests while adding new classes.

Second Iteration
It was decided that during the second iteration non-core questions will be added to the structure. For example, one such question is: "What is the topic of this thesis?". During the implementation of this question the developer is alerted to the fact that the concept 'topic' is already in the ontology, and they can then add the concept 'thesis' with the relationship 'has topic'. Additionally, because the ontology already contains a concept 'paper' with a relationship 'has topic', the developer is asked if the new concept 'thesis' is related to the concept 'paper'. In this case the relations 'paper'-'topic' and 'thesis'-'topic' gives grounds for assuming a relation between 'paper' and 'thesis'. If two things are related to a common third thing, then it is quite reasonable to investigate the possibility that they have even more in common that just the third thing. If, as was the case described here, the two relationships are the same, the potential for commonality between the two things is even higher. By simple realizations such as these the produced ontology was as efficient as possible with very little redundancy.

Third Iteration
During the third iteration it was decided that the ontology will be extended to include synonyms. For this purpose the already implemented core and non-core questions were reviewed and reformulated with different words. During this iteration one of the problematic issues was the decision of whether to include 'document' as the same as 'paper' simply with a different name; to include it as the same as 'publication'. Subsequently 'document' was identified as the same as 'publication'.

Maintenance
As the domain of VAP is dynamic and researchers continually reconsider its important aspects, the hierarchical structure of the ontology is still being modified. The maintenance of the ontology consists of adding new concepts to the ontology, modification of existing concepts and deletion of concepts.

A partial list of the concepts (for readability limited to level of depth 3) of the final version is presented:

- Content *(of a publication)*
  - Abstract *(automatic summary up to 200 words)*
  - Introduction *(cites other papers)*
  - Main body
  - References *(cites other papers)*
- Library
- Publication (has Version in a Number format)
  - Paper
    - Technical Report
    - Short conference paper
    - Conference paper
  - Thesis
  - Master's
  - PhD
- Organization (quality of imprimatur in %)
  - Professional
    - ACM
    - IEEE
    - ACS
  - SIG
- Person (can have a title: Prof., PhD.
  (could be a member of an organisation)
  (has rating in the top 10 000 authors list
  http://citeseer.nj.nec.com/allcited.html).
  - Employee
  - Columnist
  - Editor
  - Reporter
  - Author
- Confidence Indicator
  - Readers (number of readers so far)
  - Review (could be by a SIG, an individual or a Professional Organisation) (reviews have dates)
  - Award (could be by a SIG or a Professional Organisation) (awards have dates)
  - Rating
    - Cited in Introduction by
    - Cited in References by
    - Rating of the author

5. Lessons Learnt and Conclusion

The lessons that we learnt during the development of the ontology were very similar to those described in [20]. We will list some additional findings:

5.1 Choice of methodology

As noted in Section 3 the methodology to follow when developing the ontology can have effect on the productivity and future integration. When choosing especially in multi-agent systems, features such as dynamism, flexibility, feedback, component-based development and others are very important and should influence the choice of the methodology. In our case VAP provided most of these features: with its constant development often by different groups of people (dynamic and component-based development) and need of a user feedback (confidence indicators and information customization are particularly personal and subjective measures for a value of a document).

5.2 Emphasis on planning

Planning might be easily overlooked, but in our case it was the stage when decisions for the whole development had to be made. For example, choosing to implement non-core questions during the second iteration and synonyms during the 3rd iteration.

5.3 Test different hierarchies

Different ontology engineers have different tendencies when it comes to choosing between a shallow or deep hierarchy. Since the structure can affect the speed of parsing and extracting information it is good for example to exploit different possibilities. Continuous integration and constant testing allows this to be performed without significant effort.

5.4 The role of Ontological Engineer

The role of OE could jump from developing the whole ontology without much user's input to the other extreme of not considering specific issues (for example choice of tools, language, depth of hierarchy and others) but simply following the user's requirements. In our case the role of the OE was

somewhere in the middle – the competency questions were reformulated by the user after some suggestions from the OE.

We also strongly felt that the suggestion to not include all the information [20] was a useful one and particularly supported by our choice of methodology.

In this paper we have described the process of developing an ontology for the domain of value-added publishing. We argued that none of the existing methodologies was applicable for our case that required dynamism, flexibility and development in stages. The chosen methodology EXPLODE was briefly introduced and applied for the development of the ontology. We have confirmed some of the lessons learnt from previous researchers when developing a lightweight ontology for a multi-agent system. We have also given some additional consideration when developing a purposive ontology.

6. References

[1] Beck, K. extreme programming eXplained: embrace change. Addison-Wesley, Reading (2000)

[2] Berghel, H. Value-Added Publishing Communications of the ACM (1999)

[3] Berghel, H., Berleant, D., Foy, T. and McGuire, M. Cyberbrowsing: Information Customization on the Web. Journal of the American Society for Information Science, 50 (6). 505-511, (1999)

[4] Blázquez, M., Fernández, M., García-Pinar, J.M. and Gómez-Pérez, A., Building Ontologies at the Knowledge Level using the Ontology Design Environment. In 11th Workshop on Knowledge Acquisition, Modeling and Management (KAW '98), (Banff, Canada, 1998).

[5] Boicu, M. and Tecuci, G., Ontologies and the Knowledge Acquisition Bottleneck. In 17th International Joint Conference on Artificial Intelligence (IJCAI 01), (Seattle, America, 2001).

[6] Borgo, S., Guarino, N., Masolo, C. and Vetere, G., Using a Large Linguistic Ontology for Internet-Based Retrieval of Object-Oriented Components. In Conference on Software Engineering and Knowledge Engineering, Knowledge Systems Institute, 528-534, (Madrid, Spain, 1997)

[7] Bush, V. As We May Think. The Atlantic Monthly, 176 (1), (1945)

[8] Ding, Y., Fensel, D. and Stork, H.-G. The Semantic Web: from Concept to Percept, eBusinessLeadership.net, (2001)

[9] Gomez-Perez, A. Ontological Engineering: A State of the Art Expert Update - The British Computer Society Specialist Group on Artificial Intelligence, 33-43 (1999)

[10] Grüninger, M. and Fox, M.S., The Design and Evaluation of Ontologies for Enterprise Engineering. In Workshop on Implemented Ontologies, European Workshop on Artificial Intelligence, (Amsterdam, Netherlands, 1994).

[11] Hristozova, M. and Sterling, L., An eXtreme method for developing lightweight ontologies. In Workshop on Ontologies in Agent Systems, 1st International Joint Conference on Autonomous Agents and Multi-Agent Systems, (Bologna, Italy, 2002).

[12] Tveit, A.: A survey of Agent-Oriented Software Engineering. NTNU Computer Science Graduate Student Conference. Norwegian University of Science and Technology, May (2001).

[13] Kim, H.M., Fox, M.S. and Grüninger, M. An ontology for quality management — enabling quality problem identification and tracing. BT Technology, 17 (4). 131-140, (1999)

[14] Knowledge Based Systems Inc. (KBSI). IDEF5 Ontology Description Capture Overview, (Knowledge Based Systems Inc., 2000)

[15] Lister, K. and Sterling, L., Agents in a Multi-Cultural World: Towards Ontological Reconciliation. In 14th Australian Joint Conference on Artificial Intelligence, (Adelaide, Australia, 2001).

[16] Lister, K. and Sterling, L. Reconciling Ontological Differences for Intelligent Agents. in Bouquet, P. ed. Meaning Negotiation, AAAI Press (Menlo Park, America, 2002)

[17] McGuinness, D., Fikes, R., Rice, J. and Wilder, S., An Environment for Merging and Testing Large Ontologies. In 7th International Conference on Principles of Knowledge Representation and Reasoning (KR2000), (Breckenridge, America, 2000).

[18] McGuinness, D.L., Conceptual Modeling for Distributed Ontology Environments. In 8th International Conference on Conceptual Structures Logical, Linguistic and Computational Issues (ICCS 2000), (Darmstadt, Germany, 2000).

[19] Ostermayer, R., Meis, E., Bernaras, A. and Laresgoiti, I. Guidelines on Domain Ontology Building, Deliverable DO1c.2, KACTUS ESPRIT Project 8145 (1996)

[20] Sayers, C. and Letsinger, R., An ontology for publishing and scheduling events and the lessons learned in developing it. In Workshop on Ontologies in Agent Systems, 1st International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2002), (Bologna, Italy, 2002).

[21] Stoffel, K., Taylor, M. and Hendler, J., Efficient Management of Very Large Ontologies. In American Association for Artificial Intelligence Conference (AAAI-97), AAAI/MIT Press, (1997)

[22] Sycara, K., Multi-Agent Infrastructure, Agent Discovery, Middle Agents for Web Services and Interoperation. In 3rd European Agent Systems Summer School (ACAI-01), (Prague, Czech Republic, 2001)

[23] Uschold, M. and King, M., Towards a methodology for building ontologies. In Workshop on Basic Ontological Issues in Knowledge Sharing, IJCAI-95, (Montreal, Canada, 1995)

[24] van Heist, G., Schreiber, A.T. and Wielinga, B.J. Using explicit ontologies in KBS development. International Journal of Human-Computer Studies, 45 (1997)

[25] Weinstein, P. and Birmingham, W., Service classification in a proto-organic society of agents. In Workshop on Artificial Intelligence in Digital Libraries, 15th International Joint Conference on Artificial Intelligence (IJCAI 97), (Nagoya, Japan, 1997)

An Ontology for Web Service Ratings and Reputations *

E. Michael Maximilien
IBM Corporation and NCSU
maxim@us.ibm.com

Munindar P. Singh
North Carolina State University
mpsingh@csc.ncsu.edu

ABSTRACT

Current Web services standards enable publishing service descriptions and finding services by matching requested and published descriptions based on syntactic criteria such as method signatures or service category. Emerging approaches such as DAML-S use DAML to formalize richer models for expressing capabilities of services. DAML-S would go beyond WSDL in terms of the richness of service descriptions. However, neither current Web services standards nor DAML-S provide a basis for selecting a good service or for comparing services that implement the same interface.

In our view, service selection is the key problem to enable application-to-application integration, which is the essential vision behind Web services. Existing approaches don't address service selection, because service selection inherently involves trust and must consider criteria that are external to any published description of a service, whether in WSDL or DAML-S. Accordingly, this paper develops an ontology in which ratings of services (aggregated into reputations) can be organized and shared so as to facilitate service selection. This model is expressed in DAML and includes domain-independent as well as domain-specific attributes.

1. MOTIVATION

Web services promise the dynamic creation of loosely coupled information systems. However, current approaches are logically centralized and lack key functionality, especially to locate, select, and bind services meeting certain criteria of quality. We are developing an architecture wherein software agents serve as *proxies* for clients and interact with one or more *agencies*. The agencies help gather and disseminate service reputations and endorsements. However, this and other service architectures leave open some key semantic questions. Specifically, a proxy agent should be able to discover and understand new service attributes from their descriptions, especially as they evolve over time. At the same time, an agency should be able to aggregate the right information about service quality and present it suitably formally described that they can be understood by proxies.

We address these semantic questions by developing a conceptual model of a service provider's reputation for delivering quality services. The conceptual model has a generic component (e.g., attribute types and common attributes such as price, on-time delivery, and so on) and can be enhanced with domain-specific components (e.g., closeness of itinerary to desired times, which makes most sense for services in the travel domain). By combining service considerations with semantic web representations, this work fits into the recent activity on semantic Web services.

*This paper is based on [Maximilien and Singh, 2002a].

2. BRIEF OVERVIEW OF ARCHITECTURE

Any conceptual model must rely upon an execution architecture. For concreteness, we describe our recently introduced proxy-based architecture [Maximilien and Singh, 2002b]. Our conceptual model is compatible with other architectures as well, but we lack the space to describe additional candidates here.

We propose the addition of a Web Service Agent Proxy (WSAP) to access each service. For our purposes, an agent is a software component that automates some tasks for its principal. Agents communicate with other agents, accept requests from their users, and are typically autonomous. A WSAP is an agent that acts as a proxy for clients of Web services. That is, a client would have a proxy agent for each service that it needs. The agents are knowledgeable about the various service standards. All activities of the client pertaining to the given service—including requests and responses, and communications with UDDI [UDDI, 2002] registries and bindings—occur via the proxy agent for that service. In this manner, when the client needs to bind to a service, it instantiates a proxy agent, which consults outside registries as well as reputation and endorsement agencies, helps find appropriate providers, records any feedback from the client, learns from the experience, potentially shares its knowledge with the external agencies and other agents, and hopefully helps find better providers the next time around. Figure 1 provides an overview of how clients typically use these proxy agents.

3. REPUTATION DATA MODEL

A distributed trust system consists of a set of *principals*, i.e., the parties involved either as service provider or requester. The principals interact with each other over a set of services. A *rating* of a service is a vector of attribute values. The *reputation* of a service is a general opinion i.e., it aggregates the ratings of the given service by other principals. Typically, a reputation would be built from a history of ratings by various parties. An *endorsement* of a service by a principal is modeled as a boolean scalar and a time limit on the validity of the endorsement.

Although we concentrate primarily on reputation, the underlying conceptual model is quite similar for endorsements as well. This is because an endorsement effectively states that the service being endorsed offers high quality with respect to some selected attributes: price, reliability, and so on.

In our proposed approach, agents maintain and contribute ratings to others and discover reputations. However, the ratings are based ultimately on feedback received from their clients. Some service qualities such as price and delay may be calculated automatically, whereas others may require human participation. Even the latter kind, although clearly harder to automate, can be accommodated by our architecture. For such cases, the application should be designed

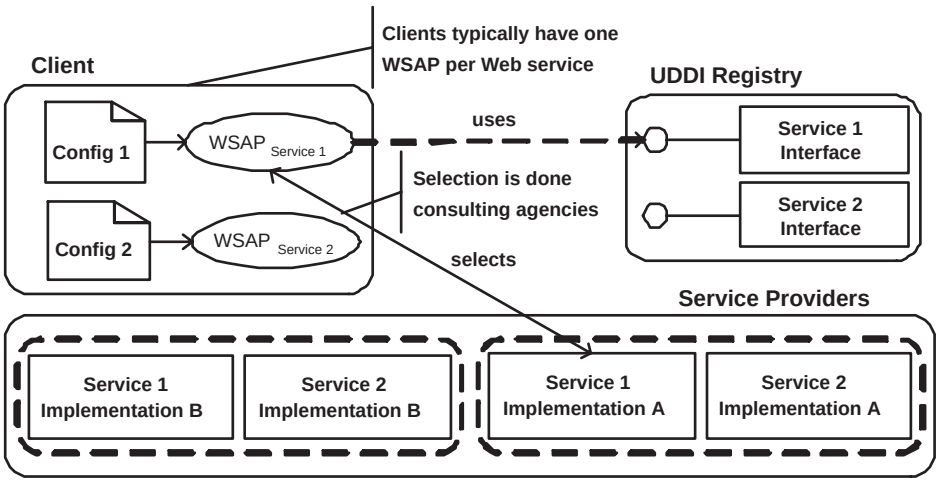

Figure 1: WSAP operation summary

so that it is possible to receive feedback from the human user after usage of the service. In this case, the agent will exploit this human feedback to learn the user's preferences.

4. CONCEPTUAL MODEL

A Web service represents a set of functions addressing a particular domain. For instance, a travel service might include functions to return a list of trips for a particular airline on a specified date, time, origin and destination airport. For each service we can extract a series of generic attributes and domain-specific attributes that apply to the service. For instance, for our travel service, a generic attribute could be the speed at which a search produces its results and a domain-specific attribute could be the accuracy of the return results—e.g., whether it includes up-to-the-minute trip information. From this example and what we discussed before, it is clear that a conceptual model for the reputation of a service must include the different categories of attributes that apply to a service.

Figure 2 summarizes our reputation model. The reputation of the service is a function of the various attributes that matter to a specific agent. For instance, an agent that cares more about accuracy of the trips returned would take into account this attribute more than the relative price attribute, which may be of greater concern to another user. In essence, the relative weight given to an attribute affects the overall reputation of the service and depends on the user of the service. This is analogous to how the ratings of human real-world services depend on the end user who is providing the rating. A car rental service that charges more but is flexible by giving convenient access at major airport may be of higher value to a business user than a person looking for a rental car for vacation purposes. So within a specific domain the reputation of the service depends on the subjective view of the user of the service on the various attributes that matters to the current agent that is proxying this service. Other factors that affect the reputation of a service include the following.

- The relative weights given to the attributes. Each agent will have preferences that biases it towards certain attributes and therefore make these attributes weigh more than others for a specific domain.
- The attribute aggregation algorithm. A simple weighted majority can be the normed algorithm but using different algorithms will affect the resulting reputation value.
- The set of endorsers of the service as well as the list of trusted endorsers for the agent. Again, matching endorsers will bias the reputation value.
- The history of the service. The reputation of older services will be affected more by previous usages.
- Any type of damping for the ratings as in [Zacharia and Maes, 2000] will affect the reputation value. Such damping is necessary to allow for a service's reputation to change easily based on its recent performance. For example, a service that acquired a bad reputation but then become better (and started receiving good ratings) can have its reputation improved since older ratings matter less than the newer ones.

Figure 3 shows a UML representation of our conceptual model, which describes the different components that make up the reputation of a service. First, a *Service* associates with one *Reputation*, which can have many *Rating*s. The reputation value is determined with a *ReputationAlgorithm* that aggregates the various *Attributes* that the agent determining the reputation chooses to consider. The reputation is also affected by a *History* that keeps previous ratings for the particular service. The rating for a service is determined by the *Principal* in question and calculated using the *RatingAlgorithm*. Any number of principals can endorse a service which might affect the calculation of the reputation since, for instance, an endorsement by a trusted third party can be considered of higher value than certain attribute values. As mentioned above, each attribute has a value and range, and associates with one or more *Domain*s. Domains act as collections of attributes for specific types of services.

4.1 Attributes model

For each domain, the attributes in that domain are important inputs to the overall rating and therefore the reputation of a service. Some attributes are common across domains and some are specific to domains. Each attribute has the following aspects.

- The value set for that attribute (and its allowed range or enumeration). For instance, an attribute such as failure rate or availability for a service can be expressed as a percentage. The speed of service function execution could instead be a simple bounded integer.

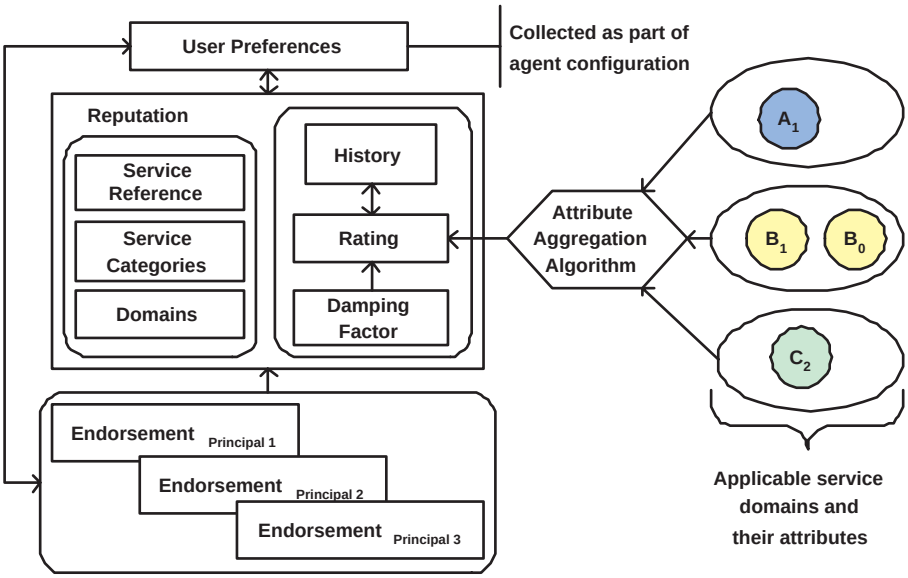

Figure 2: Generic framework model of a service reputation

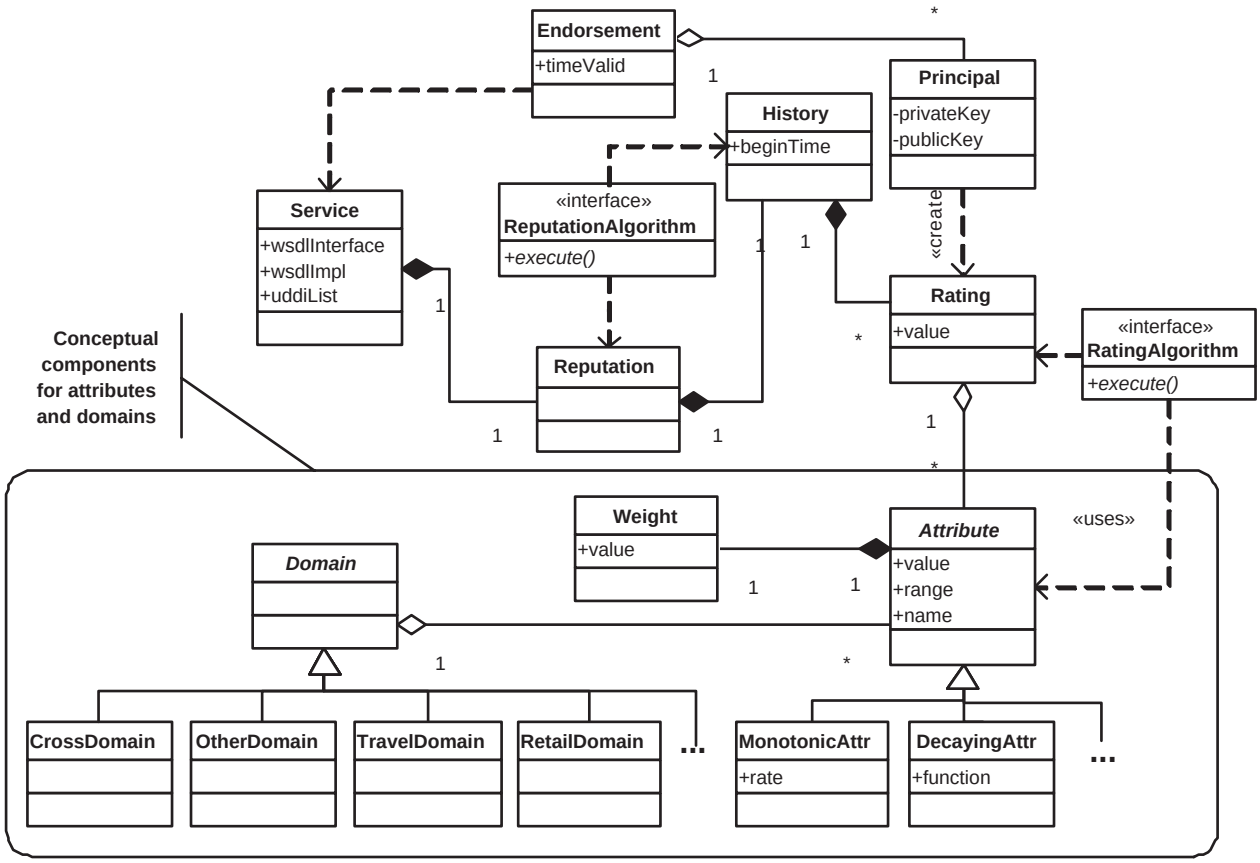

Figure 3: UML conceptual model for service reputation

- The domains that this attribute belongs to. For instance, is this a cross-domain attribute or an attribute specific for a domain? And within each domain, some attributes will be of greater importance than others—this can depend on a standard definition of attributes for a domain.
- The weight of the given attribute relative to its domain and the user preferences. This would determine the impact of this attribute on the final decision regarding a provider.
- The characteristic of the function from attribute values to ratings. For instance, some attributes such as price are monotonic, at least in typical business scenarios. That is, the more the price decreases the better. Generally all agents will consider price in the reputation calculation and have a preference for lower prices. Of course, if the price were to decrease in conjunction with reductions in the values for other attributes, the overall reputation might not improve. For instance, for the trip service we considered, if the price attributes of trips were to decrease while the promptness (on arrival and departure times) attribute were to become worse, then this decrease in price might not help the overall reputation of the trip service.

  The characteristics of attributes can be quite rich and need to be further categorized. Initially, we consider monotonically increasing and decreasing, S-shaped characteristics (where there is a substantial benefit to ratings when the given attribute improves, but only if it is above a tolerance threshold and not above a saturation threshold).
- The temporal characteristic of the attribute value. A possible temporal characteristic for attributes is decaying values where the decay function can vary from exponential to a step function. For instance, an attribute such as accuracy in the travel domain might be acceptable to allow a range of minutes up to a certain point. That is, the trip that is scheduled for 3:00 PM does not cause major harm if the actual departure time is 3:20 PM. However, a departure of 4:00 PM might become unacceptable for a user who depends on an on-time arrival with a layover of 30 minutes to catch a connecting flight. Other attributes might have values that decay more progressively rather than in a step-wise fashion.

4.2 Generic and domain-specific attributes

In general each domain that a service belongs to will have its own set of attributes that apply to all services of that domain. However, certain attributes will be cross-domain attributes.

As a specific example, a service such as car rental service will involve attributes belonging to multiple domains, such as the travel and retail domains. Attributes such as price belong to both domains, but an attribute such as flexibility of reservation changes has a specific meaning in the travel domain—clearly, allowing flexible changes to a travel reservation is a particular characteristic of a travel service that might allow important service differentiation for a particular client and the client's WSAP.

Determining the attributes that apply to a particular domain is nontrivial, but needs be decided by the community of users and providers as they settle upon ways to distinguish and evaluate different offerings. This typically occurs when markets form, where different parties position their offerings according to what they believe are their key qualities: speed, price, taste, and so on. A technical challenge here is to distribute the attributes among various domains so that an agency does not necessarily have to capture all possible attributes and a WSAP can search for services without consulting an excessive number of agencies.

4.3 Adding and disseminating attributes

Another important aspect that the proposed conceptual model allows is the creation and dissemination of new attributes for specific domains. A consequence of the attributes having a common model and the specific attributes being subclasses of the abstract attribute model is that new attributes with unknown characteristics can be added to the system. New attributes are disseminated via a domain definition that could contain references to common attributes as well as to domain-specific attributes. Our conceptual model can be readily mapped into a common schema in a standard notational framework such as the DARPA Agent Markup Language (DAML) [Horrocks, 2002]. The existence of such a schema would enable the description and widespread dissemination of attributes. Notice that the interesting component would be the conceptual model itself, not the notation, although agreeing on the notation is also essential.

4.4 Example

As a comprehensive example showing how the agent proxying a service can use the conceptual model we describe, imagine a travel reservation Web application. This application is used by agents to set up business and personal travel arrangements. Part of this application is a facility to search, select, and reserve a car rental that will be included as part of the overall travel arrangement. Since there exist many car rental companies, each advertising its services on the Web, it is easy to imagine that a common car rental Web service interface could be created. Each car rental company would provide an implementation of the service thus allowing its business to collaborate with others and thus integrate into coarse-grained services, such as a travel planning service. How is the WSAP proxying the car rental service able to pick the best service for a particular client?

Using our conceptual model, the WSAP could pick a suitable service implementation by looking at the reputation of the various implementations. The WSAP is configured with (1) the attributes that apply to the domain that the service belongs to and (2) the relative preferences that the WSAP's principal has for various attributes. According to our model, reputation is a function of the historical ratings provided by previous users filtered to take into account the attributes that matter most to the current WSAP's principal. So, for instance, if the current user weighs price as an important cross-domain attribute then services giving lower prices might be selected over those giving higher prices. Of course, the reputation for the service will depend on several attributes. For instance, some attributes such as comfort and reliability of the cars rented might matter more to a traveler who intends to use the vehicle for long subtrips than a business person on a tight budget. As another example, though the car rental service domain attributes overlap with a car selling service, certain attributes such as color choice will clearly have more significance for a buyer as opposed to a renter. Our model takes these subtleties into account, because of how the WSAPs are configured. Of course, the choice of which domain a service belongs to is an important precondition that must be satisfied prior to agent configuration. However, we are assuming that this is done as part of classifying the services prior to them being introduced for wide availability.

Our model is generic enough to allow the introduction of new attributes. For instance, if the domain definition for car rental was updated to include a new attribute such as safety, then any new agent that was configured with this attribute could take that attribute into account for its ratings calculations. The attribute's values could possibly be captured as part of a user survey or automatically by collecting information on accidents from the car types that the car rental company rented for certain periods and the relative safety

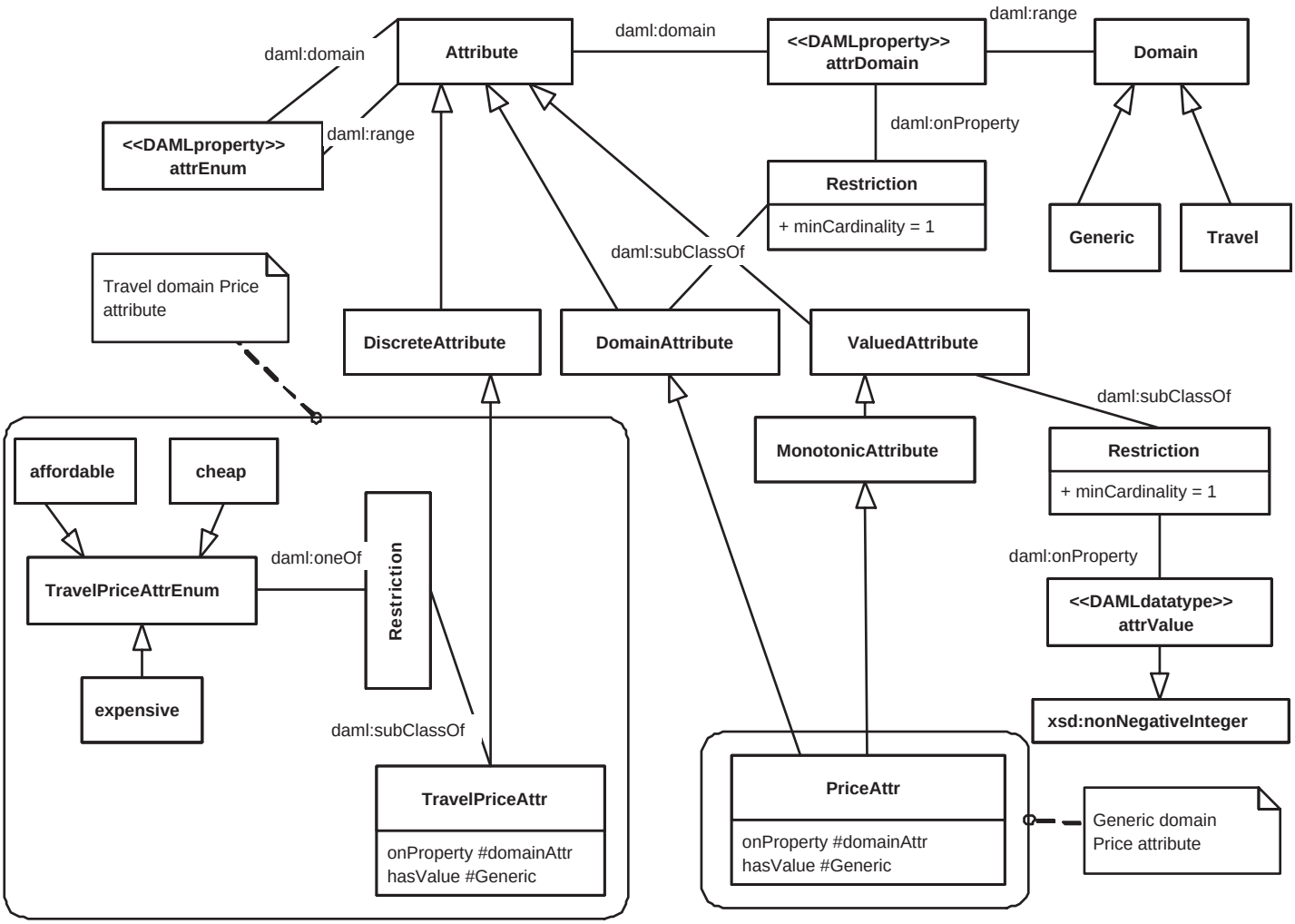

Figure 4: Part of DAML ontology for attributes showing two domain definition of price in UML [Baclawski et al., 2001]

outcome of the rentals.

A lot of engineering work goes into making a large application. The above example is no exception. The contribution of our approach is in streamlining the attributes so that their treatment is standardized up to a point and, where it is not standardized, placed explicitly under the control of individual WSAPs and their principals. New attributes can added on the fly. More importantly, the different agencies can upgrade to the new attributes or add an existing attribute that they had previously ignored. Likewise, the conceptual model enables the WSAPs to share information directly with each other, which extends their power further.

5. DISCUSSION

Our architecture opens up some interesting challenges, of which the following are germane to the topic of this paper.

Conceptual model of service attributes

Can we define a generic conceptual model for attributes reusable across domains?

Our agents are configured to capture the wishes of the application user. The agent uses this configuration information to maximize the utility of the user. However, in order for the agent to make intelligent decisions it will need more than just the reputation and endorsement agencies. It will need knowledge of attributes the user cares about, such as the following:

- The threshold of the values for attributes that the user is willing to accept.
- The risk tolerance of the user. For instance, the WSAP could find a reputable service matching the user's preferences but because it is relatively new, selecting that service could be regarded as higher risk than a known more mature service.

Answering the above question will enable our WSAP agents to efficiently and thoroughly capture the preferences of their WSAP users. Further, the service selection will often need to be fast because the user may be waiting for a service to be found. Therefore, the agent must be able to make quick decisions, comparing the user's preferences with information provided by the agencies.

Semantics of service attributes

How can we add semantics to service attributes, thereby allowing a WSAP to dynamically discover new attributes without having to be reconfigured or reprogrammed?

How can the agent acquire the knowledge of new attributes that were not specified by its client? That is, how can the agent relate the attributes specified by its client with attributes from other agents? The W3C's *Semantic Web* initiative [W3C, 2000] is a promising direction in capturing the semantics of service attributes.

Effects of attribute type on reputation

How should reputation be related to history of previous interactions? Should the effect of an interaction decay over time?

The notion of reputation is tightly bound to history and time. The reputations of human services tend to vary with time and recollection. In the digital world, history and memory can be collected easily. Because of this, the notion of reputation for humans and for agents have important differences. Some reputation systems [Zacharia and Maes, 2000] build in this decay effect. One approach to include time in our proposed architecture would be to, for instance, associate timestamps with attribute values, thereby allowing the reputation rating to weight attributes depending on their age. Further, since service quality will tend to change over time, decaying the reputation helps by reducing the effect of interactions over time, effectively increasing the currency of the evaluations.

A similar situation arises with endorsements. The goal is that it should be as easy as how people now look into the local newspaper and select a movie by looking at the number of stars it was awarded. Of course, a movie-goer may be biased towards a movie because of his knowledge of its actors, director, or producers—these intangibles will have conscious and subconscious implications to the movie-goer's decision. This is not completely the case for the software agents. However, endorsements do affect the agent's final decision. An endorsed service can similarly bias an agent towards a particular service regardless of its rating. How should agents weigh reputation ratings with respect to endorsements? What we need is a scheme by which attributes and endorsements can be systematically combined.

References

Kenneth Baclawski, Mieczyslaw K. Kokar, Paul A. Kogut, Lewis Hart, Jeffrey Smith, William S. Holmes, Jerzy Letkowski, and Michael L. Aronson. Extending UML to support ontology engineering for the semantic web. In Martin Gogolla and Cris Kobryn, editors, *UML 2001 - The Unified Modeling Language. Modeling Languages, Concepts, and Tools. 4th International Conference, Toronto, Canada, October 2001, Proceedings*, volume 2185 of *LNCS*, pages 342–360. Springer, 2001.

Ian Horrocks. DAML+OIL: a description logic for the Semantic Web. *Bulletin of the Technical Committee on Data Engineering*, 25:4–9, 2002.

E. Michael Maximilien and Munindar P. Singh. Conceptual model of Web service reputation. *ACM SIGMOD Record*, 31(4), December 2002a.

E. Michael Maximilien and Munindar P. Singh. Reputation and endorsement for Web services. *ACM SIGEcom Exchanges*, 3(1): 24–31, 2002b.

UDDI. Universal Description Discovery and Integration, 2002. http://www.uddi.org.

W3C. Semantic Web, 2000. http://www.w3.org/2001/sw/.

Giorgos Zacharia and Pattie Maes. Trust management through reputation mechanisms. *Applied Artificial Intelligence*, 14:881–907, 2000.

Towards HARMON*IA*: automatic generation of e-organisations from institution specifications

Daniel Jiménez Pastor
Department of Computer Science
University of Bath
BATH BA2 7AY, United Kingdom
dani@pitaweb.com

Julian Padget
Department of Computer Science
University of Bath
BATH BA2 7AY, United Kingdom
jap@cs.bath.ac.uk

ABSTRACT

There is a large gap between the emerging theory of institutions, or to be more accurate the norms that characterize them, how agents might reason about a symbolic representation of those norms and software tools to realize agent-trading platforms from high level specifications. We present an initial representation for institutions written in XML and their ontologies written in DAML+OIL and show how it has been used to specify aspects of simple auction house (the FishMarket [12]) from which we are able to generate automatically the components of the performative structure and the agent skeletons for an implementation on top of the JADE platform.

Keywords

Agents, ontology, multi-agent systems, institutions, norms, agent-trading platforms, institution definition language, performative structure, dialogical framework.

1. INTRODUCTION

The work presented here is a continuation of the ideas first reported in [22], in which a view was put forward of how implementations of agent organizations[1] could be generated from high-level descriptions of the structure and their norms. The content of [22] was preliminary in nature, in that the institution specification language was used as a guide for the manual development of the skeletal JADE [2] components, starting from the a description of the FishMarket [19, 12, 18] written in the Islander language [5, 4]. The aim of that paper was to extract parameterizable skeletal software components which might then be re-used in the construction of new institutions, and thus making a first step towards a tool for the rapid prototyping of institutions from their specifications. Since the main goal of the software tools featured here is the automatic generation of any kind of institution, we begin by completing the parameterizable components extracted from the FishMarket implementation, and then move on to describe progress towards creating a generic tool for the generation of e-organizations.

There are few tools in this area at the moment. One is the Islander agent institution graphical specification tool [7], which addresses aspects of organizational structure and dynamics. At a more technical level, there is the Bean Generator [17] plug-in for Protégé [16], which is used to obtain the Java files for the JADE platform [8] from an ontology specification. A more comprehensive solution, similar in scope to ours is the frame-based ontology language developed by Poggi *et al* [15], where a distinct ontology language inspired by KIF has been defined for use in the context of JADE and from which the implementation of agent systems in JADE has been generated. This is the closest in vision to what is described here. The most notable difference is their use of a bespoke, and somewhat simpler, ontology language in contrast to our use of DAML+OIL. The gap we are attempting to fill is the provision of an *integrated* tool for the automatic generation of an agent organization with its associated ontologies from its specification while aiming to utilize and to comply with current and emerging Semantic Web Group standards [24].

There are three aspects to the tools we are developing: (i) the overall HARMON*IA* framework which unifies notions of norm, rule, procedure and policy [5, 10, 21] and within this (ii) the specification of ontologies for concepts related to the organization, for example auction house, and for concepts related to the domain or domains about which agents may discourse, for example fish (!) (iii) the generation of agents to populate the organization and act as proxies for external agents wishing to interact with the organization. Thus we are motivated to attempt to create a complete tool for generating both the ontologies and the agents of the organization, so that the generated agents could be designed to use only the desired ontology, among other advantages. In the context of HARMON*IA* we foresee the chance to generate e-organizations for different agent platforms, but at this stage we are using our own IDL (Institution Definition Language) rather than Islander, JADE as the target platform and DAML+OIL [3] as the ontology language.

For the medium term, we are considering how we may target a range of agent platforms [2] and therefore a richer IDL will be required to accommodate these demands. Also, we are preparing for migration to OWL [23], shortly expected to be approved by W3C. In the longer term, a GUI tool for the graphical creation of the institution specification is planned as part of the HARMON*IA* framework (currently under development).

[1] Although it has become conventional to write of agent institutions, both for kinds of institutions and their implementations, we feel it is important to make the distinction between institutions as classes and organizations as instances (see [5] for a diagrammatic presentation of this argument), which is consistent with the defining work on the subject from an economic perspective by Douglass North [13]

[2] Despite the fact we have chosen JADE as agent platform, due to the large user community, the scope for wide deployment of generated organizations and compliance with FIPA standards [6], there is an large list of agent platforms as seen at [1], and among them we highlight the following: FIPA-OS, AAP and Zeus.

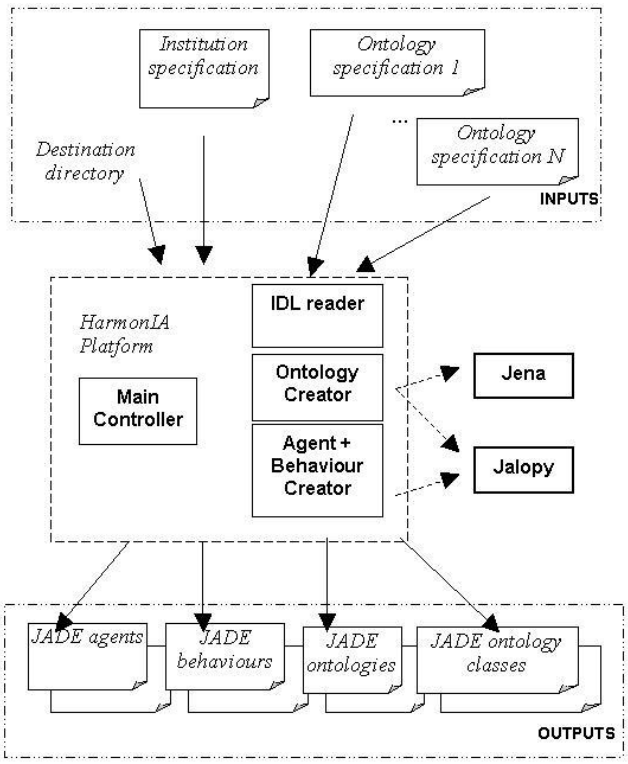

Figure 1: Elements of the HARMON*IA* architecture for the JADE platform

2. SYSTEM ARCHITECTURE

We begin with a description of the overall architecture of the toolset we are constructing – see figure 1 for an overview of the components. The organizational structure is firmly based on the work previously reported in Noriega's thesis [12] and subsequently expanded by Rodriguez Aguilar [18]. The final HARMON*IA* platform will contain much more than is shown here, since it will encompass norm specification and verification tools covering the range from norm through rule to protocol and appropriate theorem provers and model checkers as required.

2.1 Generic Institution Design

The idea norms that characterize institutions and the view of organisations as instantiations of institutions satisfying those norms are both increasingly accepted as important aspects in the engineering of agent-based systems. The particular approach we take began with the exposition by Noriega [12], and later Rodriguez Aguilar [18], of a dialogical specification of agent interaction via sets of illocutions (speech acts), the identification of interaction sequences (protocols) as so-called scenes and the linking of scenes to make a graph called the performative structure. Subsequently these ideas have been captured in the ISLANDER institution description language [5] and a graphical toolkit [4]. At the same time, the original ideas first put forward in the ISLANDER language have been refined to focus more on norms as guides and constraints on agent behaviour [14, 10] in contrast to the somewhat rigid conversation structures conceived originally. What we are presenting here are some of the preliminary thoughts on a second generation institution description language in which we are moving from the ad-hoc syntax of ISLANDER to a widely recognized representation and from the monolithic structure of ISLANDER specifications to one of composable components in conjunction with the development of the translation schemes from specification to implementation that were first sketched by Vickers [22].

The construction of an electronic organization begins with a description written according to the XML schema we have developed for the purpose. The corresponding ontology to ground the elements in this description is currently under construction.

Each institution has one or more dialogical frameworks, which determine the illocutions that can be exchanged between the agents in each scene. In order to do so, an ontology written in DAML+OIL and a list of the possible roles that agents may play are defined for each dialogic framework, which fixes what are the possible values for the concepts in a given domain. This DAML+OIL ontology will be parsed using the Jena toolkit [11].

Each agent role(see later), can have one or more behaviours, each one able to contain multiple sub-behaviours, corresponding to the functionality of JADE platform agents.

2.2 System Overview and Dependencies

Before sketching the key phases in this part of the HARMON*IA* architecture, we will briefly mention two important third party APIs we use in our tool: (i) Jena [11] is used to read the DAML+OIL ontology and hence process the information to create all the files related to the ontologies, used in the JADE environment. (ii) Jalopy [9] is a source code formatter for Java and that is responsible for the layout of all the code presented later in this paper.

2.3 Inputs

There are three inputs to this basic instantiation of the HARMON*IA* framework:

- The name of the directory where the generated organization files will be written.
- The institution description – written in the XML-based IDL.
- One or more ontology files.

The institution description is derived from the ISLANDER language first described in [5], which develops the features described in [18] and which has forms part of the institution editor reported in [4]. Various shortcomings with the ISLANDER approach and representation have lead us to develop a second generation IDL based on DAML+OIL – with the intention to migrate to OWL [23] in the very near future. In addition, for the purpose of this first exercise, we have incorporated JADE platform specific features, see the example in Figure 2, because in the short term we are focusing on delivery on one platform. For the longer term, we are considering how the XML schema may be parameterized to support a range of platforms.

As seen in Figure 2, we follow the design set out in Section 2.1. We use this file to create the data structure of the institution for the JADE platform. Each dialogic framework, with the tag of the same name, has one ontology, and as we see in the code, the tag `Ontology` indicates where to find the ontology, the type of which is defined in the `specificationLanguage` attribute (this example uses a DAML+OIL ontology), so the system knows how to parse it.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Institution id="FM2003">
  <DialogicFramework id="FishDialFrw"
     contentLanguage="PROLOG">
    <Ontology id="FishONTO"
       specificationLanguage="DAML"
       file="file:///C:/oasPaper/fish.daml" />
    <Role id="Boss"
       type="InternalRole">
      <Behaviour id="Open"
         type ="SimpleBehaviour"/>
      <Behaviour id="Discuss"
         type ="CompositeBehaviour">
        <Behaviour id="InitialResolution"
           type ="OneShotBehaviour"/>
        <Behaviour id="FinalResolution"
           type ="OneShotBehaviour"/>
      </Behaviour>
    </Role>
    <Role id="Admitter"
       type="InternalRole">
      <Behaviour id="Admit"
         type ="CyclicBehaviour"/>
    </Role>
  </DialogicFramework>
</Institution>
```

Figure 2: XML based IDL file for the JADE platform

A word of clarification is in order about the term "role". Superficially, this can be thought of as a synonym for type, but what it actually captures is both deeper and more flexible. The principle of role is borrowed from a line of research in security – both in software and in physical organizations – called Role Based Access Control (RBAC) [20] which associates ideas of responsibility, constraint and obligation with a given role and captures relationships between roles, such as whether one subsumes another and whether one role is incompatible with another, such as because it is either impossible to fulfil the requirements of each role simultaneously or because their combination may create a security hole. From the norm perspective, where we recall that a norm is an expression of a guide or constraint on behaviour, it becomes clear that a role is a coherent subset of the norms of an institution that taken together prescribe the limits of action when playing a particular role. Thus it is that in describing an institution, a key part of the modelling process also identifies roles that agents may play within that institution (e.g. buyer, seller, admitter, accountant, auctioneer, etc., see Esteva *et al* [5] for more detail and Vazquez [10] for a comprehensive treatment).

Hence we use the word "role" to refer to the constraints on an agent's behaviour. From past practice, we have found it convenient to classify roles as one of two kinds: `InternalRole` if is a staff agent, responsible for aspects of running the institution, like the Boss or the Admitter agent in an auction or `ExternalRole` for a client-agent written by a third party that visits the institution, like the Buyer or Seller agents.

Finally, in the `Behaviour` tag, we have two attributes: the `id` and the `type`, which identifies the `Behaviour` class of the JADE platform from which we are inheriting.

As it stands, the IDL file contains enough information to describe the gross behaviour of the agent, but not how it communicates, for which we need the illocutions to be used in the conversation protocols of the scenes of the performative structure. These aspects are currently under development and will be reported on separately later.

To illustrate how the framework we have operates, instead of providing a long DAML+OIL ontology file, we will show the complete UML diagram representation of the DAML+OIL file in the Figure 3. Then, to show each aspect of the translation in detail, some DAML+OIL code will be posted.

Finally, we provide solutions for the problems or constraints found while dealing with the translation of a DAML+OIL ontology into a Jade ontology, due to the particularities of the Jade platform and model incompatibilities between DAML+OIL and Java, such as multiple inheritance, multiple ranges, anonymous classes and some aspects that will be discussed in the following subsections.

2.3.1 Multiple inheritance

MI is a key aspect of DAML+OIL, but is not a feature of Java, because a class can only inherit from one class and from multiple interfaces, and therefore only the methods inherited from the superclass and not those from the interfaces may have an implementation.

But as an ontology is basically data, not operations, we just need Java classes with their attributes and only accessor methods to represent it in the Jade platform. So, we can translate all the classes of the DAML+OIL ontology into interfaces to obtain the multiple inheritance capability, and then, for each interface, generate a wrapper class with the accessor methods to get and set their attributes. The only issue we have to control is to rename attributes from different parents if they have the same name. This solution will be fully working before summer 2003, but at present, if a class in the input has more than one parent, we will only consider the first and ignore the rest (with a suitable warning). Example:

```
<daml:Class rdf:ID="BabySquid">
  <rdfs:subClassOf rdf:resource="#Squid"/>
  <rdfs:subClassOf rdf:resource="#SmallFish"/>
</daml:Class>
```

Class BabySquid as specified above has two parents: Squid and SmallFish. In the translation, only Squid class will appear as the parent.

Some other solutions based on design patterns, as the Bridge, State and Strategy design pattern have been studied, but they provide solutions basically for different method implementations that work in a plain environment, but not embedded in a framework like Jade, because Jade can only know the accessor methods of the classes, but no others.

Another solution that requires the collaboration of the ontology designer would be using a delegation – "black-box" inheritance – approach, where BabySquid would have two attributes, one of class Squid and one of class SmallFish, but we discarded it because we want automatic generated solutions. We note that the ontology language of Poggi *et al* [15] provides single inheritance making the job of translation to Java/JADE somewhat less difficult than the task we have here in working from DAML+OIL.

2.3.2 Multiple ranges

In Java, each variable can be only of one type. This contrasts with the permitted multi-range properties in DAML+OIL. A way

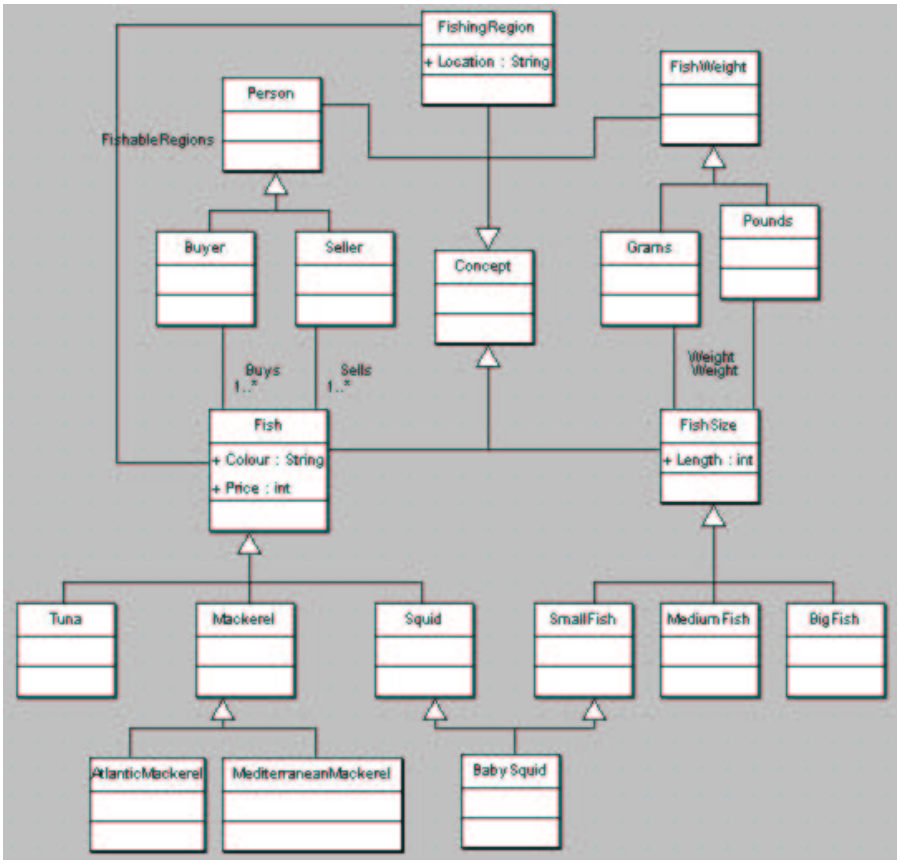

Figure 3: UML class diagram corresponding to the DAML+OIL ontology

to work around this in Java is to have many properties each with a different type instead of one property with many types. As with multiple inheritance, if this constraint is not observed, we only consider the first type given for that property and ignore the rest. Consider the following object property declaration:

```
<daml:ObjectProperty rdf:ID="Weight">
  <daml:domain rdf:resource="#FishSize"/>
  <daml:range rdf:resource="#Grams"/>
  <daml:range rdf:resource="#Pounds"/>
  <rdf:type rdf:resource =
     "http://www.w3.org/2001/10/daml+oil#UniqueProperty"/>
</daml:ObjectProperty>
```

Here, only Grams will be recognized as the possible range class (and therefore type) for the Weight object property. The conflict can be resolved by creating two different object properties for the FishSize class, named WeightGrams and WeightPounds respectively, as follows:

```
<daml:ObjectProperty rdf:ID="WeightGrams">
  <daml:domain rdf:resource="#FishSize"/>
  <daml:range rdf:resource="#Grams"/>
  <rdf:type rdf:resource =
     "http://www.w3.org/2001/10/daml+oil#UniqueProperty"/>
</daml:ObjectProperty>

<daml:ObjectProperty rdf:ID="WeightPounds">
  <daml:domain rdf:resource="#FishSize"/>
  <daml:range rdf:resource="#Pounds"/>
  <rdf:type rdf:resource =
     "http://www.w3.org/2001/10/daml+oil#UniqueProperty"/>
</daml:ObjectProperty>
```

Although the solution of splitting automatically a multiple range can be done easily, in the OWL specifications they explain that a multiple range has to be treated as the intersection of ranges, and as we explain later (see section 2.3.5) with respect to class union, intersection and negation, we do not generate any solutions for logic/set operators.

2.3.3 Identifier syntax

This is a perennial problem in any language translation task when the source language has a less restrictive identifier syntax than the target language. Thus, the input:

```
<daml:Class rdf:ID="Mediterranean-Mackerel">
  <rdfs:subClassOf rdf:resource="#Mackerel"/>
</daml:Class>
```

contains a variable which is not a valid as a Java variable because it contains a hyphen. The variable will be renamed to `Mediterranean_Mackerel` and a warning will be echoed with the new variable name. Also, variables cannot be reserved Java keywords, cannot have a hyphen as in the example, or start with a number. For all this cases, HARMON*IA* will rename the variable and print a warning.

Agent Boss imports the ontologies the agent uses

```
package FM2003;
import FM2003.Behaviours.*;
import FM2003.Ontologies.FishONTO.*;
```

Standard agent header, followed by initialization of properties and getting an instance of the ontology the agent uses.

```
public class Boss extends Agent {
    private ServiceDescription sd;
    private Codec codec;
    private Ontology ontology;
    private ContentManager manager;
    private DFAgentDescription dfd;
    private MessageTemplate mt;

    public Boss() {
        sd = new ServiceDescription();
        codec = new SLCodec();
        manager = (ContentManager) getContentManager();
        dfd = new DFAgentDescription();
        ontology = FishONTO.getInstance();
    }
```

Here we see the setup method, where the agent registers with the Service Descriptor as an internal agent and with the Directory Facilitator with FishONTO as ontology and FM2003 as ownership. After registering in the DF, creates and adds the behaviours Open and Discuss.

```
    protected void setup() {
        sd.setName(getName());
        sd.setType("InternalRole");
        sd.setOwnership("FM2003");
        dfd.setName(getAID());
        dfd.addServices(sd);
        dfd.addOntologies("FishONTO");
        try {
            DFService.register(this, dfd);
        } catch (FIPAException e) {
            System.err.println(getLocalName() +
             " registration with the DF failed because - "
               + e.getMessage());
            doDelete();
        }
        manager.registerLanguage(codec, "PROLOG");
        manager.registerOntology(ontology, "FishONTO");
        Open behaviour_0 = new Open(this);
        addBehaviour(behaviour_0);
        Discuss behaviour_1 = new Discuss(this);
        addBehaviour(behaviour_1);
    }
...
}
```

Figure 4: Boss agent imports, initialization and setup

2.3.4 Root classes

As we want to generate automatically an ontology for the JADE platform, each class must inherit from one of the following classes: AID, AgentAction, Concept or Predicate, which must also be defined in the ontology as we see below. We note that we have only as yet worked with a few simple ontologies and this will have far from demonstrated the full range of inputs – and problems – that we will need to be able to handle.

```
<daml:Class rdf:ID="AgentAction">
</daml:Class>

<daml:Class rdf:ID="AID">
</daml:Class>

<daml:Class rdf:ID="Concept">
```

```
package FM2003.Behaviours;
import FM2003.*;
import FM2003.Ontologies.FishONTO.*;
```

As in the agent imports, we import their own ontology. And we create the behaviour inside the package Behaviours.

```
public class Discuss extends CompositeBehaviour {
    private InitialResolution subBehaviour_0;
    private FinalResolution subBehaviour_1;

    public Discuss(Agent a) {
        super(a);
    }
```

As is work for the programmer which order a behaviour uses their subBehaviours, we only add as many properties as subBehaviours the behaviour has, in this case one for InitialResolution and other for FinalResolution.

```
    public boolean checkTermination(
                      boolean currentDone,
                      int currentResult) {
        //This method must be implemented
        return false;
    }
    public Collection getChildren() {
        //This method must be implemented
        return null;
    }
    public Behaviour getCurrent() {
        //This method must be implemented
        return null;
    }
    public void scheduleFirst() {
        //This method must be implemented
    }
    public void scheduleNext(
                   boolean currentDone,
                   int currentResult) {
        //This method must be implemented
    }
}
```

Figure 5: Behaviour Discuss methods

```
</daml:Class>

<daml:Class rdf:ID="Predicate">
</daml:Class>

<daml:Class rdf:ID="Fish">
  <rdfs:subClassOf rdf:resource="#Concept"/>
</daml:Class>

<daml:Class rdf:ID="Mackerel">
  <rdfs:subClassOf rdf:resource="#Fish"/>
</daml:Class>
```

From the perspective of Java generation, AID, AgentAction, Concept and Predicate are interfaces. So, class Fish will extend the Concept interface, and class Mackerel will inherit from Fish class.

2.3.5 Union, intersection and negation classes

This is one feature of DAML+OIL that is far too powerful to be used at present with programming language such Java, so we cannot provide a solution for this capability, but we will continue researching all the possibilities to benefit from this richer expressiveness of DAML+OIL or OWL.

We create a new package for every ontology, inside the Ontologies package. Then we write the imports, although we do not show them. Finally, we create many constants for the vocabulary, useful for the understanding of the code, although it is also possible not to create them, but the code would be much more untidy and less clear.

```
package FM2003.Ontologies.FishONTO;
...
public class FishONTO extends jade.content.onto.Ontology {
    protected static Ontology theInstance = new FishONTO();

    // Vocabulary
    public static final java.lang.String FISH = "Fish";
    public static final java.lang.String FISH_FISHABLEREGIONS = "FishableRegions";
    public static final java.lang.String FISH_PRICE = "Price";
```

We create all the primitive schemas at the beginning to make the code nicer, like the vocabulary. Then, we add the concepts classes to the ontology. We only show how to add the concepts, but the AID, AgentAction and Predicates can be added the same way.

```
    protected FishONTO() {
        super("FishONTO", BasicOntology.getInstance(),
            new ReflectiveIntrospector());

        try {
            PrimitiveSchema stringSchema = (PrimitiveSchema) getSchema(BasicOntology.STRING);
            PrimitiveSchema integerSchema = (PrimitiveSchema) getSchema(BasicOntology.INTEGER);
            PrimitiveSchema floatSchema = (PrimitiveSchema) getSchema(BasicOntology.FLOAT);
            PrimitiveSchema dateSchema = (PrimitiveSchema) getSchema(BasicOntology.DATE);
            PrimitiveSchema booleanSchema = (PrimitiveSchema) getSchema(BasicOntology.BOOLEAN);

            // adding Concept(s)
            ConceptSchema FishSchema = new ConceptSchema(FISH);
            add(FishSchema, Fish.class);
            ConceptSchema MackerelSchema = new ConceptSchema(MACKEREL);
            add(MackerelSchema, Mackerel.class);
```

After creating all the schemas for every class of the ontology, we add to them all their fields. We only show how to add fields to the Fish concept, to be brief.

```
            // adding Fields
            FishSchema.add(FISH_FISHABLEREGIONS, FishingRegionSchema, ObjectSchema.MANDATORY);
            FishSchema.add(FISH_PRICE, integerSchema, ObjectSchema.MANDATORY);
```

For all the schemas, we add the inheritance expressed in the model. With this, the creation of the ontology is done. We also add the getInstance method to get an instance of the ontology, used in the Agents files.

```
            // adding Inheritance
            MackerelSchema.addSuperSchema(FishSchema);
        } catch (java.lang.Exception e) {
            e.printStackTrace();
        }
    }

    public static Ontology getInstance() {
        return theInstance;
    }
}
```

Figure 6: Translation of the FishONTO ontology

2.3.6 Anonymous classes

Although it is fairly easy to implement this DAML+OIL functionality in Java by using the old Lisp GENSYM technique, at present we do not support DAML+OIL anonymous classes due to time constraints. These classes do not have meaning by themselves, rather they are used to capture abstractions embedded in other classes, and we are also mindful of the fact that OWL-Lite does not support anonymous classes, and depending on which version of OWL becomes the preferred means of expression, the problem could potentially go away.

2.3.7 Datatypes as types

Due to actual technical limitations of the Jena toolkit, and and bearing in mind future compatibility with OWL, the ontology input does not support the use of datatypes as types. In the next DatatypeProperty, we see that the range specifies a datatype, without referencing the integer schema. Thus, the following is incorrect:

```
<daml:DatatypeProperty rdf:ID="Length">
  <daml:domain rdf:resource="#FishSize"/>
  <rdfs:range>
    <xsd:integer />
  </rdfs:range>
  <rdf:type rdf:resource =
     "http://www.w3.org/2001/10/daml+oil#UniqueProperty"/>
</daml:DatatypeProperty>
```

The range part has to be rewritten with a reference to the integer schema:

```
<daml:DatatypeProperty rdf:ID="Length">
  <daml:domain rdf:resource="#FishSize"/>
  <daml:range rdf:resource =
     "http://www.w3.org/2000/10/XMLSchema#integer"/>
  <rdf:type rdf:resource =
     "http://www.w3.org/2001/10/daml+oil#UniqueProperty"/>
</daml:DatatypeProperty>
```

2.3.8 Split properties

Again, due to current technical limitations, we require the multiplicity information of the property to be supplied in the type attribute of the class tag, instead of having two tags: ObjectProperty/DatatypeProperty and UniqueProperty. So, all the information related to a property must be embedded in their tags, without splitting the information between siblings. For example:

```
<daml:DatatypeProperty rdf:ID="Price">
  <daml:domain rdf:resource="#Fish"/>
  <daml:range rdf:resource =
     "http://www.w3.org/2000/10/XMLSchema#integer"/>
</daml:DatatypeProperty>
<daml:UniqueProperty rdf:about =
   "file:/C:/oasPaper/fish.daml#Price"/>
```

Has to be expressed this way:

```
<daml:DatatypeProperty rdf:ID="Price">
  <daml:domain rdf:resource="#Fish"/>
  <daml:range rdf:resource =
     "http://www.w3.org/2000/10/XMLSchema#integer"/>
  <rdf:type rdf:resource =
     "http://www.w3.org/2001/10/daml+oil#UniqueProperty"/>
</daml:DatatypeProperty>
```

2.4 Outputs

The outputs of the system are the Java skeleton files of the organization for the JADE platform. All the files are created in the specified destination directory, while within the files, all the objects are in a package named by the institution ID. So, if the institution ID is `FM2003`, and we have as destination directory the root directory, and the code from Figure 2 as the IDL input file, we will get four kinds of files:

1. Agents – found in the root destination directory, belonging to the package FM2003:
   `FM2003/Boss.java`
   `FM2003/Admitter.java`
2. Behaviours – found in the Behaviours directory and belonging to the package FM2003.Behaviours:
   `FM2003/Behaviours/Open.java`
   `FM2003/Behaviours/Discuss.java`
   `FM2003/Behaviours/InitialResolution.java`
   `...`
3. An ontology file. In this case, is located inside package FM2003.Ontologies.FishOnto, and in a directory equal to the package name:
   `FM2003/Ontologies/FishONTO/FishONTO.java`
4. Ontology classes, located in the same package and directory as their own ontology file:
   `FM2003/Ontologies/FishONTO/BabySquid.java`
   `FM2003/Ontologies/FishONTO/Mackerel.java`
   `FM2003/Ontologies/FishONTO/Fish.java`
   `...`

We add this class to the same package as the ontology file. Again, we have omitted all the imports. As we see in the DAML+OIL examples, class Fish implements the Concept interface.

```
package FM2003.Ontologies.FishONTO;
...
public class Fish implements Concept {
    protected int Price;
    protected List FishableRegions = new ArrayList();

    public void setPrice(int value) {
        this.Price = value;
    }
    public int getPrice() {
        return this.Price;
    }
    public void addFishableRegions(FishingRegion elem) {
        List oldList = this.FishableRegions;
        FishableRegions.add(elem);
    }
    public boolean removeFishableRegions(
                                FishingRegion elem) {
        List oldList = this.FishableRegions;
        boolean result = FishableRegions.remove(elem);
        return result;
    }
    public void clearAllFishableRegions() {
        List oldList = this.FishableRegions;
        FishableRegions.clear();
    }
    public Iterator getAllFishableRegions() {
        return FishableRegions.iterator();
    }
    public List getFishableRegions() {
        return FishableRegions;
    }
    public void setFishableRegions(List l) {
        FishableRegions = l;
    }
}
```

Class Fish has two attributes: Price (single cardinality) and FishableRegions (multiple cardinality). Depending of the cardinality, we well create accessor methods to access a list or to a single type.

Figure 7: Ontology class Fish

We will now take a closer look at the contents of these files. We will start with the Boss agent file, although for brevity we omit the code that is common to every agent (takeDown method, JADE imports...). See Figure 4 and also the comments between the generated code fragments. Drilling down further we examine in more detail the behaviour Discuss, where again, for brevity, we have omitted the common code – see Figure 5 and the interpolated comments.

Depending on the JADE behaviour type we are inheriting, we must implement one or more additional methods. As this class inherits from CompositeBehaviour, me must implement the methods shown in Figure 5.

At present, we only create skeletons of the behaviour, depending on each behaviour type, but we shortly will complete the code for message passing between agents, extracted from the illocutions given in the institution description and then the bodies of these methods will also be created automatically.

Next, it is the turn of the ontology file, the details of which along with some commentary appear in Figure 6 and finally we have one of the classes of the ontology in Figure 7.

3. CONCLUSIONS AND FUTURE WORK

We have described the translation into Java (JADE) of the specification of aspects of an institution description written in DAML+OIL, noting at the same time various generic issues with respect to translation from DAML+OIL to Java. Because of the very active and changing nature of this area at the moment, a number of issues remain open, in particular, we expect very soon to move the IDL from DAML+OIL to OWL, more or less as soon as the support in the Jena API is released. We have also listed a number of aspects of DAML+OIL that could be, but are not translated yet, simply for lack of time, such as facets, anonymous classes or multiple inheritance. What we have presented here is the translation of the ontological aspects of electronic organizations: other aspects, such as the performative structure, scenes, transitions and conversation graphs are being finalized. Indeed, the completion of behaviour generation for the JADE platform is intended to be fully working before summer 2003, for which we will generate automatically the code of message passing between agents from the information extracted from the illocutions.

For the longer term, we are considering how we might support multiple target agent platforms. A new GUI tool is also under development which could be used to front end this, generating the OWL representation of the institution, which combined with schema verification tools opens up a path to round-trip engineering of institutional specifications.

4. ACKNOWLEDGEMENTS

Daniel Jiménez Pastor is a student of Computer Science Engineering at Facultat d'Informàtica de Barcelona, Universitat Politècnica de Catalunya (Spain). His work has been partially supported by an Agenticities scholarship while an Erasmus student at the Department of Computer Science, University of Bath (United Kingdom).

5. REFERENCES

[1] AgentLink – European Network of Excellence for Agent-Based Computing. `http://www.agentlink.org`, March 2003.

[2] F. Bellifemine, A. Poggi, and G. Rimassa. JADE — A FIPA-compliant agent framework. In *Proceedings of the 4th International Conference on the Practical Applications of Agents and Multi-Agent Systems (PAAM-99)*, pages 97–108, London, UK, 1999. The Practical Application Company Ltd.

[3] DAML+OIL – DARPA Agent Markup Language + Ontology Inference Layer. `http://www.w3.org/TR/daml+oil-reference`, March 2003.

[4] M. Esteva, D. de la Cruz, and C. Sierra. Islander an electronic institutions editor. In *Proceedings of The First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2002)*, 2002. to appear.

[5] M. Esteva, J. Padget, and C. Sierra. Formalizing a language for institutions and norms. In J.-J. Meyer and M. Tambe, editors, *Intelligent Agents VIII*, volume 2333 of *Lecture Notes in Artificial Intelligence*, pages 348–366. Springer Verlag, 2001. ISBN 3-540-43858-0.

[6] FIPA – The Foundation for Intelligent Physical Agents. `http://www.fipa.org`, March 2003.

[7] Islander, a graphical editor for institution specifications (software package). `http://e-institutor.iiia.csic.es/e-institutor/software/islander.html` (April 2002).

[8] Java agent development environment. `http://jade.cselt.it`, March 2003.

[9] Jalopy. `http://jalopy.sourceforge.net`, March 2003.

[10] Javier Vázquez-Salceda. *The role of Norms and Electronic Institutionsin Multi-Agent Systems applied to complex domains. The* HARMON*IA framework*. PhD thesis, Universitat Politècnica de Catalunya, 2003.

[11] Jena Semantic Web toolkit. `http://www.hpl.hp.com/semweb`, March 2003.

[12] P. Noriega. *Agent mediated auctions: The Fishmarket Metaphor*. PhD thesis, Universitat Autonoma de Barcelona, 1997.

[13] D. C. North. *Institutions, Institutional Change and Economic Performance*. Cambridge University Press, 1991.

[14] J. Padget. Modelling simple market structures in process algebras with locations. In L. Moreau, editor, *AISB'01 Symposium on Software Mobility and Adaptive Behaviour*, pages 1–9. The Society for the Study of Artificial Intelligence and the Simulation of Behaviour, AISB, 2001. ISBN 1 902956 22 1.

[15] A. Poggi, F. Bergenti, and F. Bellifemine. An ontology description language for FIPA agent systems. Technical Report DII-CE-TR001-99, University of Parma, 1999.

[16] Protege, an editor for ontologies (software package). `http://protege.stanford.edu/` (April 2002).

[17] Bean generator plug/in for protégé. `http://gaper.wi.psy.uva.nl/beangenerator`, March 2003.

[18] J.-A. Rodríguez. *On the Design and Construction of Agent-mediated Institutions*. PhD thesis, Universitat Autonoma de Barcelona, July 2001.

[19] J.-A. Rodríguez, P. Noriega, C. Sierra, and J. Padget. FM96.5 A Java-based Electronic Auction House. In *Proceedings of 2nd Conference on Practical Applications of Intelligent Agents and MultiAgent Technology (PAAM'97)*, pages 207–224, London, UK, Apr. 1997. ISBN 0-9525554-6-8.

[20] R. Sandhu, D. Ferraiolo, and R. Kuhn. The NIST model for role-based access control: Towards a unified standard. In *Proceedings of the 5th ACM Workshop on Role-Based Access Control (RBAC-00)*, pages 47–64, N.Y., July 26–27 2000. ACM Press.

[21] J. Vázquez-Salceda and F. Dignum. Modelling electronic organizations. accepted for the The 3rd International/Central and Eastern European Conference on Multi-Agent Systems -CEEMAS'03-, Prague, Czech Republic, June 2003, June 2003.

[22] O. Vickers and J. Padget. Skeletal JADE Components for the Construction of Institutions. In J. Padget, D. Parkes, N. Sadeh, O. Shehory, and W. Walsh, editors, *Agent Mediated Electronic Commerce IV*, volume 2531 of *Lecture Notes in Artificial Intelligence*, pages 174–192. Springer Verlag, December 2002.

[23] Word Wide Web Consortium (W3C). OWL – Web Ontology Language. `http://www.w3.org/TR/owl-ref`, March 2003.

[24] World Wide Web Consortium (W3C). Semantic web (daml+oil and owl working drafts). `http://www.w3.org/2001/sw`, March 2003.

Location-Mediated Service Coordination in Ubiquitous Computing

Akio Sashima
CARC, AIST / CREST, JST
2-41-6 Aomi, Koto-ku, Tokyo,
135-0064, Japan
+81-3-3599-8227
sashima@carc.aist.go.jp

Noriaki Izumi
CARC, AIST / CREST, JST
2-41-6 Aomi, Koto-ku, Tokyo,
135-0064, Japan
+81-3-3599-8297
niz@ni.aist.go.jp

Koichi Kurumatani
CARC, AIST / CREST, JST
2-41-6 Aomi, Koto-ku, Tokyo,
135-0064, Japan
+81-3-3599-8223
kurumatani@socsys.org

ABSTRACT
We propose location-mediated service coordination in ubiquitous computing. In the coordination, middle agents determine best-matched services for a user by considering the user's location based on location-ontology for ubiquitous computing. Introducing such location-aware middle agents and location-ontology into ubiquitous computing, we can extend application areas of software agents from the Internet to the real world. In this paper, we first illustrate the idea of location-aware middle agents and location-ontology. Second, we describe a multiagent architecture, called CONSORTS, as an implementation of the agents. In order to bridge the gap between device-oriented physical information in the real world and web-based conceptual information in the digital world, CONSORTS agents can translate sensor-based raw representation of the locations into conceptual one. Finally, we describe two applications of CONSORTS, an intelligent information assist system at a museum and wireless-LAN based location system.

Keywords
Web Agents, Semantic Web, Ubiquitous Computing, Service Coordination

1. INTRODUCTION
Two intelligent service frameworks, Ubiquitous (or Pervasive) computing [25] and Semantic Web[2], received much interest in both research and application in the past years. In order to improve the current web-based service frameworks, the Ubiquitous computing handles device-oriented physical information; the Semantic Web handles ontological information.

Ubiquitous computing enables computers to be aware of the context of users in the real world. The ubiquitous computing environment has numerous sensor devices and computers connecting to the networks. In the framework, the computers manage not only explicit information of the user interaction but also physical information from sensors in the real world. For example, the intelligent guide systems in a museum should manage both location and commands of the users in order to interactively offer information about the exhibitions there.

On the other hand, the Semantic Web enables computers to read the web contents by adding meta-data to the documents. The meta-data are developed on XML. One of such meta-data frameworks is RDF (Resource Description Framework) [18] and RDFS (RDF Schema) [19]. In this framework, the computers manage not only information explicitly described in the content but also ontological information of the content, which are shared with the computers on the Internet. For example, the computers handle the meta-data and the domain ontologies to understand the contents meanings, such as identifying "my father's father" with "my grand father" in a document.

Although there are many researches involved in the Semantic Web and the Ubiquitous computing, still few researches [8] [20] try to bridge the gap between device-oriented physical information in the Ubiquitous computing and web-based ontological information in the Semantic Web. In order to realize intelligent information services for our daily activities (e.g. shopping assistants, museum guide, etc.), we need a service framework that can seamlessly manage both of the information because our activities are comprehensible within the physical and ontological context.

For such seamless service framework, we have been developing a multiagent architecture named CONSORTS (Architecture for cognitive resource management with physically-grounding agents) [5]. In the architecture, middle agents, *location-aware middle agents,* can handle physical and conceptual information of environments, and determine the best-matched service according to the user's location. Bridging the gap between device-oriented physical information in the real world and web-based conceptual information in the digital world, CONSORTS agents can translate sensor-based raw representation of the locations into conceptual one.

In this paper, we describe a framework of service coordination mechanism in Ubiquitous computing. We first illustrate the idea of location-aware middle agents and location-ontology. Second, we describe an implementation of CONSORTS architecture. Finally, we describe two applications of CONSORTS, an intelligent information assist system at a museum and wireless-LAN based location system.

2. LOCATION-MEDIATED SERVICE COORDINATION
Although the ubiquitous computing is a promising framework of the intelligent services, it is still premature and has many problems. One of the problems is about coordination among

services. In a ubiquitous computing environment, a lot of services are usually co-located (e.g. navigation, guide, advertising, information retrieval, controlling devices, etc.). When numerous services can be simultaneously provided to a user, a service coordination is required to provide proper services then and there. How can we realize such a service coordination in ubiquitous computing?

One possible answer is to introduce middle agents [21]. Middle agents are matchmaker agents that match service agents with service requester agents. The middle agents mediate the communication between the agents to achieve the good performance on the Internet. For example, Paolucci et al. [17] show the potential of the middle agents by the orchestration of Web Services [24].

This middle agents work efficiently on the Internet, but it is not sufficient for the middle agents in the Ubiquitous Computing. Most researches focus on the problem how the software agents properly serve the user who operates a computer terminal connected to the Internet [13][16]. However, the middle agents in the Ubiquitous Computing must handle the information of the location where service providers and service requesters communicate with each other. If a service is semantically matched to a request but the service resource is far removed from the requesting user, the service should not be provided in the context of Ubiquitous computing; the nearer service resource to the user is often better than any other services.

2.1 Location-Aware Middle Agents

In order to realize the service coordination suitable for ubiquitous computing, we propose *location-mediated service coordination*, and introduce new type of agent, *spatio-temporal reasoner*. The spatio-temporal reasoner manages sensor-based raw information and conceptual information of the environment, such as physical locations of objects, and reasons of their spatial relations1. Using the spatio-temporal reasoner, we extend the conventional middle agent framework to realize the location-mediated coordination in the Ubiquitous computing.

The location-mediated coordination process consists of following agents:

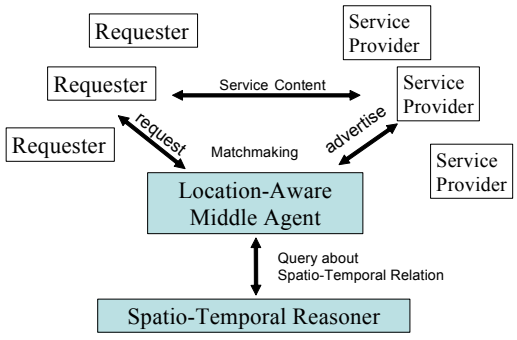

Figure 1. Location-mediated service coordination

[1] To track the object's location, we assume using sensor devices (e.g. RF-ID tags, GPS, etc.) and their information are available.

- **Service Provider Agent**: the agent that provides some types of service, such as problem solving, mediating Semantic Web services;
- **Service Requester Agent**: the agent that needs provider agents to perform some services for them;
- **Location-Aware Middle Agent**: the agent that helps locate other agents along with Spatio-Temporal Reasoner;
- **Spatio-Temporal Reasoner**: the agent that manages physical locations of Service Providers and Service Requesters, and reason about their spatial relation in cognitive way.

The coordination is as follows:

1) provider agents advertise their capabilities and physical service areas to middle agents;
2) middle agents store these advertisements, and inform their service areas to spatio-temporal reasoners;
3) spatio-temporal reasoners store these service areas;
4) a requester asks some middle agents whether they know providers with desired capabilities;
5) the middle agents ask some spatio-temporal reasoners whether the stored service areas cover the requester's location or not, and get the results, a subset of the stored service areas;
6) taking account of the results, the middle agent matches the request with the stored advertisements, and returns the result, a subset of the stored advertisements (see Figure 1.).

In this framework, the descriptions of location are separately managed in the spatio-temporal reasoners, so service contents modification is not required even if the service areas are changed. For example, the content for an information service in a museum can be used in exhibitions in other museums, by only adjusting the location information to the museum.

Additionally, this process is a natural extension of matching process proposed by Sycara et al.[21]. By integrating above agents with ordinal web agents on the Internet, we can extend application areas of Web agents (and Web Services) to ubiquitous computing environment.

3. MODELING SPACE IN COGNITIVE WAY

3.1 Conceptual Representation of Locations

To realize the location-mediated service coordination, the agents must share representations of locations (e.g. service area, the requester's location, etc.). Moreover, the representation must be human-friendly, conceptual ones. For example, to inform the user's location to others, the agents should understand "the user was in a museum on the morning", rather than "the user was at Longitude: 140.38.54 E, Latitude: 35.77.44 N at Mon Jan 13 07:47:06 2003 JST". Such human understanding of spatial

concept is generally called mereological thinking [23], reasoning about part-of relation. To realize this kind of spatial reasoning as a basic function of our agent architecture, especially of Spatio-temporal reasoner, we model ubiquitous computing environment with a mereological, tree representation. The representation consists of *part-of* relations among 4-dimentional spatio-temporal regions in the real world.

3.1.1 Location and Region

First, we formularized locations of objects with the spatio-temporal regions using the formalization by Bittner [3]. The formalization is as follows.

The symbol o ($o \in O$) denotes an object, and the domain of objects, O. The symbol x ($x \in X$) denotes a spatio-temporal region, and the domain of the regions, X. The domain of the regions consists of 3-dimentional (spatial) regions, x^s, and 1-dimenational (temporal) region x^t. An object o is exactly located at the region x is formalized as follows;

$$\forall o \in O : \exists x^s \in X : L_\tau(o, x^s)$$

where the symbol τ denotes an instance of time. Weaker definition of object's location is as follows:

$$L(o,x^t,x^s) \equiv$$
$$\exists x' : x' \le x^t \wedge \forall \tau \in x' : \exists x : L_\tau(o,x) \wedge x \le x^s$$

where the symbol $\le$ denotes the part-of relation of the domains.

Hence, *L (John, his room, this morning)* means "*John was in his room in this morning*".

3.1.2 Tree Representation

We formularize the ubiquitous computing environments as a tree-structure of spatio-temporal regions based on Bittner [3]. A tree structure of spatio-temporal regions is denoted by G : $G = (R, \subseteq)$ where $R : (r \in R)$ denotes a set of spatio-temporal regions, $\subseteq$ denotes a part-of relation between two regions. The regions in G have following natures.

$$1)\ \exists r_i : r_i \subseteq r_i$$
$$2)\ \exists r_i, r_j : r_i \subseteq r_j \wedge r_j \subseteq r_i \rightarrow r_j = r_i$$
$$3)\ \exists r_i, r_j, r_k : r_i \subseteq r_j \wedge r_j \subseteq r_k \rightarrow r_i \subseteq r_k$$

where r_i, r_j, r_k is a region.

3.2 Mereological Reasoning

Now, we can infer and query the object's location using this notation. For example, given two statements: "John was in his room in this morning", and "His room is part of this building", we can infer "John was in this building in this morning", we can infer the "Jhon's location" as follows.

$$L(John, his\ room, this\ morning) \wedge$$
$$his\ room \subseteq this\ building$$
$$\rightarrow L(John, this\ building, this\ morning)$$

Similarly, we can define the users that can receive the service in ubiquitous computing environment, denoted by C, as:

$$C = \bigcup \{\exists Person : L(Person, anArea, aPeriod)\}$$

By using this kind of logical inference, Spatio-Temporal Reasoner can reason about their spatio-temporal relation. Replying results, a subset of the stored service areas, denoted by SR, Spatio-Temporal Reasoner can reason as

$$SR = \bigcup \{\exists r_{service} : L(person, r_{service}, aPeriod)\}$$

where $r_{service} : r_{service} \in R_{service}$ denotes a service area of a set of service areas in the environment.

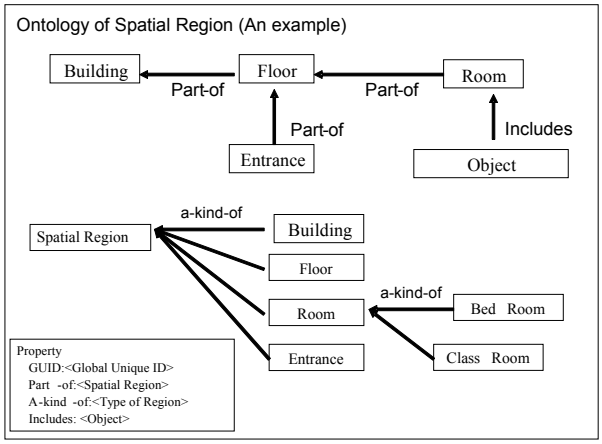

Figure 2 A part of location ontology

3.3 Location Ontology

Based on the above formalization, we define a location ontology for the ubiquitous computing. Figure 2 shows the location ontology for describing spatial region. We define similar type of ontology for describing temporal region. We define the relation of spatial regions, and their property in the ontology. Part-of relation in the figure is the relation denoted by $\subseteq$ in before section. The agents can describe object's location, and reason about spatial-relation based on the ontology.

3.4 Spatio-Temporal Reasoner

Spatio-Temporal Reasoner manages "spatio-temporal inference engine" and "spatial information repository". The repository manages the device-oriented physical information of the ubiquitous computing environments, such as location of users and

surrounding sensor devices. Location data are constantly updated, given by "device wrapper agents".

Spatio-temporal reasoner also manages conceptual representation of the environment, tree-structure (G in Section 3.1.2). The middle agent and various service agents, such as navigation planning, searching the nearest device to the user, use the tree-representation of environment, and do not use the device-oriented raw representation. It means that the reasoner hides such complex representations from other agents and humans.

Hence, one of important function of spatio-temporal reasoner is bridging the gap between sensor-based information in the real world and cognitively comprehensive annotated information in the digital world (see Figure 3.). For the translation between two representations, we define static mapping functions beforehand.

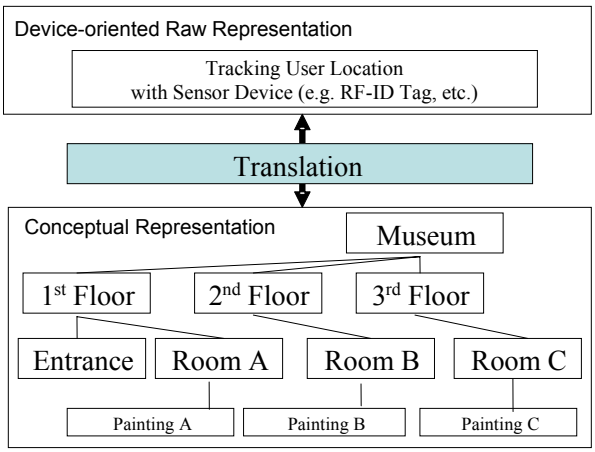

Figure 3 Data representations of spatio-temporal reasoner

4. LOCATION-AWARE MIDDLE AGENT

Location-aware middle agent is an agent that help locate other agents along with Spatio-Temporal Reasoner The middle agent orchestrates the various services of agents that manage sensor information of the physical environment and ontological information of the web services. Although the middle agents control the matchmaking process, the spatio-temporal reasoning is outsourced to Spatio-Temporal Reasoner.

4.1 Matchmaking

The location-aware middle agent matches the request against the stored advertisements with taking account of the results from Spatio-Temporal Reasoner. In our prototype system, we tentatively define the matchmaking process as following expression.

$$S_i = argMax_{i=1,2..n}\{u(S_i)\}$$
$$u(S_i) = \lambda f(SC_i) + (1.0 - \lambda) g(SR_i)$$
$$(0.0 \le \lambda \le 1.0)$$

Where S_i denotes one of available services in current situation. $u(S_i)$ represents a utility function of the service S_i. λ denotes a constant value for a normalized weight to matching degree of contents for utility function $u(S_i)$. $f(SC_i)$ represents a matching degree of a service content SC_i. $g(SR_i)$ represents a matching degree of service area SR_i. The middle agent selects the most suitable service which has the maximum utility value among the services. This coordination scheme is a basic function in We can implement complex services, such as service composition, as facilities of the middle agents.

Because the matchmaking process that we proposed is an extension of the ordinal matchmaking process among agents on the Internet, the process model of DAML-S[7] can be apply to the description of service content. Therefore, we omit the descriptions for service contents, and their matchmaking process $f(SC_i)$ in this paper. We concentrate to describe the location based matchmaking process $g(SR_i)$.

4.2 Location-based Matchmaking

Location-based matchmaking process has two methods: the method that calculates matching degree by *part-of* relations, and by *a-kind-of* relations. Each of two functions calculates the matching degrees between services and a user.

4.2.1 Matching Degrees by Part-of Relations

Each of services has a service area that corresponds to a node of the tree representation of the environment. This matching method calculates the matching degrees based on part-of relations of spatio-temporal regions, which include the service areas, or the requester are located at. At first, the agent filters the service areas where the requester is located, then, calculates matching degrees of each service areas.

The matching degrees of the service areas are discrete values corresponding to the path lengths from root node to the nodes of the service areas on a *part-of* relation tree. In the tree representation in Figure 2, if the user is located at "Entrance", the matching degree is 2. In other words matching degree of narrower region has high value (e.g. g(*Room A*) >g(*1st Floor*) in Figure 3).

4.2.2 Matching Degrees by a-kind-of Relations

This matching method enables the requester to access the information generally related to facilities of the place. For example, when a user is located at a library, the user would like to access sorts of book guidance services. This service is not related to the physical location of the library but related to the cognitive facilities of the library. This kind of location-mediated coordination is suitable for assist services in a lot of places where human activities take place (e.g. station, library, museum, etc.).

The matching degrees are calculated based on *a-kind-of* relations. When the service providers and the requester are located at the different place, a-kind-of relations are used. At first, the agent looks at the a-kind-of property of the region where requester is located, and pick up the services that have the same a-kind-of property. Then, the agent calculates the matching degree of each service provider.

The matching degrees of the service areas are discrete values corresponding to the path lengths from root node to the nodes of the service areas on a *a-kind-of* relation tree. In the tree representation in Figure 2**,** if the user is located at "Bedroom", the matching degree is 2. Matching degree of narrower regions have high values (e.g. g(*Bedroom*) >g(*Room*) in Figure 2).So the user receives the information services related to bedrooms rather than rooms.

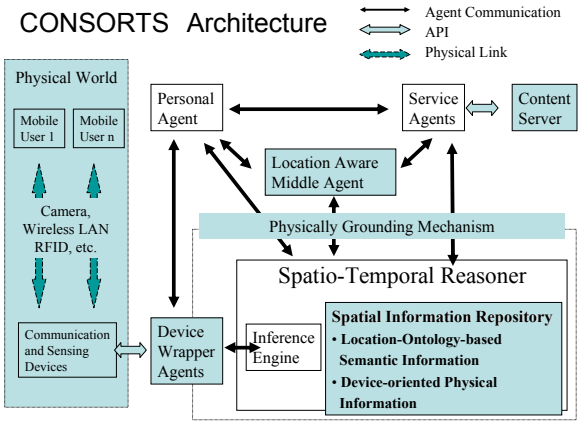

Figure 4 Outline of CONSORTS architecture

5. IMPLEMENTATION

In order to realize human-centered intelligent information services in ubiquitous computing, we have been developing CONSORTS (Architecture for cognitive resource management with physically-grounding agents), an agent-based software architecture that mediates the real world and a digital world, for assisting human in accessing intelligent information services. In this section, we describe the CONSORTS architecture as an implementation of the location-aware middle agents, and confirm the functionality of the middle agents in context-aware information assist service in a museum.

5.1 CONSORTS Architecture

The key concepts of CONSORTS are "physically-grounding" and "cognitive resources"[15]. By using sensory information brought by ubiquitous environment, agents have a grounding in physical world, and manage physical resources (especially spatio-temporal resources) in a cognitive way, i.e., they can recognize, reorganize, and operate raw physical resources as conceptual resources.

Figure 4 shows an outline of CONSORTS Architecture, which consists of following type of agents.

- ***Services Agents*** (Service Provider Agent) provide services, such as tour guide, navigation, and information presentation. Services agents realize the flexible service by using semantic information through the content servers. The content servers are resources on the Internet, especially Semantic Web and Web Services.
- ***Personal Agents*** (Service Requester Agent) request service behalf of a user. The agents communicate with Device wrapper agents, and manage them as user interface devices.
- ***Location-aware Middle Agents*** manage to interaction to mediate between agents and a user. The agent sends the request to the middle agents, and controls the interaction between service agents and the user. In current version, "Matching by *a-kind-of* Relation" is not implemented.
- ***Spatio-Temporal Reasoners*** manage physical locations of Service Providers and Service Requesters, and reason about their spatial relation in cognitive way. The agents manage both device-oriented physical information in the real world and cognitively comprehensive information in the digital world
- ***Device Wrapper Agents*** hide the diversity of physical device and legacy application. Device Wrapper Agents realize device-independent applications by wrapping the raw information, which is derived from legacy devices and applications, with the standardized agent communication language.

5.2 Context-Aware Information Assist Service in a Museum

Based on the CONSORTS architecture, we have implemented context-aware information assist service in a museum. In this system, the agents are aware of the distance between a user and paintings in the museum. The service system is that when the user approaches a painting, the agents automatically provide information of the painting via the user's portable display device.

In the museum, user can access general museum information based on the procedure of *matching degrees by a-kind-of Relations*. In addition, when a user is located near a painting, the user can receive the information of the painting via the user's portable display device from the agents. This service is based on the procedure of *matching degrees by part-of Relation.* Since, the information of the painting is derived from the Internet resources (e.g. Web services, Search Engine, etc.).For example, if the user needs more information of the painting, the user should push the "tell me" button of the portable device. The CONSORTS agents notice the user's request, make a search query about the painting with the necessary information they've already had, e.g. the painting's name and user's preference, and access the internet search engine, e.g. Google Web Services [10], behalf of the user.

Figure 5 shows a snapshot of the monitor display of the system. You can see a museum map in a main window, two information windows with some pictures, and a message window displaying agent communication. In the map, human icons represent current locations of the users; the yellow zones represent service zones of the museum service agents; the blue lines represent users' trajectories. The information windows correspond to the screen of users' portable devices. In this demonstration program, the users randomly roam in the museum. If a user enters the yellow zone, an information window pops up in the screen to display the picture information.

Currently, users, users' behavior, and a museum in this system are simulated ones. However, we have designed the device wrapper agent to hide the diversity of the devices API, we can integrate the physical devices into the system on the CONSORTS.

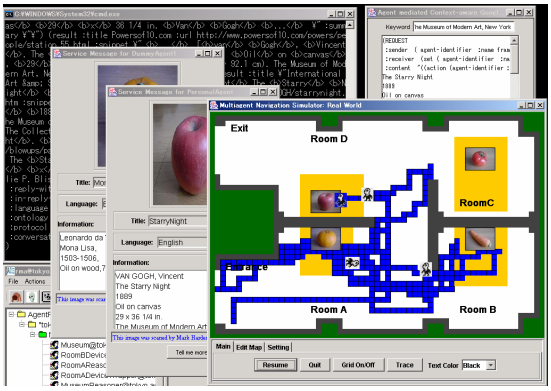

Figure 5 Snapshot of the assistance system at the museum

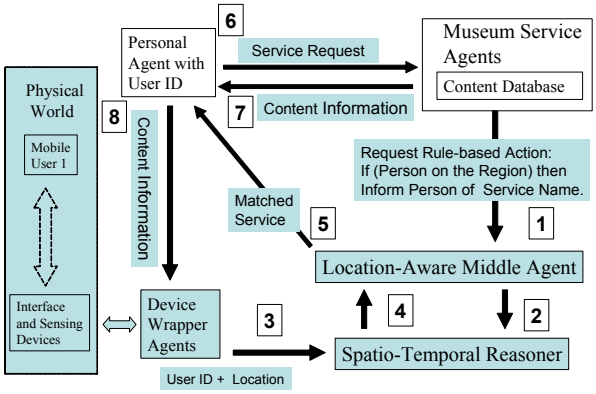

Figure 6 Message flows in the assistance system at the museum

Figure 6 outlines the agent communication in the museum system. First, the service agent requests the middle agent to continuously infer whether users are located on a service region associated with a service name ([1] in Figure 6). The middle agent requests the Spatio-Temporal Reasoner to request to inform when users are located on a service region ([2] in Figure 6). On the other hand, the device wrapper agent informs the spatio-temporal reasoner where a user with unique id is located ([3] in Figure 6). When the spatio-temporal reasoner notice a user is located on a service region, the reasoner informs the middle agent of the user id ([4] in Figure 6). The middle agents tell the personal agents of the user available service name ([5] in Figure 6). Then, the personal agent requests a service based on the service name ([6] [7] in Figure 6). Finally, the personal agent requests the device wrapper agent to show the service content on a user's portable display device ([8] in Figure 6).

Table 1 A message in the assistance system at the museum

```
(REQUEST
 :sender      PersonalAgent
 :receiver    MuseumAgent
 :content     "((action
                (agent-identifier
                          :name MuseumAgent)
               (provide
               (person :name PersonalAgent)
               (service :name Monalisa
                   :provider MuseumAgent)))"
  :language   fipa-sl
  :ontology   location-ontology
)
```

All messages of the agent communication are described with FIPA-ACL (Foundation for Intelligent Physical Agents Agent Communication Language)[9]. The message content is described with FIPA-SL. Table 1 shows a message which represents "Personal Agent request to Museum Service Agent to provide the service named Mona Lisa (showing information about the painting of the Mona Lisa)." This message flows at arrow-line [6] in Figure 6. Although we have already implemented RDF-based descriptions as a content part of the message, we adopt FIPA-SL as the content language for interoperability with other agent platform in this system. We are also planning to adopt DAML families [11][6] in the near future. We have implemented the CONSORTS agents using JADE [12], a software framework to develop the agent system that conforms to FIPA specifications. Using standardized ACL like FIPA-ACL, the system can connect to open agent systems, such as AgentCities [1]. Hence, using the CONSORTS architecture, FIPA-agents can share the physical information and devices in the museum, and open the possibility of new services which use the physical resources over the world.

In this demonstration system, we have shown the potential of the distributed open agent systems that can bridge device-oriented physical information in the real world and web-based conceptual information in the digital world.

5.3 Wireless-LAN based Location System

Based on the CONSORTS architecture, we have implemented Wireless-LAN (IEEE 802.11b) based Location System. In this system, the agents are aware of user location by watching the status of the wireless-LAN stations. The service system is that when the PC or PDA with a user connects with one of the stations, the system detects the connection, and store a MAC address of the network card of the PDA with physical location of the station. Because a MAC addresses is a globally unique, the system can track the location of the card (with the user) globally.

Figure 7 shows an outline of the location system. In the system, Web services, which provide location information like sensor

devices, are integrated with a FIPA-agent platform, and the location information is shared with FIPA-agents.

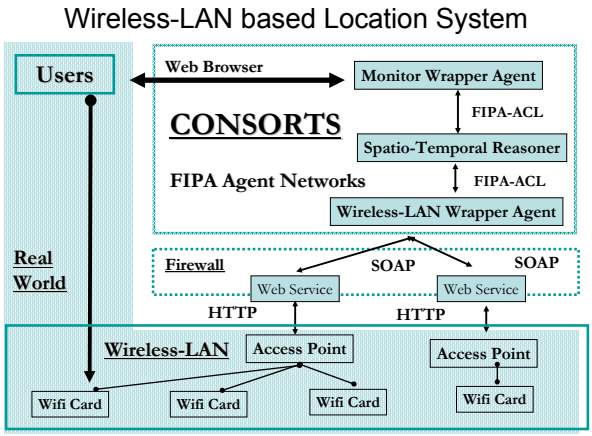

Figure 7 Outline of wireless-LAN based location system

We have experimentally confirmed the functionality with real wireless-LAN stations that located on various cities, (e.g. Barcelona, Tokyo, etc.) in the world. Figure 8 shows a snapshot of the monitor window of the location system in that experiment.

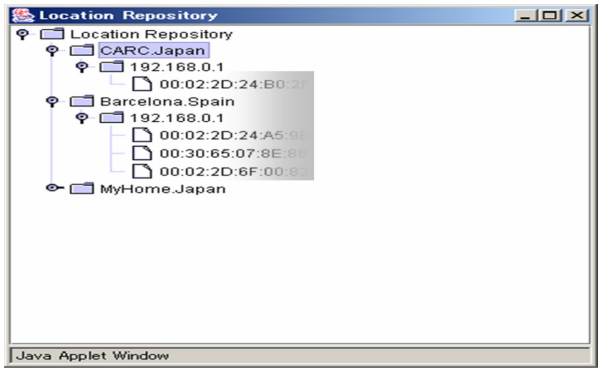

Figure 8 Snapshot of the location repository

In this experiment, we have shown the potential of the distributed open agent systems that can access and unify the real world information covering all over the world.

6. RELATED WORK

CoolAgent [4] is a context-aware software agent system. The agents can share the context information described by RDF that can be an ontology description framework for the Semantic Web. However, CoolAgent system does not provide the design framework of ubiquitous agents, such as location-aware middle agent. In this paper, we formally describe the design pattern of the coordination of ubiquitous agents.

There are some researches for middle agents[21][22][17]. As we described before, the work in this paper is extension of their works in ubiquitous computing. Hence, their results, such as integration of DAML-S, must be useful for our future research.

7. FUTURE WORK

As future work, we are planning to introduce the concept of "Mass User Support"[15]. Existing context-aware applications provide information for a single user. However, if the applications provide some users with the same information at the same time, serious conflict with users' behavior, such as a long queue formed in front of a tourist attraction, may be aroused. In that situation, the total performance to assist users' activities will decrease. Thus, we are planning to realize the service which has facility to resolve the conflict with multi users' requirements and to implicitly modify the providing information for each user in order to maintain the performance of the whole system. We are implementing the facility using market mechanism, called "user intention market."[14]

8. CONCLUSION

In this paper, we described a framework of service coordination mechanism in Ubiquitous computing. We first illustrated the idea of location-aware middle agents and location-ontology. Second, we described an implementation of CONSORTS architecture. Finally, we described two applications of CONSORTS, an intelligent information assist system at a museum and wireless-LAN based location system.

In our demonstration systems, we have shown the potential of the distributed open agents in the real world. In the Semantic Web[2], an "agent" is defined as '*the program that collects information from diverse sources, processes the information and exchanges the results with other programs.*' We have realized such agents in the context of the Ubiquitous computing, and extended application areas of the agents from the Internet to the real world.

9. REFERENCES

[1] AgentCities Web, http://www.agentcities.org/, 2003

[2] Berners-Lee, T., Hendler, J. and Lassila, O., the Semantic Web. *Scientific American*, 2001.

[3] Bittner, T., Reasoning about qualitative spatio-temporal relations at multiple levels of granularity. In F. van Harmelen (ed.): ECAI 2002. *Proceedings of the 15th European Conference on Artificial Intelligence*, IOS Press, Amsterdam, 2002

[4] Chen, H. and Tolia, S., Steps towards creating a context-aware agent system. *TR-HPL-2001-231*, HP Labs, 2001.

[5] CONSORTS Architecture Web, http://consorts.carc.jp/, 2003

[6] DAML+OIL (March 2001) Reference Description, http://www.w3.org/TR/daml+oil-reference, 2002

[7] DAML Service, http://www.daml.org/services/, 2002

[8] Finin, T. and Perich, F., editor. *Proc. of the AAMAS Workshop on Ubiquitous Agents on embedded, wearable, and mobile devices*, 2002

[9] The Foundation for Intelligent Physical Agents (FIPA), http://www.fipa.org/, 2002.

[10] Google Web APIs, http://www.google.com/apis/, 2003

[11] Hendler, J. and McGuinness, D. L., DARPA Agent Markup Language (DAML). *IEEE Intelligent Systems*, 15 (6), 72–73, 2001.

[12] Java Agent DEvelopment Framework (JADE), http://sharon.cselt.it/projects/jade/, 2002

[13] Jennings. N., An agent-based approach for building complex software systems. *Communications of the ACM*, 44 (4): 35-41, 2001.

[14] Kurumatani. K., User Intention Market for Multi-Agent Navigation - An Artificial Intelligent Problem in Engineering and Economic Context; Kurumatani, K., Chen, S.-H.,Ohuchi, A. (eds.), *Proceedings of The AAAI-02 Workshop on Multi-Agent Modeling and Simulation of Economic Systems, Technical Report WS-02-10*, pp.1-4, AAAI Press, 2002.

[15] Kurumatani. K., Social Coordination in Physically-Grounded Agent Architecture, to appear in *Proceedings of Landscape Frontier International Symposium*, 2002.

[16] Maes, P., Agents that Reduce Work and Information Overload. *Communications of the ACM*, 37(7): pp.31-40, ACM Press, July 1994.

[17] Paolucci, M., Kawamura, T., Payne, T. R., and Sycara, K., Semantic Matching of Web Services Capabilities. In *Proceedings of the 1st International Semantic Web Conference (ISWC2002)*, 2002

[18] Resource Description Framework (RDF), http://www.w3.org/RDF/, 2002

[19] RDF Vocabulary Description Language 1.0: RDF Schema, http://www.w3.org/TR/rdf-schema/, 2002

[20] Sashima, A., Kurumatani, K., and Izumi, N., Physically-Grounding Agents in Ubiquitous Computing, *Proc. of Joint Agent Workshop* (JAWS2002) , pp.196- 203, 2002.

[21] Sycara, K., Decker, K., and Williamson, M., Middle-Agents for the Internet, *Proceedings of IJCAI-97*, January 1997.

[22] Sycara, K., Klusch, M., Widoff. S., and Lu, J., Dynamic Service Matchmaking among Agents in Open Information Environments, *SIGMOD Record (ACM Special Interests Group on Management of Data)*, Vol. 28, No. 1, 47-53. , March, 1999

[23] Varzi, A., C. and Casati, R., Parts and Places. The Structures of Spatial Representation, Cambridge, MA, and London: MIT Press, 1999.

[24] Web Services Activity, http://www.w3.org/2002/ws/, 2002.

[25] Weiser, M., The computer for the 21st century. *Scientific American*, 94-104, Sep 1991.

Collaborative Understanding of Distributed Ontologies in a Multiagent Framework: Design and Experiments

Leen-Kiat Soh
Computer Science and Engineering
University of Nebraska
115 Ferguson Hall
Lincoln, NE
(402) 472-6738

lksoh@cse.unl.edu

ABSTRACT

In this paper, we describe our work-in-progress with collaborative understanding of distributed ontologies in a multiagent framework. As reported earlier, the objective of this framework is to improve communication and understanding among the agents while preserving agent autonomy. Each agent maintains a dictionary for its own ontology and a translation table. Our current work has focused on how neighborhood profiling, the translation tables, and query experience influence the collaborative activities among the agents. We have built an infrastructure prototype and conducted a series of comprehensive experiments from the viewpoint of agent executions, query scenarios, and translation credibility values. The specific goals of our analyses are to investigate (a) the learning of useful neighbors for sharing queries, (b) the efficiency of query handling in different real-time scenarios and with different resource constraints (such as the number of threads and available translations), and (c) the effects of different concepts and query demands on collaborative understanding. This paper reports on the results that we have collected so far.

Keywords

Multiagent systems, distributed ontology learning, dynamic profiling

1. INTRODUCTION

In our previous work, we described a distributed ontology learning framework in a multiagent environment [1]. There are two ways that an agent can learn to improve its ontology. First, users can teach them—by supplying a list of words and what the classifying concepts are for that list of words. Second, an agent can learn through its interactions with its neighbors. As a result, each agent learns its own concepts based on its experiences and specialties. When a new concept arrives, the agent needs to incorporate it into its dictionary and its translation table. This is supported by three important components: conceptual learning, translation, and interpretation, with a Dempster-Shafer belief system [2].

Our research focuses on developing and analyzing the operational components of our framework, applied to a document retrieval problem. Each agent interacts with a user who submits queries based on keywords. These keywords are known as concepts in the agents. The goal of this problem is satisfy as many queries as possible and as well as possible. An agent may turn to its neighbors for help. Thus, this collaboration facilitates the distributed ontology learning.

To improve communication, these agents must be able to understand each other. Thus, our research goals are to (1) promote understanding among agents of a community, thus reducing communication costs and inter-agent traffic, (2) improve cooperation among neighbors of a community, thus enhancing the strength (productivity, effectiveness, efficiency) of a neighborhood and supporting the distributed effort of the community, (3) encourage pluralism and decentralization within a multi-agent community—specialization of agents of a community since each agent can rely on its neighbors for tasks not covered by its capabilities, and (4) enable collaborative learning to improve the throughput of the community, the intelligence in communication and task allocation, the self-organization within the community, and integrity of the community.

At the current phase of our research, the objective is to understand how collaborative understanding of distributed ontologies is impacted by operational issues such as queries, the number of communication threads, the variability within the translation tables and so on. Therefore, in this paper, we focus on the operational design of our infrastructure and the investigations on how neighborhood profiling, translation tables, and query experience influence the relationship among collaborative agents. Our experiments are aimed at studying (a) the learning of useful neighbors for sharing queries, (b) the efficiency of query handling in different real-time scenarios and with different resource constraints, and (c) the effects of different ontological concepts and query demands on collaborative understanding. In Section 2, we briefly outline the methodology of our framework. Then we describe our implementation. Subsequently, we discuss our experiments and results. Finally, we conclude.

2. FRAMEWORK

In our framework, the multiagent system is one in which agents can exchange queries and messages to learn about each other's ontology. To improve the communication and collaboration efficiency, agents determine whether some translation is worth learning, which neighbors to communicate to, how to handle and distribute queries, and how to plan for agent activities. The framework consists of two sets of components. The operational components allow the agents to work together in a multiagent system. The ontological components allow the agents to communicate and

understand each other. In this paper, we focus on the operational components of our framework.

When an agent receives a query, it checks the query against its ontology knowledge base. A query comes with a concept name and the number of documents or links desired. If the agent cannot satisfy the query, it will contact its neighbors. If the agent recognizes the concept name but does not have enough documents or links to fulfill the requirement, then it will approach its neighbors to obtain more links. If the agent does not recognize the concept name, then it passes the query to its neighbors. Every agent is equipped with *N* number of *negotiation* threads. For each contact, an agent has to activate one of these threads. So, if an agent does not have available inactive negotiation threads, it will not be able to collaborate with other agents. Hence, even if the agents do understand each other's ontologies, it is possible that due to the query frequency and the resource constraints, the agents may not be able to utilize that understanding to help solve a query problem. Since this collaborative activity requires Please refer to [3] and [1] for details on our original design of operational and ontological components, respectively.

2.1. Operational Components

There are three important operational components: query processing, action planning, and query composition. Note that in our framework, an agent sends out a query to its neighbor when it needs to find some additional links for that some classifying concepts. Agents are required to compose queries as well as they also need to relay or distribute queries to other agents by modifying the queries in their own words. Finally, for the system to be effective, the query distribution and the ontology learning behavior are supported by an action planning component that makes decision based on the agent's environment such as message traffic and neighborhood profile.

2.2. Ontological Components

There are three important ontological components in our framework: conceptual learning, translation, and interpretation. We represent an ontology item as a vector. Each vector consists of the classifying concept and then a list of words describing that concept. A concept may have many different supporting documents. Different concepts may be used to classify the same list of words, resulting in different supporting items. Moreover, for each concept, the agent also learns the description vector, combining all relevant experience cases together. This allows the system to incrementally learn and evolve existing ontologies. Currently, we are still implementing the ontological components and thus will not report further on them in this paper.

3. METHODOLOGY & DESIGN

An agent performs two types of learning. It learns incrementally, refining its concepts whenever there is a new submission. It also learns collaboratively, refining its translation table whenever there is a query that prompts the agent to ask for help from its neighbors. Figure 1 depicts the current status of operational components of an agent in our framework.

As shown in Figure 1, there are nine important modules:

(1) Interface: This module interacts with the user to obtain queries and to provide queried results. Currently, we have (simulated) software users that automatically generate timed queries for our experiments. Each software user submits its queries through a socket connection with the interface.

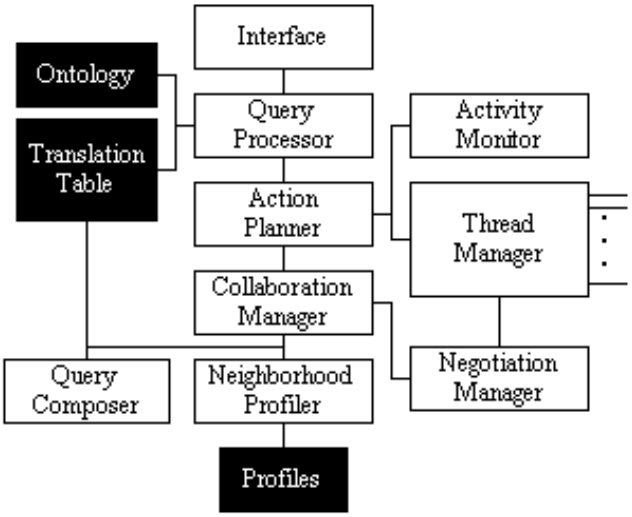

Figure 1 The current design of the operational components of an agent in our framework.

(2) Query Processor: This module receives a query from the Interface module and processes it. It first checks the agent's ontology base. If the query matches one of the concepts in the ontology, the module retrieves the number of links available. If the query does not find a match in the ontology, the module examines its translation table. If there are available translations, that means a collaboration is possible.

(3) Action Planner: This module serves as the main reasoning component of the agent: (a) If the number of internal links satisfies the query, then the action planner simply provides those links to through the Interface module to the user; (b) otherwise, if the agent understands the query and finds available translations, it initiates its collaborative activities; (c) if the agent does not understand the query, it will relay the query to another agent; and (d) finally, if there are no available translations, the link retrieval process stops and the agent reports back to the user. Whether a collaboration is feasible depends on the current status of the agent, as recorded by the Activity Monitor and Thread Manager modules. If the agent does not have enough resources for a collaboration, the link retrieval process terminates.

(4) Collaboration Manager: When the action planner calls for a collaboration, this module takes over. The objective of this module is to form an appropriate group of neighboring agents to approach and distribute the query demands (link allocations) accordingly among them. To design such a collaboration plan, this module relies on the Neighborhood Profiler module, and the translation table. Each neighbor is given a *utility* measure based on the translation credibility value, the past relationship and the current relationship. Note that in our original thesis [1], each translation has a credibility value: two concepts are similar to only a certain degree. The past relationship is the viewpoint of the agent of its neighbor of their interactions (or negotiations in our framework) monitored and stored by the Profiler module. The current relationship is captured by the Activity Monitor module to indicate whether the agent is currently engaged in any negotiations with the particular neighbor. A neighbor has a high utility if the translation credibility of the query in question is high, if the

lation credibility of the query in question is high, if the past relationship is strong, and if there is not any current interaction. The collaboration manager ranks these neighbors based on the utility measure and then assigns the query demands accordingly, with the help of the Query Composer. The manager assigns more links to neighbors with higher utility proportionally to maximize the chance of retrieval success. It also collects the negotiation results, sorts the received links based on the credibility, and filters out low-credibility links when it has more links than desired.

(5) Query Composer: Based on the allocation of query demands, this modules composes a specific query for each neighbor to be approached. As previously mentioned, each query is associated with a link requirement that specifies the number of links desired. A query will also include the name of the originator and a time stamp when it is first generated. If the query is based on a translation, then the translated concept name is used. If the agent does not recognize a concept and needs to relay a query it has received to a neighbor, it simply uses the queried concept directly.

(6) Neighborhood Profiler: The design of this module is based on our work in coalition formation. As we will later in Section 4, we keep track of the past relationship between the agent and each neighbor. The relationship is a composition of four basic numbers: *_numHelp* (the number of times the agent provides help to the neighbor), *_numSuccess* (the number of times the agent successfully solicits help from the neighbor), *_numRequestFrom* (the number of times the agent receives a request from the neighbor), and *_numRequestTo* (the number of times the agent initiates a request to the neighbor) [4]. Based on these numbers, we can derive helpfulness, usefulness, importance, and reliance of each neighbor, from the viewpoint of the agent.

(7) Activity Monitor: This modules keeps track of the activities in a *job vector*—whether the agent is processing a query on its own, or negotiating with other neighbors for more links, or entertaining a request by a neighbor. Each *job* is described with a list of attributes such as the originator, the executor, the task description, the current status, and so on.

(8) Thread Manager: This module housekeeps the threads of the agent. It is a low-level module that activates the threads, updates and monitors the thread activity.

(9) Negotiation Manager: This module manages the negotiation tasks. In our current design, the interaction between two agents does not involve negotiations as the two simply exchange information. However, our long-term plan views negotiation as an important part of ontology interpretation in a distributed environment. Negotiations that are too time consuming, stagnant, or no longer useful will be modified or aborted; negotiations that are successful will be learned; and so on. We will adapt our previous work in reflective negotiations [5] to distributed ontology in this framework.

Together with these nine operational components are three dynamic knowledge or data bases: ontology, translation table, and profiles. The profiles keep track of the relationships between the agent and its neighbors, updating the neighborhood parameters. The ontology is a *dictionary* listing the concepts that the agent knows. Each concept has a list of supporting documents or links. The translation table consists of translations between each concept that the agent knows and its neighbors. Each translation is accompanied with a credibility value. Some neighbors may not have concepts that are similar to a concept that the agent knows and the credibility value for those entries in the translation table is NIL. Table 1 shows an example of a translation table for agent A1. In the example, A1 has four neighbors. It knows of concepts such as "basketball" and "car". For "basketball", it is similar to N1's "NBA" with a credibility of 2.1, N2's "Bball" with a credibility of 1.0, and N4's "Basketball" with a credibility of 3.4. However, it does not have a translation for "basketball" between itself and N3.

Concepts	**N1**	**N2**	**N3**	**N4**
basketball	*NBA* 2.1	*Bball* 1.0	*NIL*	*Basketball* 3.4
car	*NIL*	*Auto* 2.1	*Car* 1.0	*Move* 1.0
...				

Table 1 A translation table example.

4. IMPLEMENTATION

We have implemented all the nine modules of our agent as depicted in Figure 1 in C++. Each agent receives its user queries from a software user through a socket connection, and communicates with other agents through a central relay server module through socket connections as well. Each agent generates and maintains its neighborhood profile during runtime dynamically.

For our experiments, each agent is equipped with a translation table right from the start. Note that in our original distributed ontology framework [1], the entries in a translation table is learned over time based on the experience of each agent. In this paper, we focus on the operational design of collaborative understanding of distributed ontologies and assume that each agent has a translation table to begin with.

In addition, each agent is equipped with an ontology database. This database lists all the concept terms that an agent knows. For each concept, there is a list of links (or documents) that are examples that illustrate the concept. Indeed, when interpreting two concepts, we simply compare the similarities of the two lists of links supporting the two concepts. Currently, we are building this interpretation module.

5. DISCUSSION OF RESULTS

We have performed a comprehensive set of experiments. In this Section, we will describe our experimental setup and then discuss the results.

5.1. Experimental Setup

Here is the setup of our experiments:

There are five agents supporting a software user each. All agents are neighbors and can communicate among themselves. All five agents and their threads are run on the same CPU.

Every agent has a unique set of nine concepts in its ontology. Each concept has five supporting links.

Each agent has a translation table where each cell of the table indicates the translation between a local concept and a foreign concept in a neighbor and the translation's credibility value. If a translation is not available, we use the symbol NIL.

Each software user has a query configuration file. Thus, instead of manually submitting these queries, the software user simply reads them from the file and sends them to the corresponding agent. For each query in a configuration file there are (a) a cycle number, (b)

the queried concept name, and (c) the number of link desired. The cycle number indicates when the query will be submitted to the agent. (A cycle's time varies as this measures a loop of activities of an agent.) Each configuration file has about 300 cycles, and two batches of exactly the same query scenarios. We want to investigate whether the agents are able to improve in their response time in the second batch after learning how to form collaborations better through neighborhood profiling.

In the first batch of query scenarios,

(1) Cycles 0-10: Every user queries about all different concepts its agent has in the ontology. Each agent is also able to satisfy the query demand on its own. During this segment, each agent does not need to collaborate. All queries across the users are submitted at the same cycles.

(2) Cycles 11-40: Every user queries about all different concepts its agent has in the ontology. However, each agent is not able to fulfill all queries on its own. During this segment, each agent needs to collaborate. All queries across the users are submitted in a staggered manner. User 1 submits all its nine queries first; user 2 submits its queries after 3 cycles; and so on.

(3) Cycles 41-70: Every user queries about all different concepts its agent has in the ontology and each agent is not able to satisfy the queries on its own. Also, the number of links desired for every query is twice that in the second segment. Extensive collaborations are needed. Queries are also staggered in this segment.

(4) Cycles 71-80: Every user queries about different concepts its agent does *not* have in its ontology. This forces the agent to relay the queries to other neighboring agents. Queries are packed and not staggered in this segment.

(5) Cycles 81-110: The setup of this segment is similar to that during cycles 11-40, but with concepts that each agent does not have in its ontology. Queries are staggered.

(6) Cycles 111-120: During this segment, two users query about concepts that their agents do not have in their respective ontologies, two users query about only some concepts that their agents do not have in their respective ontologies, and one user queries about concepts that its agent has in its ontology. The queried number of links is small and no negotiations are needed.

The second batch starts around Cycle 150, and repeats the above query scenarios. Figure 2 gives a brief overview of our query scenarios.

Our query scenarios are staggered and packed to investigate the response behaviors of the agents. Since the number of negotiation threads is limited for each agent, packed queries with high link demands may lead to only partial link retrievals. Our query scenarios also come with low and high link demands. Low link demands do not require or require fewer collaborations, while high link demands prompt the agents to plan for collaborative actions. Finally, an agent may or may not know some of the queried concepts. The agent's ontology specifies this knowledge. When an agent knows the queried concept, it has more options, approaching different neighbors for help. When it does not know the queried concept, then it shifts the responsibility to one of the neighbors, essentially making itself a relay station.

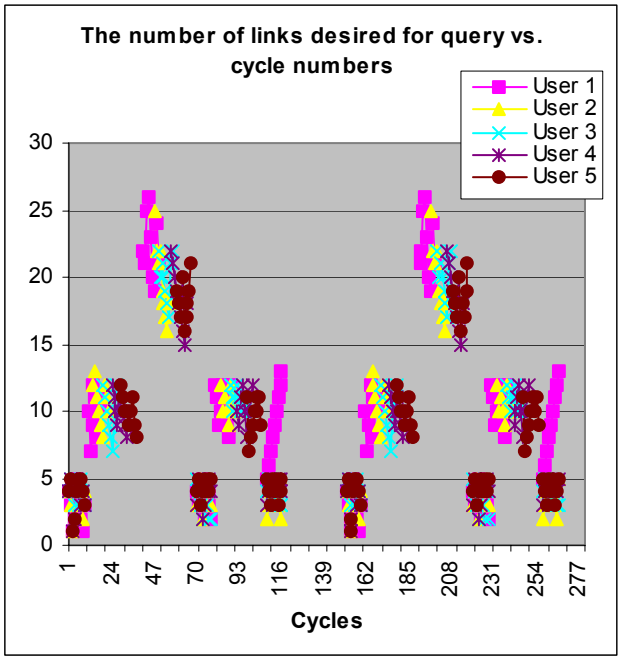

Figure 2 The number of links for the queries submitted by the software users to the agents for each cycle.

Given the above query scenarios, we further vary two sets of parameters: the number of negotiation threads and the credibility values in the translation tables. We vary the number of negotiation threads between 0 and 5. When the number is 0, the agents do not have collaborative capabilities since they can contact other agents. When the number is 5, an agent can simultaneously conduct 5 negotiations. Thus, this number directly impacts the resources that the agents have to collaborate to satisfy queries. This is relevant to *operational* constraints. There are also six sets of translation tables. In the first set, all credibility values of all translations are above zero. In this situation, every concept that one agent knows has four translations. In the second set, one of the agents has what term as a "narrow ontology". That is, its translation table contains many NIL translations, above 50%. (See Table 1.) In the third set, two agents have narrow ontologies. In the fourth set, three agents do; in the fifth set, four agents do; finally, all agents do. With these sets, we want to see how successful the agents are in satisfying high-demand queries. This is relevant to *ontological* constraints.

Given the six different numbers of negotiation threads and six sets of translation tables, we carry out a total of 36 runs using the same set of query scenarios.

5.2. Parameters Collected

Our long term effort is to study the learning of distributed ontologies, including the self-modification of the translation credibility values, and the exchange of ontological knowledge among the agents. However, at the time of the writing of this paper, we have not conducted a focused analysis on that.

Instead, our current experiments concentrate on two sets of parameters:

(1) Neighborhood Profile Parameters: For each neighbor, an agent collects parameters documenting the outcomes of their past

interactions. These parameters are also used in the computation of a neighbor's utility measure, as described in Section 3. Table 2 documents the definitions of these parameters.

(2) Query Result Parameters: For each query, an agent collects parameters documenting the characteristics of the query and the query outcome. Table 3 documents the definitions of these parameters.

Parameters	Definitions
_numSuccess	The number of successful negotiations that the agent has initiated to neighbor *i*
_numHelp	The number of successful negotiations that the agent has received from the neighbor *i*
_numRequestTo	The total number of negotiations that the agent has initiated to the neighbor *i*
_numRequestFrom	The total number of negotiation requests that the agent has received from neighbor *i*
_successRate	*_numSuccess/ _numRequestTo*
_helpRate	*_numHelp/ _numRequestFrom*
_requestToRate	*_numRequestTo/ _totalRequestTo* where *_totalRequestTo* is the sum of all negotiations that the agent has initiated
_requestFromRate	Presently this number is not updated, as our negotiation design does not incorporate the argumentative reasoning in [5]. However, we plan to re-visit this number in the future once the interpretation module is completed. This number tells the agent how much neighbor *i* relies on the agent

Table 2 Neighborhood profile parameters.

Parameters	Definitions
_originator	The originator of the query, either from a software user (ID) or another agent
_cycle	The cycle ID when the query is first generated
_numLinksDesired	The number of links desired by the query
_numLinksRetrieved	The number of links retrieved at the end of the retrieval process and presented to the user, always smaller than *_numLinksDesired*
_conceptName	The query keyword
_successQuality	*numLinksRetrieved/numLinksDesired*
_duration	The actual elapsed time between the receipt of a query and the presentation of the query results to the user
_listLinks	The list of links retrieved and presented to the user at the end of the retrieval process

Table 3 Query result parameters.

5.3. Results

Our overall, longterm plan of analysis aims at analyzing the results at eight different levels. At level 0, we derive an overview of the correctness and assessment of the results. At level 1, we want to analyze the agents' retrieval quality in the two similar batches of queries. At level 2, we aim to compare across the agents and see whether there are significant patterns. At level 3, we want to look into the retrieval results of each segment. Note that each segment has its unique set of characteristics (Section 5.1). At level 4, we want to investigate the role of the concepts. Some concepts may have few supporting links and some have many. At level 5, we will analyze the impact of different queries on the quality of the retrieved results. A query with a high-link demand may not necessary result in poorer results than one with a low-link demand. At level 6, we plan to examine closely the impact of the translation tables with narrow and wide ontologies, and how distributed ontology learning may help improve the tables for better query effectiveness. Finally, at level 7, we will study the operational impact of the threads as a constrained resource.

In this paper, we report on some preliminary level-0 analyses. Figures 3-7 show the graphs of *_successQuality* vs. the number of threads for each software user. Here are some observations:

(1) The average *_successQuality* of a user's queries increases as expected when the number of threads increases. This is because for high-demand queries that call for collaborations, the agent has more resources (i.e., negotiation threads) to use.

(2) The average *_successQuality* of a user's queries drops significantly whenever the corresponding agent has a narrow ontology. However, the drops are more significant when the number of threads is smaller. This indicates that link retrieval, in our application, benefits from the collaborative distributed ontology design. When agents are able to collaborate more often, the *_successQuality* of a query is higher.

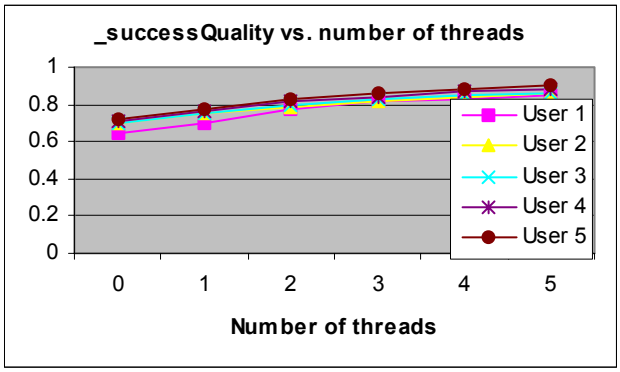

Figure 3 The average *_successQuality* value of each user's queries vs. the number of threads where no agents have narrow ontologies.

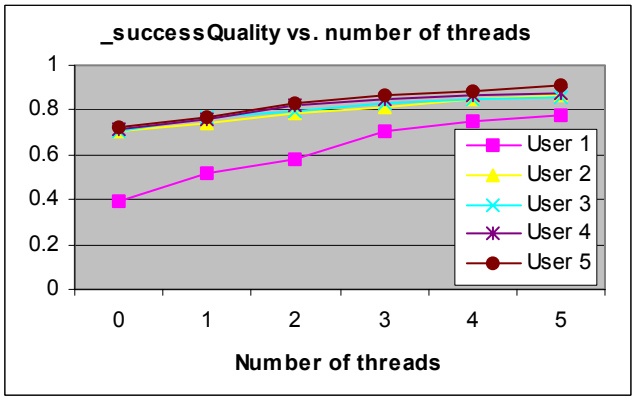

Figure 4 The average *_successQuality* value of each user's queries vs. the number of threads where agent 1 has narrow ontology.

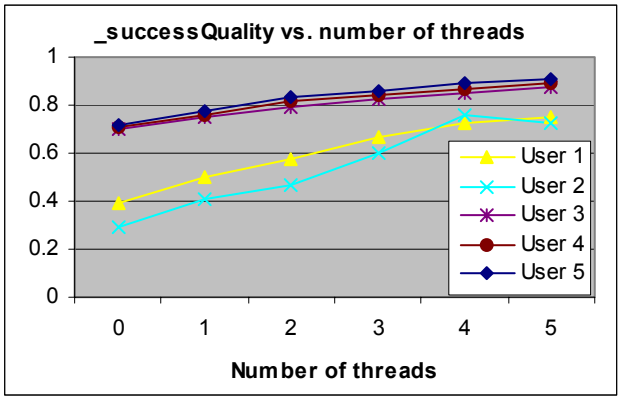

Figure 5 The average *_successQuality* value of each user's queries vs. the number of threads where agents 1 and 2 have narrow ontologies.

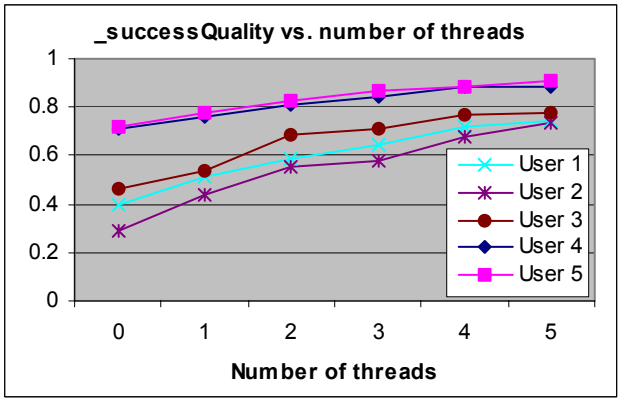

Figure 6 The average *_successQuality* value of each user's queries vs. the number of threads where agents 1, 2, and 3 have narrow ontologies.

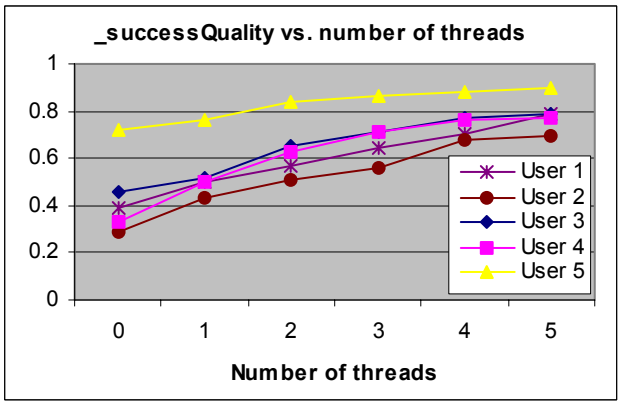

Figure 7 The average *_successQuality* value of each user's queries vs. the number of threads where agents 1, 2, and 3 have narrow ontologies.

(3) Figure 8 shows the average *_successQuality* and standard deviation of all queries for each number of threads. As we can see, with a higher number of negotiation threads, queries are satisfied more successfully (high average values), and also more consistently (low standard deviation values).

(4) Figure 9 shows the average *_successQuality* for agents with narrow ontologies and those with non-narrow ones. Note that if agent *A1* does not have a translation for mapping its concept name *C1* to any of agent *A2*'s, that does not necessarily mean that *A2* does not have a translation mapping one of its concepts to *A1*'s concept name *C1*. This is by design as we ultimately aim to show how collaborative agents can learn new translations or refine old ones as they help each other in satisfying queries. As observed, the number of narrow ontologies does not impact the success quality. From the operational point of view, this is unexpected. When the number of narrow ontologies within the multiagent system increases, we expect that more agents would *relay* queries to their neighbor, and that would cause the negotiation threads to be used more frequently.

(5) Figure 10 shows the average *_duration* (in seconds) for each query to be processed and presented back to software user 1 (by only agent 1), for different numbers of negotiation threads. As observed, when the number of threads increases, it takes longer for a query to be responded to. This observation was not anticipated. However, upon further analysis, we realize the following. When an agent has more threads, not only it can approach more neighbors for help, but it also receives more requests for help from other agents. As a result, the agent manages more tasks and slows down its processes for retrieving and supplying results to the software users. This indicates an oversight in our design with regards to the efficiency of our implementation. We are currently reviewing our program code to pinpoint the places where we could optimize the multi-threaded programming portion. We will also perform the same analysis on all other software users and agents to see whether the same patterns are observed as well.

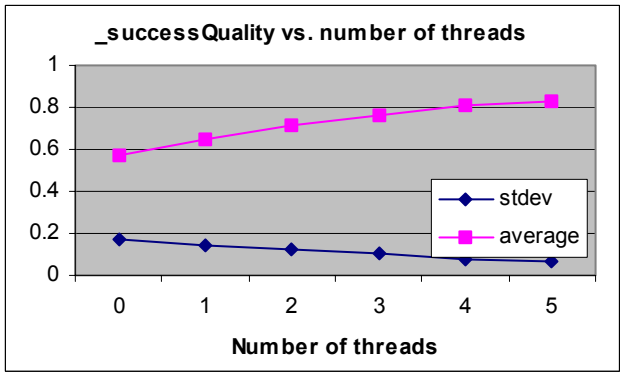

Figure 8 The average and standard deviation of the *_successQuality* for all users vs. the number of threads.

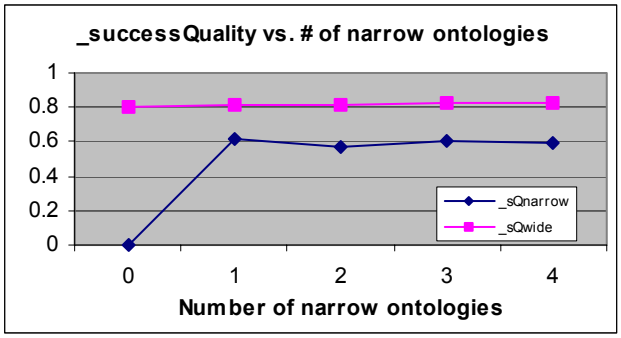

Figure 9 The average *_successQuality* for agents with narrow ontologies and agents with non-narrow ontologies. The *_sQnarrow* value for the 0 narrow ontologies is not applicable.

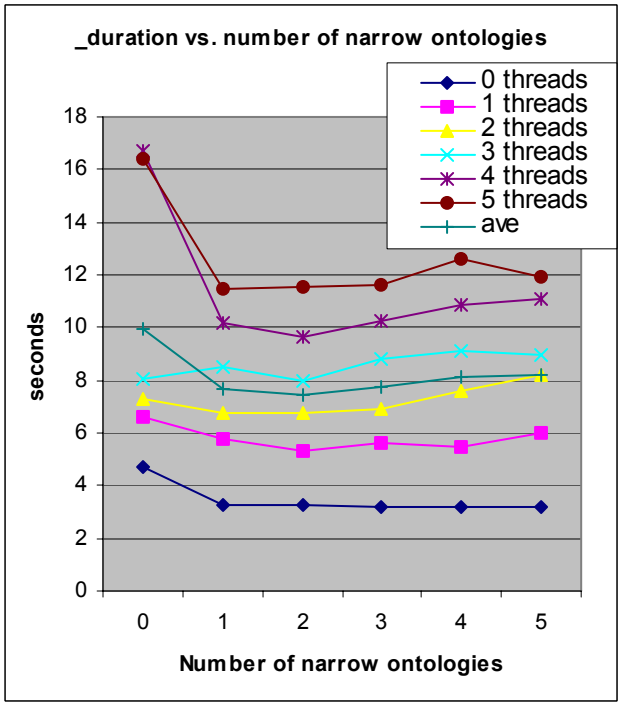

Figure 10 The average *_duration* for agent 1, for different numbers of threads, vs. the number of narrow ontologies.

(6) From Figure 10, the average *_duration* values for the different numbers of narrow ontologies are 9.96, 7.66, 7.41, 7.73, 8.15, and 8.24 seconds, respectively. The multiagent system where the agents do not narrow ontologies, unexpectedly, have the highest average *_duration* value. This value drops, has a minimum when the number of narrow ontologies is two, and then climbs up consistently for the next three sets. We are currently investigating the reasons behind this curve, to at least explain the data of the 0-narrow ontology case. Coupling the above observation with that in from Figure 9, we see that when the number of narrow ontologies increases (starting from number = 2), even though the *_successQuality* value remains mostly the same, the *_duration* value starts to dip. This clarifies somewhat our study.

(7) Figure 11 shows the average neighbor profile of agent 1 of its neighbors: *_numSuccess*, *_numHelp*, *_numRequestTo*, and *_numRequestFrom*. The values of *_numHelp* and *_numRequestFrom* are the same; that is, the *_helpRate* is 100%. For this agent 1, the number of times it has requested for help is smaller than the number of times it has entertained other agents' requests. This indicates that the query scenarios tend to invoke collaborations, causing the originating agents to ask for help from many different neighbors. From the graph, we see that the agent approaches more neighbors for help as it has more negotiation threads. However, when the number of threads is 5, the rate levels off just a little, indicating that a convergence may occur when the number of threads is larger than 5. This means that in our current experimental setup, our link demand is still more than what the agents can handle.

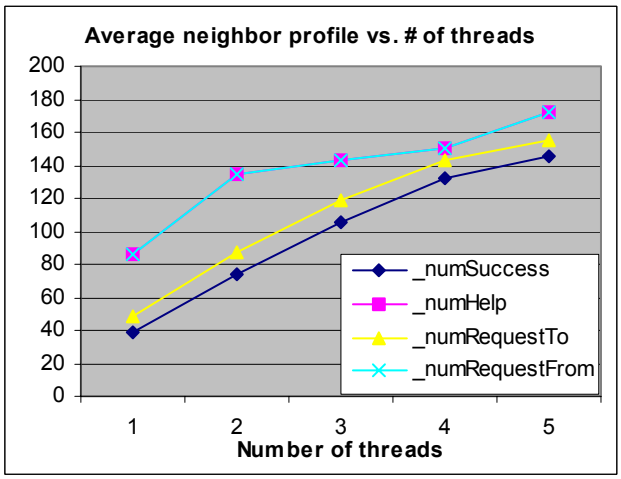

Figure 11 The average neighbor profile for agent 1 of its neighbors vs. the number of threads

(8) Figure 12 shows the average *_successRate* vs. the number of threads available. As observed, the agent is able to negotiate more successfully when the number of threads increases. This is expected since with more threads available, an agent is able to entertain more requests. Coupling this with Figure 11, we see that agent 1 is able to conduct *more* negotiations *more* successfully when the number of threads increases—more effectively and more efficiently. This is a good indicator that would help guide the design of distributed ontology learning in our work.

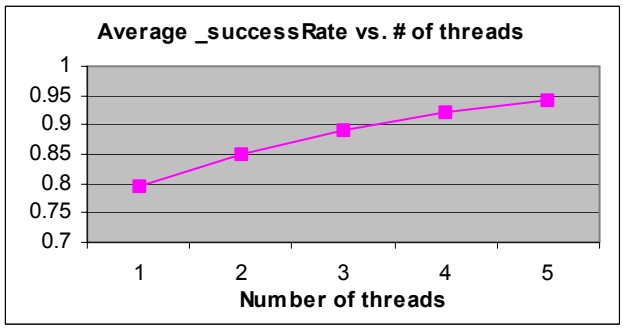

Figure 12 The average neighbor profile for agent 1 of its neighbors vs. the number of threads

(9) Figure 13 shows the *_requestToRate* vs. the number of threads available. As observed, when the number of threads is 1, agent 1 relies on agent 2 (or N1) almost heavily. This is due to the fact that in the beginning of an agent, all neighbors are weighted very

similarly; as a result, the agent will approach the first neighbor that it knows. However, as the number of threads increases, the agent is able to collaborate more with other neighbors. As a result, the reliance on N1 greatly decreases. Meanwhile, the reliance on the other three neighbors steadily increases. This is a good lesson, as we now know that in order for the system to exhibit un-intended bias favoring one neighbor over next, we need to have enough number of threads, laying the groundwork for the distributed ontology learning design of our work.

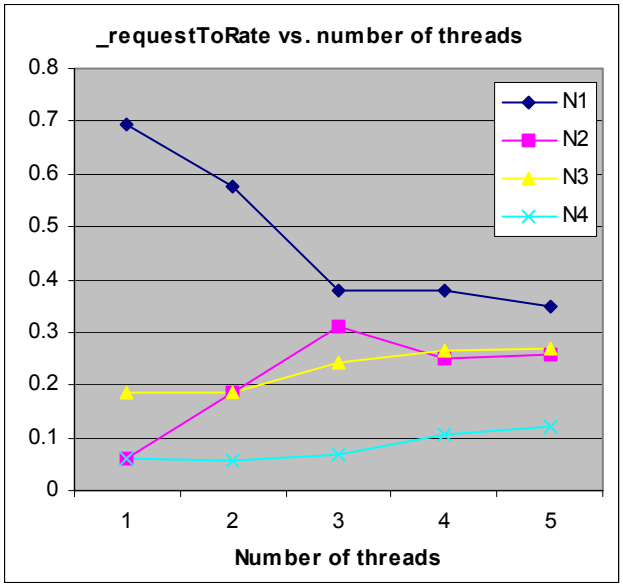

Figure 13 The *_requestToRate* from agent 1 to its neighbors, N1 (agent 2), N2 (agent 3), N3 (agent 4), and N4 (agent 5) vs. the number of threads.

6. CONCLUSIONS

In this paper, we have described our work-in-progress with collaborative understanding of distributed ontologies in a multiagent framework, focusing on the operational components. We have outlined the methodology and design of our framework. The methodology involves building agents with key operational components to support ontological functions such as query processing, query composition, negotiation, and collaboration. We have also briefly discussed our implementation. We have focused mainly on our on-going experiments. We have described our query scenarios, translation tables, and ontologies, as well as two key sets of parameters colleted from our experiments: neighborhood profile and query result parameters. Our experiments have generated a lot of data that we are currently reviewing and investigating. We have reported on some preliminary, low-level analyses to give an overall assessment of our system's feasibility and correctness. In general, we see that the number of negotiation threads available to each agent in the system has a key role in determining the *_successQuality* of a query task, the average *_successRate* of a negotiation, and the degree of collaboration among agents. We also see that the number of "narrow" ontologies influences the agents' behaviors negligibly. We plan to look into this finding further.

Our immediate future work includes (1) completing the 7 levels of analyses identified in this paper to analyze our infrastructure, (2) finishing the interpretation module to add complexity into the negotiation protocols, (3) activating the learning mechanism so that the translation credibility values can be revised dynamically, and (4) investigating the usefulness of the utility measure and its impact on the accuracy of translation. For the last item, remember that the utility measure of a neighbor is based on the credibility of the particular translation as well as the agents' relationships. That means, even if a neighbor is very knowledgeable (with high credibility), an agent may not approach that neighbor for help if the *_successRate* is low. As a result, our distributed ontology learning may be biased towards how close two agents have collaborated, and factor in less importantly the actual accuracy of the translation. Thus, in a way, we are addressing a type of *operational* distributed ontology: agents learn ontologies that are useful and credible to them, instead of only learning ontologies that are highly credible to them.

7. ACKNOWLEDGMENT

The author would like to thank JingFei Xu for her programming and running the experiments for this project.

8. REFERENCES

[1] Soh, L.-K. 2002. Multiagent, Distributed Ontology Learning, *Working Notes of the 2nd AAMAS OAS Workshop,* July, Bologna, Italy.

[2] Shafer, G. 1976. *A Mathematical Theory of Evidence*, Princeton, NJ: Princeton University Press.

[3] Soh, L.-K. 2002. A Mutliagent Framework for Collaborative Conceptual Learning Using a Dempster-Shafer Belief System, *Working Notes of AAAI Spring Symposium on Collaborative Learning Agents*, Stanford, CA, Mar 25-27, pp. 9-16.

[4] Soh, L.-K. and Tsatsoulis, C. 2002. Satisficing Coalition Formation among Agents, *Proceedings of AAMAAS'02*, July, Bologna, Italy.

[5] Soh, L.-K. and Tsatsoulis, C. 2002. Reflective Negotiating Agents for Real-Time Multisensor Target Tracking, in *Proceedings of IJCAI'01*, Seattle, WA, Aug 6-11, pp. 1121-1127.

A UML ontology and derived content language for a travel booking scenario

Stephen Cranefield, Jin Pan and Martin Purvis
Department of Information Science
University of Otago
PO Box 56, Dunedin, New Zealand
scranefield@infoscience.otago.ac.nz

ABSTRACT

This paper illustrates an approach to combining the benefits of a multi-agent system architecture with the use of industry-standard modelling techniques using the Unified Modeling Language (UML). Using a UML profile for ontology modelling, an ontology for travel booking services is presented and the automatic derivation of an object-oriented content language for this domain is described. This content language is then used to encode example messages for a simple travel booking scenario, and it is shown how this approach to agent messaging allows messages to be created and analysed using a convenient object-oriented application-specific application programmer interface.

1. INTRODUCTION

This paper is a response to the challenge problem for the AAMAS 2003 Workshop on Ontologies in Agent Systems. The challenge problem [1] was based on the description of a travel agent domain previously developed for an ontology tool assessment exercise organised by the Special Interest Group on Enterprise-Standard Ontology Environments within the European Union's OntoWeb research network [2]. The OAS'03 challenge was to "describe the design and (preferably) an implementation of a multi-agent system in that domain" with emphasis on "the ways in which ontological information is referenced, accessed and used by agents".

In this paper we illustrate the application of our previous work on the use of the Unified Modeling Language (UML) for ontology and content language modelling [3] and the automatic generation of Java classes from these models [4]. This work rests on four observations:

- The Unified Modeling Language is a widely known and standardised modelling language with a compact graphical notation, an XML-based serialisation format, and a lot of existing tool support. We believe that the use of UML for ontology modelling has great benefits in terms of industry acceptance of agent technology. Its principal weakness is the lack of (official) formal semantics, but we believe that ongoing efforts in this direction will remove this shortcoming.

- Much current software development is done using the Java programming language, and the majority of widely used agent development tools are based on Java. Programmers using these tools are most familiar with the use of object-oriented representations and application programmer interfaces (APIs).

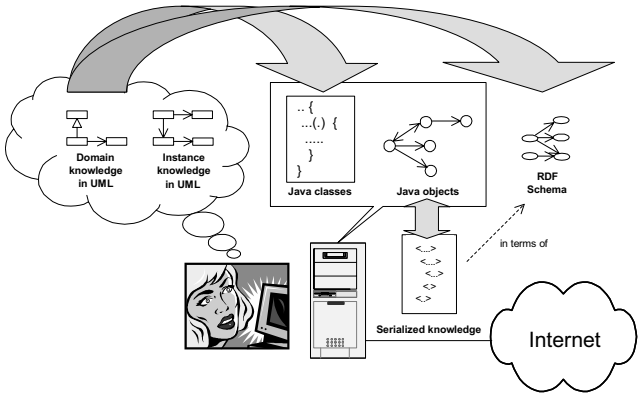

Figure 1: Overview of our approach

- The use of object-oriented structures to refer to domain objects within messages is convenient, but must be restricted to precise well understood usages in order to avoid semantic problems. [5].

- Multi-agent systems must coexist and interact with other distributed systems (both technological and human). These other systems have existing techniques for referring to domain objects using reference schemes such as World Wide Web Uniform Resource Identifiers (URIs). This style of reference goes beyond the notion of "standard names" (logical constants that denote each domain object) that lie behind the semantics of FIPA ACL's `query-ref` communicative act, and it is desirable to allow agents to use a more general notion of object reference when answering queries.

These observations have led us to develop our UML-based model-driven approach to implementing multi-agent systems. In Section 2 we give a brief overview of this approach, before presenting a simple UML travel booking ontology in Section 3, a discussion of the automatically generated ontology-specific content language in Section 4 and an illustration of its use in an agent application in Section 5. The paper closes with some comments on the applicability of this techique and some areas for future work.

2. OVERVIEW OF OUR APPROACH

Figure 1 presents a schematic overview of our approach to designing and implementing the message-handling component of agent systems.

The designer of an agent must have a mental model of the conceptual structure of the domain (the ontology) as well as an understanding of the structure of information describing instances of these concepts and their relationships. We believe the graphical nature of UML makes it a powerful tool for visualising these models: an ontology can be represented by a UML class diagram and instance information can be conveyed as a UML object diagram that shows the values of object attributes and the links (instances of associations) that exist between objects.

When creating the agent application, the programmer must translate these mental models into structures that can be manipulated within a programming language. When using Java, the natural counterpart to a concept in an ontology is a Java class. Although other representations can be used, such as string-based encodings of languages defined by grammars, the most convenient representation for a Java programmer is to have Java classes corresponding directly to the concepts that the agent will need to refer to when manipulating information about the world. To make this possible, we have defined XSLT [6] stylesheets that produce Java class definitions from an XMI [7] serialisation of a UML model (currently we support XMI 1.0 for UML 1.3) [4].

As agents need to communicate information about the world, it is beneficial to provide a straightforward mapping from the progammer's model of the domain (inter-related Java objects in our case) and the content language used to encode information within messages. However, standard agent content languages such as FIPA SL and KIF use a string-based logical representation. These are also generic and weakly typed languages in which domain concepts can only be referred to by name, rather than by more stongly typed mechanisms such as instantiation, and thus messages that do not conform to the agent's known ontologies can only be detected by run-time analysis. As an alternative to this approach, our Java classes generated from the ontology have a built-in serialisation mechanism that allows networks of inter-related objects describing domain objects to be included within messages. The serialisation uses the XML encoding of the Resource Description Framework (RDF) [8], which makes reference to concepts defined in an RDF schema that is also generated automatically from the ontology in UML [9].

This mechanism can also be used to serialise entire messages, including the outer agent communication language (ACL) layer. By defining the ACL in UML as well, and defining a set of UML 'marker' interfaces representing the concepts (such as *predicate* and *action description*) that comprise the required argument types for the ACL's various communicative acts, it is possible to conceptualise messages with arbitrary content languages (if modelled in UML) as object diagrams (see Figures 5 and 6 later in the paper).

Figure 2 illustrates how this technology can be integrated with a Java-based agent platform, and highlights a crucial aspect that addresses the third observation from the introduction: the need for careful use of object-oriented representations within messages. The figure shows a number of UML models: an ontology (top left), ACL and generic (i.e. SL-like) content language definitions, and an ontology-specific content language (top right). The ACL and the content languages are given as input to the XMI-to-Java transformation, and this results in Java classes that provide an object-oriented application programming interface that sits on top of the platform's built-in messaging system classes. However, the ontology is not directly translated to Java classes. We regard an ontology

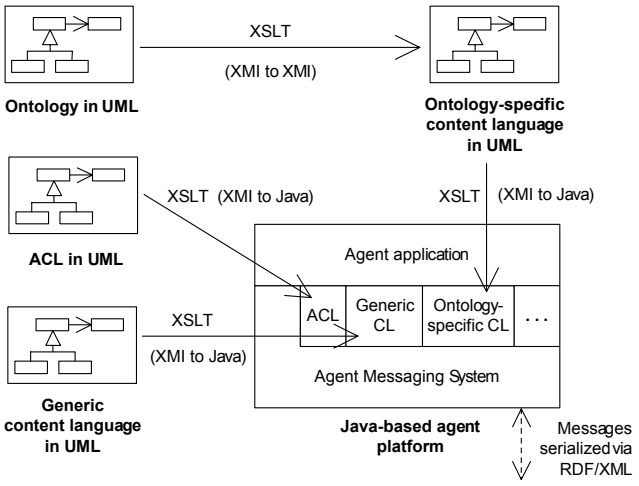

Figure 2: Integration with an agent platform

as a model of the problem domain, not as a model of the language used to encode *information* about the domain. In other words, an instance of an ontological class `Dog` would be an actual dog, not a description of a dog. When a structured expression corresponding to the structure of the `Dog` class appears within a message, this cannot be taken to be playing the role of a logical *term* (which always has a unique denotation), but instead might (depending on the context) play the role of a *proposition* (stating that an object with the specified properties exists) or an *identifying reference expression* (a reference to a possibly non-existent or non-unique object by describing its attribute values) [5, 3].

To avoid any confusion between the notions of ontology and content language, we provide the facility to use domain-specific object-oriented expressions within messages by generating from the ontology a UML model representing a specialised ontology-specific content language. From this, Java classes can be generated as for the ACL and generic content languages models. The generated ontology-specific content language for the travel booking domain is described in Section 4 and its use to create messages is illustrated in Section 5.

3. A TRAVEL BOOKING ONTOLOGY IN UML

Figure 3 shows a simple ontology in UML for the travel booking scenario. This uses two stereotypes, «resourceType» and «valueType» from a UML profile for ontology modelling that has been presented previously [3]. A resource type is a type of class for which the instances have an intrinsic identity, i.e. two instances with the same attribute values can be distinguished from each other. There is a possibility that an object of that class might be referred to using an identifier such as a unique name in some naming system, a UUID, or a World Wide Web Uniform Resource Identifier (URI). The semantics of the stereotype declare that the class has an additional optional association with a class representing some type of reference (e.g. the concept of a URI). This type is declared using a tagged value in the resource type class declaration, but this feature will not be used in this paper. The resource types in the travel booking ontology are `Customer`, `Consultation`, and `Place` and its subclasses `Hotel`, `City` and `Airport`.

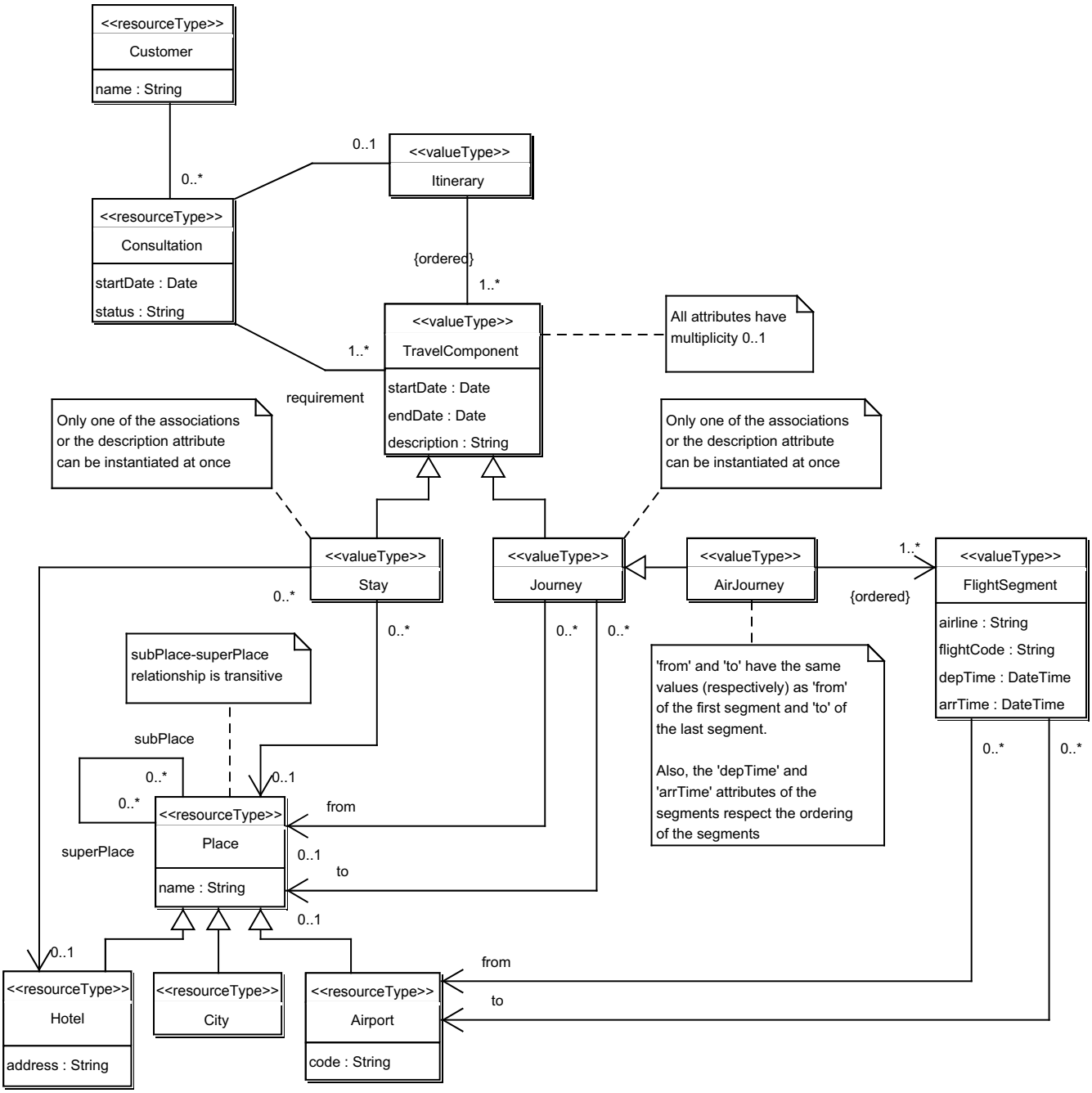

Figure 3: A travel booking ontology

A value type is a class with the opposite property: two instances with the same attribute values cannot be distingushed. Essentially it defines a type for (potentially complex) structured values that can be treated as logical terms within messages. Although there may be concepts included in an ontology which intrinsically seem to have this property, in many other cases the labelling of a class as a value type is a pragmatic decision about how instances of that type will be treated during inter-agent communication. It is a declaration that the Semantic Web principle that anything can be referred to using a URI will *not* be applied to instances of this class. Agents can expect to receive values of these types explicitly within messages, rather than have them referenced using URIs or other reference types. Also, they do not need to include mechanisms to keep track of references for those types. For example, in the ontology shown, the `Itinerary` class is declared to be a value type. Neither party in a travel booking conversation needs to be prepared to store references associated with itineraries, whereas it is expected that customers and consultations may be referred to by ID codes. This does not mean that an agent cannot make a query about an existing itinerary, but it must be done indirectly, e.g. by using an identifying reference expression that means "the itinerary associated with the consultation beginning on 15 July 2003 for the customer with code C05321".

The ontology defines a class `TravelComponent` which represents both customer requirements and the proposed components of an itinerary returned by the travel agent. This dual use is achieved by defining the attributes of the `TravelComponent` class and the associations of its subclasses `Stay`, `Journey` and `AirJourney` to be optional. A requirement can then be vaguely specified by providing only some of the possible information about a travel component. In an extreme case, only a value for the `description` string attribute might be provided (although this paper does not attempt to explain how a software agent might understand a textual description of the customer's requirements). For a travel component that is associated with an itinerary, it is expected that all information is provided, with the possible exception of the `description` attribute (this constraint could be included in the ontology, but is not modelled at present). Note that a consultation object may be linked directly with travel components representing the customer requirements as well as indirectly with other, different, travel component objects via an itinerary. The latter represent the final bookings.

The ontology includes a number of constraints presented as notes in dog-eared rectangles. These could be defined in more detail using the UML's Object Constraint Language, but are shown here in English for clarity. It is not intended that these constraints be used for inference in the current design—rather they serve as part of the specification for the correct implementation of agents using this ontology.

The ontology is not intended to be a complete model of the travel booking domain. It does not include many concepts needed for a realistic account (including the cost for a given itinerary). Also, to keep the model simple it does not use some features of UML that could provide a better model, such as the definition of an enumerated type defining a set of allowed values for the `Consulation` class's `status` attribute. For simplicity we regard the types `String`, `Date` and `DateTime` as being 'built in' primitive types in our UML profile which are handled specially during the generation of Java classes.

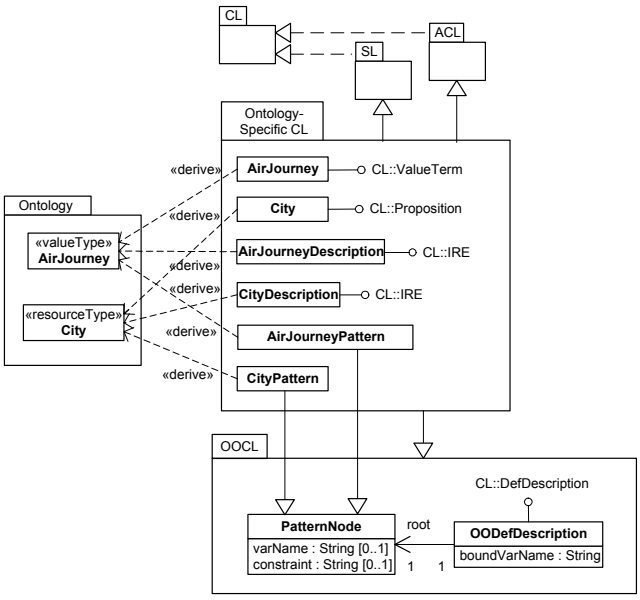

Figure 4: Derived classes in an ontology-specific content language

4. THE ONTOLOGY-SPECIFIC CONTENT LANGUAGE

Figure 4 presents an overview of the classes that are generated from the ontology to form the ontology-specific content language. The UML package in the middle of the diagram ("Ontology-Specific CL") contains the generated classes. This also includes classes from two other packages: SL (a UML model of a generic content language based on FIPA SL) and ACL (a UML model of a FIPA-style agent communication language). The inclusion of these additional classes allows complex statements to be formed using connectives from the SL language and also the use of ACL expressions to represent communicative actions (the details of this are beyond the scope of this paper). The CL package contains the set of marker interfaces that represent the generic types of expression that content languages are designed to describe (such as propositions and action descriptions). There is also a package OOCL shown. This defines some support classes used to create identifying reference expressions as networks of inter-connected "pattern nodes". These pattern node networks are used to describe an object by its properties and (possibly complex) inter-relationships with other objects.

To illustrate the nature of the derived classes in the ontology-specific content language we show the classes that correspond to two particular classes in the ontology: one that is a value type (`AirJourney`) and one that is a resource type (`City`). Each of these classes results in three generated classes in the ontology-specific content language. (Note that the dashed arrows labelled «derive» are UML dependencies, so they are directed from each derived class back to the one it depends on.)

As discussed in Section 3, an instance of a valuetype can be treated as a logical term within a content language, and so a corresponding class with the same name and structure (e.g. `AirJourney`) is generated and declared to implement the `CL::ValueTerm` interface. Some associations between `AirJourney` and other classes may need to be modified when translated to the new content lan-

guage, e.g. a reference to a resource type must be replaced by a reference to a derived ...Description class for that resource type (this type of class is discussed below). However, the details of the mapping rules for value types and for resource types are beyond the scope of this paper.

An agent might also want to refer to a value type instance using an identifying reference expression. Therefore, for each value type class there are two corresponding generated classes that can be used for this purpose: a simple ...Description class and a more complex ...Pattern class. The description class (e.g. AirJourney Description) implements the interface CL::IRE to show that that this can be used as an identifying reference expression (in particular, as a definite description—the only type of IRE currently supported). Under the mapping, all attributes and associations become optional because (for example) although an air journey in real life must necessarily have at least one flight segment, it is possible to refer to an air journey simply by specifying its date or departure and arrival cities.

The ...Pattern class is the same as the ...Description class, except it also extends the class OOCL::PatternNode and any association with another class must be changed to be an association with the appropriate ...Pattern class. The use of this type of class is illustrated in Figure 5 (which is discussed later in the paper).

For resource type classes, there can be no derived class that implements the CL::ValueTerm interface as it is not semantically meaningful to embed instances of that type within a message[1]. Instead, corresponding ...Description and ...Pattern classes are generated, as for value types. In addition, a class implementing CL::Proposition is generated in order to allow a convenient object-oriented form of proposition about objects to be used within messages. For this generated class, all attributes and associations become optional.

Further details of this approach to generating ontology-specific content languages can be found elsewhere [3], although the presentation here takes account of some subsequent minor updates to that previous work.

5. USING THE GENERATED CONTENT LANGUAGE

In this section we illustrate the use of an ontology-specific content language generated from the travel booking ontology. Figures 5 and 6 show UML object diagrams representing (respectively) query and response messages in a conversation between a customer and a travel booking agent. The query is an instance of the class QueryRef (predefined in the ACL package). This corresponds to the FIPA query-ref message type which represents a question asking another agent to identify an entity that satisfies particular properties. The content part of the message (represented by a link from the QueryRef object in Figure 5) is a instance of the class OODefDescription shown at the bottom of Figure 4. This class represents an object-oriented version of the iota binding operator from FIPA SL. It has an attribute boundVarName representing a variable name to be used to refer to the subject of the query-ref. This object is then linked to a network of typed pattern nodes, each of which describes some object in terms of its attributes and relationships with other objects. One of these pattern nodes is expected to have a varName attribute value matching the boundVarName value of the OODefDescription object. The other nodes may also have variable names specified, and these may be referred to within Object Constraint Language expressions appearing as the values of the optional constraint attribute of other nodes (this feature is not used in Figure 4). The message in the figure represents the following query:

Given a customer named Stephen Cranefield having a consultation with the requirements of flying from Dunedin to Melbourne on 14 July 2003 and needing accommodation there from the 14th until the 19th, what is the associated itinerary? (For simplicity, we assume our hero wishes to remain uncommitted after the 19th).

Figure 6 shows a UML object diagram representing a possible reply to this request. The message is an instance of the InformRef class from the ACL package. This is a structured version of FIPA ACL's inform-ref message type with two content expressions: a definite description (generally this will be the one that was included in the preceding query-ref) and an expression that identifies the entity that satisfies the query—this may be a value of a primitive type or a value type, a reference to an object (e.g. a URI), or another, hopefully more detailed, definite description. In the case of Figure 6, the definite description (not depicted in full) is the same as the one contained in the query message, with two additional links that provide references for the customer and the consultation objects.

The bottom part of Figure 6 represents the answer to the query and contains an instance of the Itinerary value type that comprises fully detailed value type instances for the air journey and the stay. The details about the hotel, airports and cities are encoded by links to ...Description objects, which describe those external instances of resource types in terms of their attribute values and some required relationships between the objects.

Note that these diagrams conceptualise the messages as UML object diagrams. As shown in Figure 1, the messages are physically realised as Java objects within the agent at run time and as RDF documents when being transported between agents. The Java classes are generated using an XSLT stylesheet [4] and they include code that handles the marshalling and unmarshalling of messages between the in-memory Java representation and the RDF serialisation format [9].

Figure 7 shows an example of how the query message can be created and sent from Java code, using the generated classes for the ACL and the content language (this is based on a simplication of the ACL model presented previously [3]).

There is no doubt that using this object-oriented API to construct messages is far more cumbersome for the programmer than writing a string in FIPA SL. However, it is likely that most messages will be constructed dynamically within code rather than by a static sequence of Java statements as shown in the figure. This approach also has the benefits of being strongly typed and model-driven: support for new ACLs and content languages can quickly be provided once they have been defined using UML. Furthermore, any disadvantages for the creation of messages are balanced by advantages in the analysis of incoming messages: it is much easier to examine a message using its object structure than by performing string matching operations.

[1] We would argue that even electronic entities such as instances of electronic currency are best regarded as external objects that agents refer to using references.

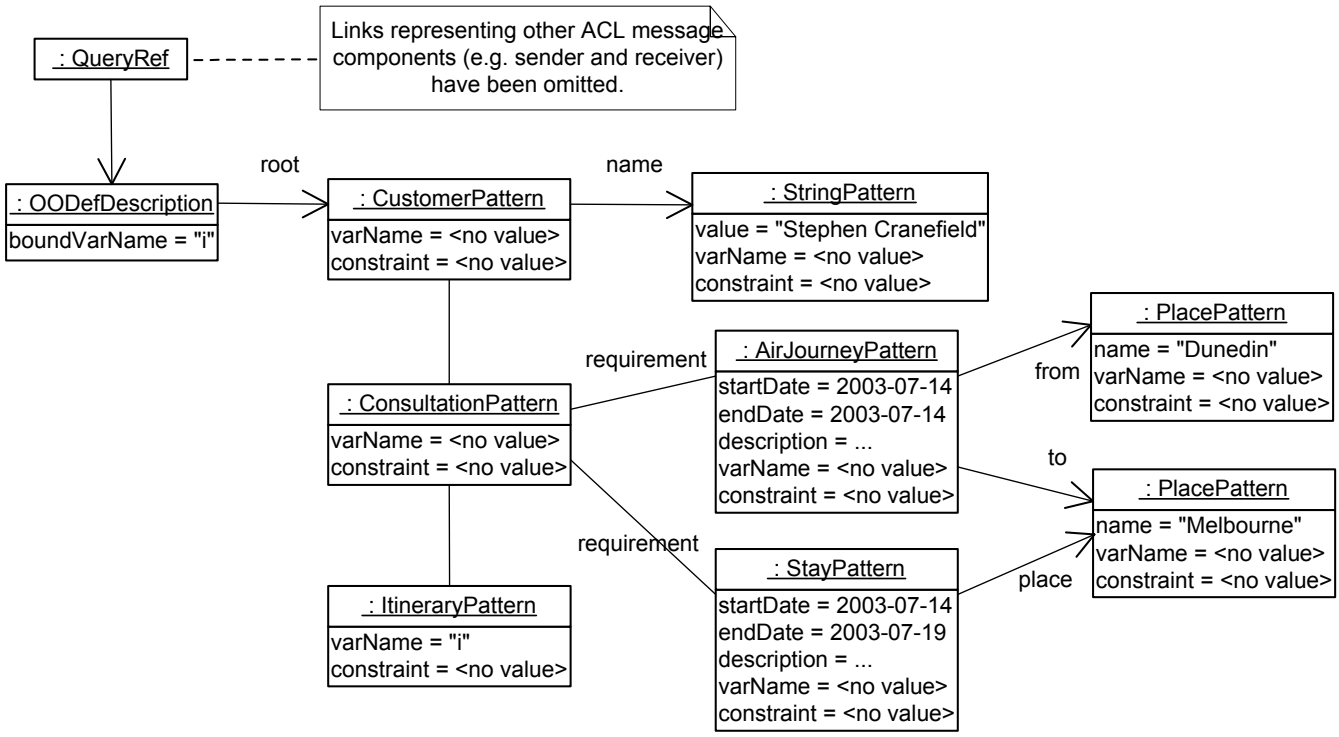

Figure 5: A travel booking request message

6. CONCLUSION

This paper has addressed the OAS'03 workshop challenge problem by illustrating the application of our UML-based model-driven approach to defining ontologies and then automatically generating related ontology-specific content languages along with corresponding Java classes and an RDF-based serialisation mechanism. We believe this approach has strong benefits for the software engineering task of designing and implementing agents to peform particular tasks in a given domain. Our work does not currently provide support for the construction of agents that are expected to have more general abilities, where inference may be required in order to determine how to respond to messages (although inference mechanisms based on object networks could be developed).

This technique is being incorporated into the Opal FIPA-compliant agent platform developed at the University of Otago.

An important avenue for future work is the enhancement of the API offered to programmers for constructing messages by providing a larger range of constructors in the generated classes. It would also be highly desirable to develop a technique for annotating the UML definition of ACLs and content languages with information that describes a concrete string-based syntax in such a way that parsers for this language can be generated automatically. This will allow interoperation with traditional FIPA agent platforms and will also give programmers the option of using the string-based syntax for creating messages within agent application code.

7. REFERENCES

[1] OAS 2003 Committee. OAS'03 challenge problem. http://oas.otago.ac.nz/OAS2003/Challenge/challenge.html, 2003.

[2] OntoWeb project. Project Web pages. http://www.ontoweb.org, 2003.

[3] S. Cranefield and M. Purvis. A UML profile and mapping for the generation of ontology-specific content languages. *Knowledge Engineering Review*, 17(1):21–39, 2002.

[4] S. Cranefield, M. Nowostawski, and M. Purvis. Implementing agent communication languages directly from UML specifications. In *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2002)*, volume 2, pages 553–554. ACM Press, 2002.

[5] Stephen Cranefield and Martin Purvis. Referencing objects in FIPA SL: An analysis and proposal. In *Proceedings of the Workshop on Agentcities: Challenges in open agent environments, 2nd International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2003)*, 2003.

[6] XSL transformations (XSLT) version 1.0. World Wide Web Consortium Web page, 2003. http://www.w3.org/TR/xslt.

[7] XML metadata interchange specifications. Object Management Group, 2003. http://www.omg.org/technology/documents/modeling_spec_catalog.htm#XMI.

[8] Resource Description Framework. World Wide Web Consortium Web page, 2003. http://www.w3.org/RDF/.

[9] S. Cranefield. UML and the Semantic Web. In I. Cruz, S. Decker, J. Euzenat, and D. McGuiness, editors, *The emerging Semantic Web*, pages 3–20. IOS Press, Amsterdam, 2002.

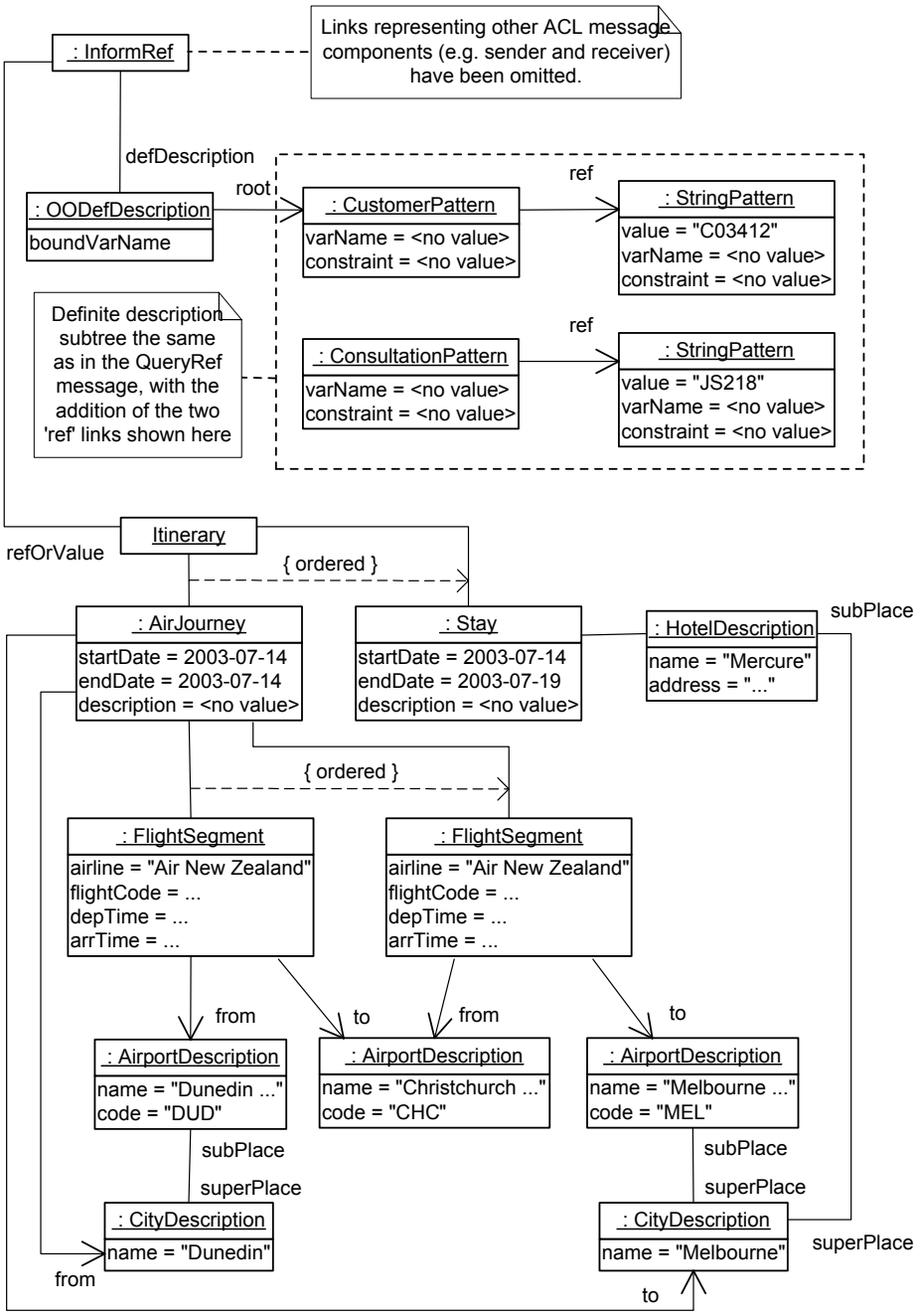

Figure 6: A reponse message from the travel booking agent

```
// Construct query structure containing variable i
CustomerPattern cust = new CustomerPattern();
ConsultationPattern cons = new ConsultationPattern();
ItineraryPattern itin = new ItineraryPattern();
StringPattern custName = new StringPattern();
AirJourney journ = new AirJourney();
Stay stay = new Stay();
Date monday = new Date(2003, 7, 14);
Date saturday = new Date(2003, 7, 19);
PlacePattern dunedin = new PlacePattern();
PlacePattern melbourne = new PlacePattern();
custName.setValue("Stephen Cranefield");
cust.setName(custName);
dunedin.setName("Dunedin");
melbourne.setName("Melbourne");
journ.setStartDate(monday);
journ.setEndDate(monday);
journ.setFrom(dunedin);
journ.setTo(melbourne);
stay.setStartDate(monday);
stay.setEndDate(saturday);
stay.setPlace(melbourne);
itin.setVarName("i");
Set reqs = new HashSet(); reqs.add(journ); reqs.add(stay)
cons.setRequirement(reqs);
cons.setItinerary(itin);
// Construct message object
Message m = new QueryRef(new AgentRef("agent1"), // Sender
                         // Recipients:
                         Collections.singleton(new AgentRef("agent2")),
                         // Content:
                         new OODefDescription("i", patternNetwork));
// Send message
m.send();
```

Figure 7: Using the generated Java code to create and serialise a message

An Initial Response to the OAS'03 Challenge Problem

Ian Dickinson
Hewlett-Packard Laboratories
Filton Road, Stoke Gifford
Bristol BS34 8QZ
U.K.
ian.dickinson@hp.com

Michael Wooldridge
Department of Computer Science
University of Liverpool
Liverpool L69 7ZF
U.K.
m.j.wooldridge@csc.liv.ac.uk

ABSTRACT

We present our initial response to the OAS '03 Challenge Problem. The Challenge Problem proposes an agent-assisted travel scenario, and asks what the role of ontologies would be to support the agent's activity. We discuss a belief-desire-intention (BDI) approach to the problem using our Nuin agent platform, and illustrate various ways in which ontology reasoning supports BDI-oriented problem solving and communications by the agents in the system.

Keywords

Agent applications, BDI agents, Ontology, Semantic web

1. INTRODUCTION

The call for papers for the AAMAS '03 workshop on Ontologies and Agent Systems (OAS'03) includes a challenge problem, adapted from an exercise by the OntoWeb project to assess different ontology environments. The challenge problem outlines a set of objectives for an agent-assisted travel planning system, in which an agent-based travel agent must co-operate with other agents to book a trip for a human client.

We have been investigating the design and development of belief-desire-intention (BDI) [16] agents for use in the Semantic Web [6]. One outcome of this research is a BDI agent platform, Nuin, which has been designed *ab initio* to work with Semantic Web information sources. At the time of writing, Nuin is still a work in progress. Nevertheless, we have investigated how key parts of the OAS challenge problem would be addressed by our platform.

This paper reviews the salient features of the Challenge Problem, in the context of a BDI agent. We briefly review some of the characteristics of the Nuin platform, before presenting a series of vignettes that show how we address some of the challenges in the Challenge Problem. As it represents a rich and plausible scenario, we are continuing development of a complete solution to the Challenge Problem using the Nuin platform.

2. OUTLINE PROBLEM

The scenario for the Challenge Problem is based on a travel agent in New York, for which we are asked to develop an agent-based application. The travel agency's clients come to make bookings for trips they wish to take, and the agency is responsible for making reservations with various travel service providers (airlines, hotels, train companies, etc) to satisfy the client's needs. The Challenge Problem description gives a rich description of the kinds of knowledge possessed by various players in the scenario, from which we distil the following principal objectives and assumptions:

1. Clients come to the travel agency with more-or-less specific objectives for their trip, for example a departure date and destination, a tourist attraction to visit or an academic conference to attend.
2. Clients have individual preferences about many aspects of the travel services that may be booked, including dietary choice, smoking/non-smoking, cost, comfort level, choice of provider, etc.
3. The travel agency does not posses the data to arrange trips or trip segments, but must request this information from other agents. For simplicity in building the model, we assume that the travel agency does know the identities of the supplier agents (a more realistic interpretation would be to require that the agency contacts suppliers through a brokerage or advertising service).
4. Requests from the travel agency to the suppliers may be made at varying levels of specificity (for example "a flight from Washington to London" vs. "a seat on BA1234 from Washington to London").
5. Interaction with the suppliers will produce multiple potential solutions to the client's initial request. Priority should be given to solutions that match the clients' preferences, noting that the preferences may not be unambiguously consistent.
6. Solutions may specify constraints that are not relevant to the client ("no dogs in the hotel"), or may be of unknown relevance.

The vocabularies used by different suppliers and the travel agency may vary – for example, one may use kilometres while another uses miles.

2.1 Issues from the challenge problem

From the distilled problem statement, we highlight the following challenges for agents to address in this scenario. Note: this is not intended to be an exhaustive list.

- modelling the motivations and attitudes of the actors – there are a number of actors in the scenario, and we assume each to be predominantly self-interested. The BDI model accounts for the mental attitudes of a given agent, but to correctly represent the scenario, the travel agency agent, for example, must also account for at least the goals and preferences of the user. We do not assume that user preferences can, in practice, be reduced to a utility function.
- ownership and responsibility – arguably, the motivations of the travel agency itself should be accounted for. For example, should the agent recommend suppliers that have high commission rates for agency, even if the utility to the client is neutral or reduced? Should the agent explicitly model its contract to the client and the travel agency?
- reconciling vocabularies – different agents or services will use different vocabularies, for the same or

overlapping concepts. A simple example is the use of miles and kilometres for distance, but other examples will be more subtle or complex.

- varying degrees of detail in queries – at different stages in the trip design process, queries will have differing degrees of specificity. For example, "is it possible to take a train from London to Paris", compared to "when would a train departing Waterloo at around 10:00 on the 27th arrive in Paris?"
- checking solutions for acceptability – testing for basic feasibility, including such constraints as not being in two separate vehicles at the same time
- ranking and critiquing candidate solutions – given that more than one possible solution exists, the user's preferences should be used to rank the solutions. This is likely to be needed incrementally, to control the growth of the search space.
- choosing which constraints to relax during negotiation – if a good solution is not available with the current constraints, it may be that relaxing some of them will yield an acceptable solution. For example, a three star hotel might have to be used to keep the cost within the client's budget.
- choosing when to ask the client to resolve choices or provide more preference constraints – this involves managing the dialogue with the client to neither require them to 'brain-dump' their entire preference set at the beginning, nor to be barraged with low-level questions.

Not all of these issues are addressed by the use of ontologies, though it would seem that the use of an ontology representation has some impact on the solutions to most, if not all, of them.

3. OVERVIEW OF THE NUIN PLATFORM

Nuin [9] is an agent platform we have created to assist agent designers to program deliberative agents, with a particular emphasis on BDI [16] agents. Nuin is founded on Rao's AgentSpeak(L) [15], and extended to make a practical, Java™-based programming tool. In this section, we briefly introduce some of the key features of Nuin, in order to provide some background for the solution vignettes in section 4.

A key objective in developing Nuin was to create a flexible platform for building practical agents from high-level abstractions, such as beliefs, desires, intentions and plans. The emphasis has not been on building agent infrastructure services: we assume the existence of an underlying services architecture, and Nuin provides a services abstraction layer that allows to bind to a particular service fabric, for example the Jade agent platform [5]. Nuin's architecture is influenced by the FIPA abstract architecture [12], in order to better utilise existing agent infrastructure projects. We do not, however, assume that Nuin will operate only in an FIPA environment.

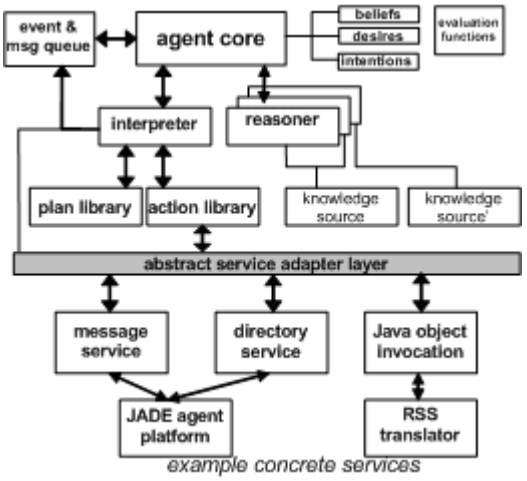

Figure 1: Outline Nuin architecture

As figure 1 shows, each Nuin agent has an interpreter, which runs one or more scripts to provide the agent behaviour. An agent also has a set of beliefs, as first-order sentences, and any number of other knowledge sources, each of which is labelled by a distinct symbol. A knowledge source can be wrapped by one or more reasoners, which provide additional services over the storage and retrieval of asserted sentences. Backwards-chaining reasoners attempt to solve queries that they are given, in essence by building a proof tree. Forwards-chaining reasoners opportunistically assert additional entailments when formulae are asserted into the knowledge store.

The key abstraction in defining an agent's behaviour is the *plan*. Following AgentSpeak(L), a plan has one or both of: a triggering event condition or a logical postcondition. Internal control flow within the agent is managed by a queue of events, which can be exogenous or endogenous, and include messages from other agents as a sub-type. A triggering condition is a Boolean expression formed from two predicates over events: `on( E )` is true when E unifies with the current event at the head of the event queue, whereas `after( E )` is true when E unifies with an event from the agent's memory of past events. The body of a plan is a set of individual actions, composed with either a sequence operator (`;`) or a non-deterministic choice (`|`). Plans may invoke other sub-plans directly or by post-condition, and may recurse.

Currently, a plan library is supplied to the agent as part of its script. However, there is no *a priori* reason why the plans could not be dynamically generated by an online planner, and this is a capability we intend to add in the future.

All of the abstractions shown in figure 1 are specified using Java interfaces, and created using the design pattern Factory Pattern. This makes it very easy for a programmer to provide a customised variant of a particular part of the system. This flexibility and extensibility was a key design goal for the platform. The configuration of the agent is specified as an RDF document, the URL of which is passed to the agent as a start-up parameter.

Given that we want to develop agents for the Semantic Web, we allow RDF stores as knowledge sources, using Jena [13]. In addition, all internal symbols are URI's. Jena's ontology reasoners are used to extend the entailments in the RDF stores, where OWL or DAML+OIL sources are available.

The next section illustrates the use of Nuin in a series of vignettes addressing some of the challenges outlined in section 2.1. Note that the script syntax illustrated is also a configurable aspect of our platform. The encoding illustrated is NuinScript,

but this is only one possible syntax that can parse into the internal abstract syntax form. An XML encoding is also planned.

4. SOLUTION EXAMPLES USING NUIN

4.1 Preamble: use of ontologies

In the challenge problem description, a sample ontology for this domain is provided by Corcho et al [7]. We decided to create our own ontology, although it shares some characteristics with that of Corcho and colleagues. Our ontology is written in DAML+OIL [2], which allows us to use richer constructs than that in the sample ontology. For example, figure 2 shows a 5-star hotel in our formulation:

```
<daml:Class rdf:ID="QualityRating">
 <daml:oneOf rdf:parseType="daml:collection">
  <travel1:QualityRating rdf:about="#OneStar"/>
  <travel1:QualityRating rdf:about="#TwoStars"/>
  <travel1:QualityRating
rdf:about="#ThreeStars"/>
  <travel1:QualityRating
rdf:about="#FourStars"/>
  <travel1:QualityRating
rdf:about="#FiveStars"/>
 </daml:oneOf>
</daml:Class>

<daml:Class rdf:ID="FiveStarHotel">
 <rdfs:subClassOf rdf:resource="#Hotel"/>
 <rdfs:subClassOf>
  <daml:Restriction>
   <daml:onProperty rdf:resource="#rating"/>
   <daml:hasValue rdf:resource="#FiveStars"/>
  </daml:Restriction>
 </rdfs:subClassOf>
</daml:Class>
```

Figure 2: DAML+OIL ontology fragment for five-star hotel

Compare this with the definition from the sample ontology (slightly abbreviated):

```
<rdfs:Class rdf:ID="hotel5star">
 <rdfs:comment>First class hotel</rdfs:comment>
 <rdfs:subClassOf rdf:ID="#hotel" />
 <NS0:numberOfStars>5</NS0:numberOfStars>
</rdfs:Class>
```

Figure 3: RDFS fragment from OAS'03 call for papers

RDFS does not have the machinery to declare that quality ratings may have exactly one of one, two, three, four or five as values. Nor is it possible to infer in RDFS that having a five-star rating and being a hotel *entails* being in the class FiveStarHotel. In the sample RDFS ontology, membership of this class must be stated explicitly. Finally, we note that the RDFS ontology requires class 'hotel5star' to be treated as an instance, since the class itself is the subject of the statement '`numberOfStars 5`'. Looking ahead, we intend to switch to using OWL [8] as our ontology language [1]. The use of classes as instances necessarily places the `hotel5star` construct in the OWL Full variant of that language, for which it is known that inferencing is expensive and incomplete. Note that we have chosen not to use cardinality restrictions to define hotel star-classes. Consider this definition

[1] The only reason we have not yet adopted OWL is that tool support is limited, partly because, at the time of writing, the OWL specification is not yet complete. This situation is improving rapidly, however, and we anticipate switching to OWL very soon.

```
<daml:Class rdf:ID="FourStarHotel">
 <rdfs:subClassOf rdf:resource="#Hotel" />
 <rdfs:subClassOf>
  <daml:Restriction>
   <daml:onProperty rdf:resource="#hasStar" />
   <daml:cardinality rdf:value="4" />
  </daml:Restriction>
 </rdfs:subClassOf>
</daml:Class>

<Hotel rdf:about="http://quite-nice.com">
 <hasStar rdf:value="*" />
 <hasStar rdf:value="*" />
 <hasStar rdf:value="*" />
 <hasStar rdf:value="*" />
</Hotel>
```

Figure 4 Using cardinality restrictions as for hotel classes

Figure 4 suggests that the Quite-Nice hotel is a four star hotel, but we cannot know this for certain. Given the open world nature of the semantic web, we cannot be sure that we have collected all of the relevant statements about URI `http://www.quite-nice.com` – we may yet discover an additional `hasStar` statement. We can only rule out the Three Star and below classes.

Space does not permit a full explanation of our sample ontology in this paper. Elements of the ontology will be introduced below as needed. The full ontology is, however, available online at: `http://jena.hpl.hp.com/ontologies/travel1`.

4.2 Initial client to agent communication

At the beginning of the process, the client's basic goal to take a trip of a certain form must be communicated to the agent. We leave aside the machinery of the human-computer interface (important though it is), to consider the process. The agent must have access to two kinds of knowledge:

- the primary goal that initiated the travel request, and
- the client's travel preferences

We assume that a message is delivered to the agent with the first of these, and that the second can be queried from a general database of known preferences. Since we are interested in Semantic Web agents, we assume that the client preference information is available in at least RDF (if not DAML+OIL or OWL).

What should the message contain? An important choice is whether the agent is seen as a collaborative partner, or a subordinate. In the second case, the message might be a FIPA `request` message, which takes an action as parameter. The action is essentially an encoding of "book a trip respecting these constraints". The agent would directly adopt an intention to carry out the action. The first case would correspond to sending a FIPA `inform` message [2] saying "the client has a goal to go on a trip, with these constraints". We would then rely on the agent being programmed with social or behavioural rules that would translate this recognition of the user's goal into an intent of its own to assist with the development of the travel plan. For the scenario of a single client walking into the travel agency's office seeking to make a booking, the difference between these two approaches is slim. Indeed, the collaborative approach adds extra complexity that the directive approach avoids. However, consider the often quoted desire for proactive behaviour in agents. The recognition of the user's goal may arise by inference, rather than by a directive from the user. If the agent is

[2] Note that the FIPA ACL specification [11] does not include a performative that directly delegates a goal to another agent.

able to infer that the user has a goal to make a trip (e.g. by having a paper accepted at a conference), it can proactively instigate the travel planning process.

Both approaches are supported by Nuin. Figure 5 shows a plan fragment[3] that reacts to an incoming message that the user has a goal to make a booking, and creates a suitable intent.

```
plan
 on message()
     {fipa:performative ~ fipa:inform,
      fipa:content ~ goal() {
          user ~ ?u,
          makeTrip ~ ?t,
          constraints ~ ?c
      }
     }
 do
   holds desire( cooperate, ?u ) ;
   intend-that
     finalised( trip( ?t, ?proposal ), ?c )
end.
```

Figure 5: plan to adopt user goal as agent intention

Thus: if a message is received informing the agent that the user has a goal to make a given trip, and the agent desires to be cooperative with that user (it may, of course, be predisposed to be generally cooperative), then adopt an intention to achieve a finalised proposal starting from the initial conditions `?t` and respecting constraints `?c`. We can make use of an ontology of different booking types to generalise this condition slightly:

```
plan
 on message()
     {fipa:performative ~ fipa:inform,
      fipa:content ~ goal() {
          user ~ ?u,
          makeBooking ~ ?b,
          constraints ~ ?c
      }
     }
 do
   holds desire( cooperate, ?u ) ;
   holds rdf:type( ?b, makeTrip ) ;
   intend-that
     finalised( trip( ?b, ?proposal ), ?c )
end.
```

Figure 6: plan to detect a trip booking and adopt an intention

Figure 6 shows a plan that reacts to any booking request, but checks that it can infer a trip booking before proceeding. The `rdf:type makeTrip` may be stated directly, or it may be an entailment from the ontology class hierarchy, or rely on other semantic entailments from the ontology language definition.

4.3 Interactions with suppliers

The travel agency's agent does not handle provisioning of the various elements of the trip itself. It will therefore need to communicate with the various suppliers in order to decide on flights, rail journeys, hotels and so on. It could be the case that the interface to each supplier is a web service, and the agent's job would then be to invoke the web service by fashioning a suitable SOAP [17] call, or whatever the appropriate mechanism is for that service. This can be accommodated in Nuin by either designing a custom web-service action that gets invoked from the script, or by registering a Java object binding that gets invoked by the built-in `invoke service` action. However, for the purpose of this exercise, we assume that the suppliers are also agent-based, and that collaboration becomes a problem of inter-agent communication.

First we note that a similar problem arises between agents as between the client and the travel agency agent. Should the agents invoke actions on the other agents, or delegate an intention or goal? One determining factor may be the need to build a coherent and optimal solution according to the client's preferences. The travel agency agent could determine which of the customer's preferences were relevant to a given subgoal, and pass these to the supplier agent. Indeed, if the client's preferences are available as a Semantic Web source, then (ignoring the important details of security and privacy) the supplier agents could access the client's preferences directly. The potential difficulty here, though, is building a globally optimal solution. Having each supplier agent construct an optimal segment of the journey does not guarantee that the overall solution is optimal. It may well be possible to use inter-agent negotiations among the whole community of stakeholder agents to build a globally optimal solution, but that is not the focus of our current research. Therefore we assume that the travel agency agent sends queries to the supplier agents, and assembles the solution pieces into an overall trip proposal. All negotiations are then pair-wise, with one of the parties always being the travel agency agent. The travel agency agent is solely responsible for optimising the solution.

The FIPA `query-ref` performative seems appropriate for the task of seeking solution elements from the suppliers. But what should the content of the message be? At the beginning of the process, we may know that John wants to travel from Madrid to Washington. We could query all known transportation services providers for routes that originate in Madrid. This, however, would generate many air-routes from Madrid, including those taking John away from the USA, plus road and train journeys to France and Portugal. We could ask for routes starting in Madrid and terminating in Washington DC, which would allow airlines to report their suggested routes (via Paris Charles de Gaulle, for example). Another tactic would be to use the geographic elements of the ontology to test whether a supplier is able to provide a single journey to a given region (e.g. Madrid to the Eastern USA) and use this to prune the search space by querying in more detail only those agents that can provide suitable routes in principle. This tactic may be invoked directly from the agent's script; it may also be invoked by the agent monitoring the responses to queries, noticing a high branching factor in the search space, and adopting an improved strategy. The current version of Nuin does not support this meta-monitoring directly. We will investigate convenient mechanisms for doing so in future versions.

We make the distinction in our ontology between journeys, routes and bookings. Initially, we query the supplier agent for information on routes. A route has a start and end location, distance and vehicle. A given instance of a route may start at Madrid airport and end in Paris Charles de Gaulle, and use an Airbus A320. We can infer that an A320 is an `AirbusPlane` which is an `Airplane`, thus this trip is also in the class `AirTravel` because `AirTravel` is defined as the class that

[3] *Syntax note*: terms with fixed arity are encoded like Prolog terms, with a functor and fixed argument list between parentheses. However, many structured terms have variable numbers of arguments (consider the FIPA message structure). Nuin supports both constructions: a Prolog-like term may be decorated with an additional list of named parameter-value pairs, of the form `functor() {key ~ value, … }`. Unification is extended to unify named arguments as well as positional arguments.

has `vehicleType` of class Airplane. Figure 7 shows a fragment of our ontology class hierarchy (using Protégé [14]):

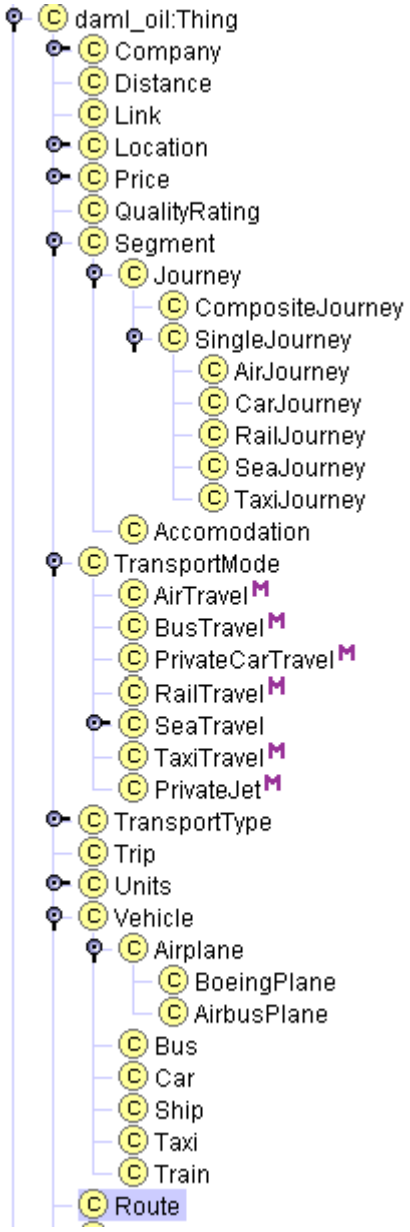

Figure 7: section of ontology class hierarchy

This approach highlights a particular difficulty with ontology development: when to uses classes vs. instances. We can define `A320` as an instance of the class `AirbusPlane`, and for many applications it is sufficient to know that a given route uses an (i.e. some unknown) A320. But for other applications, such as aircraft maintenance or scheduling, we need to know which individual aircraft, so `A320` should be a class, and instances of it would be named by the individual aircraft identifiers. But to define the route, naming the individual plane is incorrect, since different actual planes will fly the route on different days. Using DAML+OIL (or OWL), we can define an auxiliary Route subclass using a restriction:

```
<daml:Class rdf:ID="A320Route">
 <daml:subClassOf rdf:resource="#Route" />
 <daml:subClassOf>
  <daml:Restriction>
   <daml:onProperty
rdf:resource="#vehicleType"/>
   <daml:toClass rdf:resource="#AirbusA320" />
  </daml:Restriction>
 </daml:subClassOf>
</daml:Class>
```

Figure 8: class description for routes that fly A320's

For any `A320Route`, we can infer that the transporting vehicle is an Airbus A320, even if we don't know which one. It is an open question, however, whether the extra complexity introduced by this definition is worthwhile, or whether we should have multiple ontologies (e.g. one for travel and one for maintenance) and a process for translating between them when necessary.

In Nuin, we implement the process of sending the `query-ref` as a `message send` action, followed by a `suspend until` the reply is received. This works for a single communication with another agent. If, however, there are multiple agents involved, a better alternative would be to send a series of messages out, and have plans that trigger on the incoming reply messages. There are two difficulties with the second approach: firstly, enough state has to be asserted into the agent's beliefs (or other KS) to allow the agent to continue developing the plan from that point, and secondly it is harder for the agent to monitor a lack of response from the remote agents and adapt accordingly. We solve the first by assigning each partial trip its own unique identifier, and use the `reply-with` field to relate incoming answers to the results of previous planning. This then generates a set of new, extended partial plans that get new identifiers. For the second problem we do not have a convenient solution. A possible future extension to the Nuin platform will be to include first-class support for the FIPA interaction protocols [3]. Either directly as a result of supporting interaction protocols, or as a result of implementing the necessary supporting code, we hope that a clear and practical solution to the meta-monitoring problem will emerge. Note that, in our opinion, it remains an open question as to whether the ability of PRS-based agent architectures to recurse to meta-level planning is a viable solution to this problem (without creating enough complexity in the agent plan to make it difficult to perform software design and maintenance).

4.4 Reconciling vocabularies

In general, determining the correspondences between two (or more) ontologies is a very difficult task, requiring extensive human intervention [10]. Once the mapping between two ontologies is defined, it is possible that translations between a value expressed in one ontology and a value expressed in another can be automated. Some transformations are fairly straightforward, such as the units conversion (e.g. from km to miles and vice versa).

In a multi-agent system, there is a open question about whose responsibility it is to do ontology conversion. One possibility is for each agent to have a normal form that it uses for its own knowledge representation. Each received sentence would then normalised, using the information from ontology mappings where necessary. This would cope well with allowing communications from agents that used different measurement units, for example, providing that the units themselves are explicit in the ontology. An alternative is that the ontology used by the receiving agent is advertised in a public directory, and it is the originating agent's responsibility to do any necessary

translations before sending a sentence as part of a message. A further alternative is an intermediate position between these two, where the agent community includes translator agents that can handle two-way translations between agents using different ontologies. A version of the intermediary architecture may be needed when providing large semantic web or other legacy information sources into the agent community. It is often impractical to translate the entire information source to a different ontology, but it may well be possible to wrap the information source with a mediating agent that dynamically performs the necessary ontology-based transformations on queries and results. We used this strategy effectively in a project that used DMOZ [1] information in a distributed knowledge-sharing application [4]. Rather than convert the very large DMOZ data set to RDF, it was stored in a custom database layout and queries and query results were dynamically translated to RDF as needed.

Using Nuin, we can define a plan that triggers when incoming messages are received, and use this to check that the message content is in a suitable ontology. If not, it may be a simple action to do the translation locally if the agent is capable of doing so, or the agent may adopt an intention to translate the message content to a suitable ontology. This intention may then be discharged in different ways, for example by sending a request to the translator agent. Once the message is expressed in a known ontology, an event is raised to trigger further processing on the message content.

Our current experiments with the Challenge Problem make the simplifying assumption that the global ontology is shared. This assumption is only valid for such a self-contained exercise. Any realistic scale of application, especially one that uses open semantic web information sources, will be exposed to the ontology reconciliation problem.

4.5 Critiquing and ranking solutions

As the travel agency agent begins to assemble solutions to the client's requested travel goal, it will be faced with a rapidly expanding search space. In order to improve its chances of success, it should choose to pursue only those partial solutions that are promising. If the agent waits until solutions (i.e. travel plans) are complete to critique them, it is likely still to be processing long after the client's patience has run out and they have left the store. This implies that we must be able to critique partial solutions to the problem, and select which ones will be further expanded. We note that planning algorithms have been studied extensively for many years in AI, and it is not our intent in this short paper to revisit the many choices that a planning system can adopt to be able to plan effectively. Pending deeper investigation of this topic, our current design uses a simple forward-chaining means-end search algorithm. As mentioned above, we assign each partial solution a unique identifier. A solution is a series of segments, each of which is either a journey segment or an accommodation segment. The journey segment identifies the route, and may be composed of a series of individual journeys. A segment has an associated cost.

Reviewing the Challenge Problem text, we hypothesise that the following represent typical preferences a client may have over journey segments:

- type vehicle (e.g. Airbus A370)
- cost
- quality rating (first class, business class, economy, five star, etc)
- existence of facilities (TV, Internet connection, smoking rooms, pool)
- preference of mode of transport (fly vs. drive) – which may be conditional on other factors, such as accessibility of airport
- distance to local amenities (sightseeing, ski, beach, etc)

Some of these preferences will be fixed, some context dependent. On a business trip, customers might be less cost-sensitive than on a personal vacation (or vice versa!). In summer, distance from ski resorts is less important than distance from the beach.

We would like to explore making this preference information as widely available as possible, so encoding it as a semantic web resource seems plausible (we ignore for the time being important requirements to do with security and privacy).

One natural approach is to consider the various categories of alternatives that the client might prefer as ontological classes. Thus, a customer who prefers non-smoking hotel rooms has a preference for a room in the class `NonSmoking` over class `Smoking`. A simple way to encode this in the client's profile is shown in fig 9:

```
<Preference>
  <prefer rdf:resource="#NonSmoking" />
  <over rdf:resource="#Smoking" />
</Preference>
```

Figure 9: First attempt at encoding user preferences

```
<Preference>
  <prefer>
    <NonSmoking />
  </prefer>
  <over>
    <Smoking />
  </over>
</Preference>
```

Figure 10: alternative encoding for user preferences

This example uses classes as individuals, so again, exceeds the limitations of OWL DL and OWL Lite. An alternative approach would be to treat the preference arguments as expressions, using RDF blank nodes (bNodes) as existential variables (an interpretation sanctioned by RDF theory). This transforms the preference from fig 9 into fig 10:

The difference between these approaches may be subtle to readers unfamiliar with RDF. In the first encoding (fig 9), the arguments to the preference relation are the classes themselves. In the second encoding, the term `<NonSmoking />` is RDF shorthand for:

```
<rdf:Description>
  <rdf:type rdf:ID="NonSmoking" />
</rdf:Description>
```

that is, an anonymous node of type `NonSmoking`.

To use this second encoding, the agent must match the existential query implicit in the graph to the data at hand. This exploits a feature of RDF (not, it must be admitted, a universally loved feature) that meta-level information can be encoded in the same formalism as the object-level information. The preference query can be seen as expressing a predicate over the proposed solution classes, but is encoded in the same graph structure as the data itself.

By using pair-wise preferences of this kind, whichever approach is adopted, we obtain a partial ordering over sets of solutions. The reified Preference relationship is transitive, so a data source aware of this fact could pre-compute the transitive closure of preferences. Thus, if the client stated their preference was for 5-star hotels over 4-star, and 4-star over 3-star, the transitive closure would allow two proposed segments, one for a

5-star hotel and one for a 3-star hotel to be ranked correctly. Since the ordering would be partial, however, not all solutions could be ranked, so the solution evaluator would need to allow for sets of equally preferred candidates at any one time.

The client should be able to order their preferences, so that the preference for a certain cost band is allowed to dominate over the preference for smoking rooms, or vice versa. This could be achieved by adding a weight to the each `Preference` instance, or allowing preferences that ranked other preferences recursively. It is not clear which, if either, of these choices would work better in practice, and more experimentation is needed.

Again, speculating about the design (we have not yet implemented the solution ranking mechanism), we could encode context-dependent preferences by adding a condition clause to the `Preference` instance. The problem we foresee here is that there is no standard mechanism, *de facto* or otherwise, for encoding general predicates in RDF. Thus any mechanism that allowed the encoding of "if summer-time" on a preference of `NearBeach` over `NearSkiRun` would be dependent on a processor being aware of the encoding scheme used. The choices presented above, assuming that the existence of `Preference` is recognised, stay closer to standard RDF interpretations.

Given that we can achieve a satisfactory encoding of user preferences, we must then incorporate them into the strategy for prioritising the search space. We envisage a plan that is triggered by the asserting of a partial solution into the agent's beliefs KS, and which would rank the new solution against the current unexpanded partial solutions. Thus each partial solution is in one of two states: either it has been selected for expansion, or it has not been expanded yet, but is sorted according to the partial order defined by the user's preferences. It would only be necessary to find the highest ranked plan that has not yet been expanded that is preferred over the new solution, so searching from the front of the candidates list will be effective.

A more open question, and one that we have not yet addressed, is to be able to critique full and partial solutions, rather than just rank them. For example, if the agent was able to determine that a client could save a substantial amount of money by accepting a certain hotel that meets all criteria except having in-room Internet connections, it may be able to propose this to the user. Alternatively, such deductions might form the basis for negotiation strategies that suggest which factors to yield on, and which to stand firm on. This seems to be a fruitful area for future investigation.

4.6 Determining acceptable solutions

Before proposing a solution to the client, the agent must be certain that it has met the client's expressed criteria for the trip. We have not yet stated in this paper how the client's constraints are to be specified. This is in part because we run into limitations of standard ontological languages, since we will need constraints on the literal values of instance properties, and this is not an area that current ontology languages address.

Assuming that we have an appropriate canonicalization of the string form of a date, we can test for equality between two departure dates. But if the client specifies a departure date of "10-July-2003", domain knowledge is needed to recognise that "10-July-2003 10:16" is acceptable. Moreover, the client may actually want specify a departure date of "around the 10th of July" or "between the 4th and the 10th of July".

We may also want to specify that the trip includes a visit to the Statue of Liberty. While we can – just – imagine the creation of a pseudo-class `VisitToStatueOfLiberty`, and subsequently a check that some segment of the trip is subsumed by this pseudo-class, it is hard to see what the definition of the class would be in practice.

We thus currently define the constraints as a list of logical predicates that are interpreted by problem solvers other than the ontology reasoners. However, it remains an interesting area for speculation and future research whether there is a reasonably simple constraint language, that could be combined with a description-logic–like reasoner to give a richer means of checking consistency in candidate solutions.

5. Evaluation and conclusions

We have presented some vignettes of parts of the solution to the OAS'03 Challenge Problem using our BDI agent platform, Nuin. The key goal in the Challenge call for papers is to explore how agents would actually use ontological information. Much of the foregoing discussion represents our design thinking, since we have only begun to build the complete solution.

Our agents are strongly knowledge-based, and use logical sentences and mental attitudes for their internal modelling. Ontological information is clearly useful compactly describing the domain of discourse (especially if the same ontology is shared with other agents), and allows the agent to use class and property hierarchies to generalise and specialise queries and results.

Given our interest in building agents for the semantic web, we have restricted ourselves to the common semantic web ontology languages: DAML+OIL and OWL. Both of these languages' designs are based on description logic (DL) reasoning. The use of description logic reasoners in practical agent applications is not a widely explored topic, due in part to a limited availability of DL reasoners. More such reasoners are now becoming available, and we can expect more research into this area in future. A key component of the description logic approach is *class description*, and we have shown above a few instances of using class descriptions in the agent's reasoning. Using class descriptions and a meta-level `prefers` predicate to encapsulate the client's preferences appears to be a useful way to make those preferences available to a wider range of semantic web services. The limitations of description logic sentences, however, suggest that richer representations will need to be developed to encode a broadly useful sub-set of the client's general preferences.

While we have shown the use of ontology information by BDI agents, both as additional open knowledge sources for the agent to access, and as additional entailments that the agent reasoners can draw upon, we nevertheless feel that this is only a preliminary account of the integration of these two areas. Further practical experiences will help to resolve this, and we continue to develop a complete implementation of the Challenge Problem in the Nuin framework. We also look forward to the development of theoretical treatments of the interactions between the principles of deliberative agents and the principles of description logics.

6. ACKNOWLEDGEMENTS

We would like to thank the anonymous OAS'03 reviewers for their detailed comments on the original version of this paper. Due to a short deadline and a lack of space, we have not been able fully to address all of their comments, but we hope to do so in future publications. Thanks also to Dave Reynolds of HP Labs for his comments and suggestions.

7. REFERENCES

1. *ODP - The Open Directory Project.* http://www.dmoz.org
2. *The DARPA Agent Markup Language (DAML+OIL)*. 2001.
   Web site: http://www.daml.org
3. *FIPA Interaction Protocol Specifications*. 2003. http://www.fipa.org/repository/ips.php3
4. Banks, Dave, Cayzer, Steve, Dickinson, Ian, and Reynolds, Dave. *The ePerson Snippet Manager: a Semantic Web Application.* (HPL-2002-328) HP Labs Technical Report. 2002.
   Available from: http://www.hpl.hp.com/techreports/2002/HPL-2002-328.html
5. Bellifemine F., Poggi A. & Rimassa G. "Developing Multi Agent Systems With a FIPA-Compliant Agent Framework". *Software Practice and Experience.* Vol. 31:2. 2001. pp. 103–128.
6. Berners-Lee, Tim, Hendler, James, and Lassila, Ora "The Semantic Web". *Scientific American*. 2001.
7. Corcho, O., Fernandez-Lopez , M., & Gómez-Pérez , A. *An RDF Schema for the OAS Challenge Problem*. 2003. http://oas.otago.ac.nz/OAS2003/Challenge/MadridTravelOntology.rdfs
8. Dean, Mike, Schreiber, Guus, van Harmelen, Frank, Hendler, Jim, Horrocks, Ian, McGuinness, Deborah L., Patel-Schneider, Peter F., and Stein, Lynn Andrea . *OWL Web Ontology Language Reference*. 2003. http://www.w3.org/TR/owl-ref/
9. Dickinson, I. & Wooldridge, M. "Towards Practical Reasoning Agents for the Semantic Web". In: *Int. Conf. on Automomous Agents and Multi-Agent Systems (AAMAS'03).* 2003. pp. to appear.
10. Dou, D., McDermott, D., & Qi, P. "Ontology Translation by Ontology Merging and Automated Reasoning". In: *Proc. EKAW Workshop on Ontologies for Agent Systems*. 2002. pp. 3-18. http://cs-www.cs.yale.edu/homes/dvm/papers/DouMcDermottQi02.ps
11. FIPA. *FIPA ACL Message Structure Specification.* (XC00061) 2000. http://www.fipa.org/specs/fipa00061/
12. FIPA. *Abstract Architecture Specifiation*. 2002. http://www.fipa.org/specs/fipa00001/
13. HP Labs. *The Jena Semantic Web Toolkit*. 2002. http://www.hpl.hp.com/semweb/jena-top.html
14. Noy N. F., Sintek M., Decker S., Crubezy M., Fergerson R. W. & Musen M. A. "Creating Semantic Web Contents With Protege-2000". *IEEE Intelligent Systems*. Vol. 16:2. 2001. pp. 60-71. http://www-smi.stanford.edu/pubs/SMI_Reports/SMI-2001-0872.pdf
15. Rao, A. "AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language". In: *Proc. 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW '96)*. Springer-Verlag, 1996. pp. 42–55.
16. Rao, A. & Georgeff, M. "BDI Agents: From Theory to Practice". In: *Proc. First Int. Conf on Multi-Agent Systems (ICMAS-95)*. 1995.
17. W3C. *Simple Object Access Protocol (SOAP) 1.1*. 2000. http://www.w3.org/TR/SOAP/

Guidelines for Constructing Reusable Domain Ontologies

Muthukkaruppan Annamalai
Department of Computer Science & Software Engineering
The University of Melbourne
Victoria 3010, Australia
mkppan@cs.mu.oz.au

Leon Sterling
Department of Computer Science & Software Engineering
The University of Melbourne
Victoria 3010, Australia
leon@cs.mu.oz.au

ABSTRACT

The growing interest in ontologies is concomitant with the increasing use of agent systems in user environment. Ontologies have established themselves as schemas for encoding knowledge about a particular domain, which can be interpreted by both humans and agents to accomplish a task in cooperation. However, construction of the domain ontologies is a bottleneck, and planning towards reuse of domain ontologies is essential. Current methodologies concerned with ontology development have not dealt with explicit reuse of domain ontologies. This paper presents guidelines for systematic construction of reusable domain ontologies. A purpose-driven approach has been adopted. The guidelines have been used for constructing ontologies in the Experimental High-Energy Physics domain.

1. INTRODUCTION

The World Wide Web has become the *de facto* medium for distributed user community to share digital information, posing a challenge for users to effectively utilise the accessed information. The next generation agentised web promises to dispense with some of the human effort through machine processable metadata linked to ontologies. The working definition of ontology is a specification of shared conceptualisation [4]. An ontology presents a shared understanding of how the world is organised in a particular aspect of the domain and specifies the meaning of terms that make up the vocabulary in the domain of discourse – a necessity for information access and interoperability.

Constructing ontologies from scratch to support domain applications requires a great deal of effort and time. Alternatively, reusable domain ontologies provide opportunities for developers to exploit and reuse existing domain knowledge to build their applications with much ease and reliability. A common belief is that reusable ontologies ought to be conceived and developed independent from application and context of its use. The consequence of adhering strictly to this notion is that the developed reusable domain ontologies are: **a)** usually over-generalised and omit useful knowledge; **b)** often are also sparse constructs because it is not easy to determine which part of the concrete domain knowledge can be reused, particularly when the capturing of the domain knowledge is attempted in a top-down fashion; and **c)** necessitates modification and considerable extension work before it can be utilised. A better alternative is to be able to develop reusable ontologies without over-compromising their usability in the domain.

The approach we propose is to first ask which kind of domain knowledge should be encoded; bring together the relevant pieces of knowledge; only then identify which of those pieces of knowledge can be reused and isolate them. This paper outlines a strategy to develop reusable domain ontologies in this manner. We will discuss this issue in the context of a case study to develop reusable Experimental High-Energy Physics (EHEP) ontologies for the Belle collaboration *(http://belle.kek.jp/belle/)*. The distributed scientific community collaboratively sets-up experiment, accumulate event data, generate simulation data, construct software tools to analyse the data, carry out data analysis, publish their findings, and progressively build on each other's work. We aim to show that suitable ontologies be developed and reused to facilitate this scientific community to produce and share information effectively on the agentised web [1, 2]. The reusable domain ontologies will serve as a basis for communication, integration and sharing of information pertaining to experimental analysis within the collaboration.

The paper is organised as follows. In Section 2, we sketch out our basic strategy for developing reusable domain ontologies. This strategy is further elaborated and illustrated in Sections 3 and 4. The development guidelines are also given in here. Finally, in the concluding section we summarise the contribution of this paper.

2. STRATEGISING THE DEVELOPMENT OF REUSABLE DOMAIN ONTOLOGIES

We have devised a strategy for developing reusable domain ontologies by putting together the key points in traditional and modern ontology development methodologies and principles, such that of [3, 4, 5, 8, 9, 10]. The notable features in these methodologies will be pointed out as we advance through this paper.

There are two types of ontologies in our environment, namely, *domain* and *purposive* ontologies. A domain ontology captures an area of the domain generically, and serves as a reusable knowledge entity that can be shared across this domain. The domain ontologies are loosely coupled to one another, reflecting the association between the different facets of the domain captured by the respective ontologies. On the other hand, a purposive ontology explicitly defines the terms for supporting specific purpose or use. The purposive ontology encodes specialised domain knowledge by composing various reusable domain ontologies and then affecting the necessary application-specific extensions.

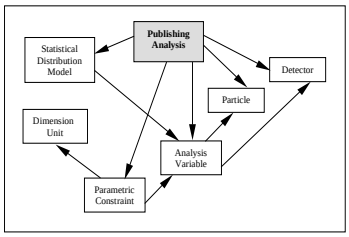

Figure 1: Interconnected Ontologies

The ontological commitment required to support a particular need is embodied in a set of reusable domain ontologies as illustrated in Figure 1. The filled box denotes the purposive ontology modelled for publishing EHEP analysis. Each empty box represents generic domain ontology. An arrow points at the ontology that holds the definition of the referred terms. A generic ontology is linked to another via concept relations. This links depicts the dependencies between the ontologies in this conceptualisation of the domain. This kind of loose coupling allows scalable modifications of the domain ontologies.

The value of reusable ontologies is long recognised by ontology researchers. The research on the technology for supporting knowledge sharing and reuse originated in Stanford's Knowledge Systems Laboratory (KSL) [7]. It spurred the development of reusable, general ontologies (such as Ontology of Time, Money, Measure, etceteras) for the KSL's Ontolingua server. The design principles espoused in [4] were commonly used to develop these ontologies that are meant to be shared across different knowledge domains.[1] While the past attempt was concerned with the development of domain-independent single ontologies, we focus on constructing inter-depending ontologies that are small and easier to reuse in a particular domain.

There appears to be no clear methodology for building domain ontologies that can be reused with minimal extensions. Moreover, a disagreeable prospect arises from the conventional idea that reusable domain ontologies ought to be conceived and developed independent from their application, due to the following reasons: **a)** Unlike general knowledge, which ground out in primitives with assumptions that are ordinarily understood, domain knowledge deals with domain-specific jargon; **b)** It is not feasible to think of knowledge needs of all foreseeable domain applications; **c)** The intended meaning of some terms can be different according to context of application; and **d)** It is also impractical for knowledge engineers to describe all knowledge they know about the domain. Hence, it is difficult to see how it would be possible for knowledge engineers to rely on their intuition alone to build ontologies that adequately capture the reusable knowledge in their domain.

Given this situation, we maintain that it is only sensible if we allow our current needs to dictate the creation of usable domain ontologies that are also reusable.[2] Modelling according to the purpose and use helps to determine what features of the domain knowledge should be encoded and provide a focus for knowledge acquisition. Consequently, we advocate development of purposive ontologies, and simultaneously pursue the creation of reusable domain ontologies.

Our basic strategy is spelt out as follows: **a)** Adopt a bottom-up view of the domain to conceptualise the knowledge required to support a specific need, and build the conceptual model; **b)** Identify potentially reusable chunk in the conceptual model and generalise it; and **c)** Formalise the generalised domain model into reusable domain ontology. In what follows, we will elaborate this strategy and illustrate it using simple examples.

3. PURPOSE-DRIVEN CREATION OF REUSABLE DOMAIN ONTOLOGIES

We begin by modelling the purposive ontology, which defines the vocabulary for the purpose of describing the experimental analysis in collaboration documents, such as research notes and research papers. The ontology can be applied for document annotation, query-answering and information retrieval.

3.1 Modelling the Purposive Ontology

We set out to accomplish this task by creating the concept model, which will serve as the foundation for the purposive ontology. Individuals in this conceptualisation are typically defined as concepts, and are constrained by properties, relations and axioms. The taxonomic and cross relationships among the concepts are explicitly specified. We used Protege-2000 *(http://www-protege.stanford.edu)*, a frame-based modelling tool to construct the concept model. Figure 2 describes a partial hierarchy of top-level concepts in the model that was developed to conceive this purposive ontology.

The model is elaborated from scientific collaboration documents, mainly books, research papers, existing standard HEP terminology and consulting the EHEP physicists. Our initial discussion with the EHEP physicists and related literature review enabled us to recognise some of the distinct concepts required for describing a typical EHEP experimental analysis. They are Signal event, Background event, Kinematic variables, Topological variables, Particle, etceteras. We called them the hook-concepts, as they serve as hooks (or links) for structuring additional concepts into the concept model. Initially, our competency questions [5], that are the questions we want the ontology to answer, revolved around these hook-concepts.[3] Subsequently, new concepts affiliated with the existing concepts emerge, which are organised in the hierarchical model by bottoming-up and middling-out processes [10]. The internal structures of these concepts are defined to limit their possible interpretation and relations in

[1]Clarity, Coherence, Extendibility, Minimal encoding bias and Minimal ontological commitment are prescribed as suitable design principles.

[2]As a matter of fact, the modern ontology development methodologies [8, 9] follow the tradition of knowledge engineering and back the development of application-oriented ontologies.

[3]In essence, the competency questions identify the kind of domain knowledge that should be encoded. Examples of competency questions are: What are the kinematic cuts performed in $B \rightarrow \rho\pi$ event analysis? What are the suppressed background events?

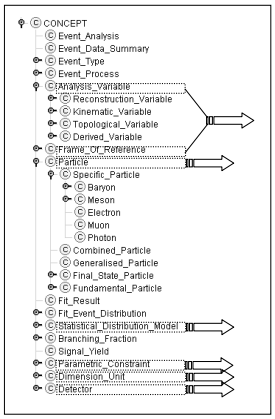

Figure 2: Externalising Reusable Domain Models

the ontology. This cycle is continued until the model is satisfactorily developed, that is when the set of compiled competency questions and their related answers can be clearly represented using the terms defined in the ontology. Each cycle evolves the model closer to the desired form.

3.2 Abstraction of Reusable Chunks

The concepts in the model are distinctly organised according to their role in elucidating particular aspects of the domain. As a rule, homogeneous concepts tend to cluster under a common parent concept and are potentially reusable in another situation (in whole or part). The reuse potential of these clustered concepts is reflected by their coherent nature.[4] The task of the knowledge engineer is to examine each cluster of concepts to identify the chunk that has reuse potential. The reusable chunk is isolated from the model and generalised into an independent reusable domain model. For example, the definition about *Particle* and its sub-classes in the concept model (shown in Figure 2) can be externalised from this model and componentised as reusable knowledge entity. Using this simple technique, we generated all the domain ontologies supporting the purposive ontology for describing experimental analysis as depicted in Figure 1.

Ontology Development Guideline I. We summarise the operational guidelines for developing purposive and domain ontologies (that does not reuse existing ontologies) as follows: **a)** Specify purpose and uses of ontology; **b)** Identify hook-concepts; **c)** Formulate competency questions; **d)** Identify new terms required to precisely formulate the competency questions and their generated answers; **e)** Define the new terms (concepts, properties, relations and axioms). Structure the concept into the concept model; **f)** Evaluate concept model against the set of competency questions and make the necessary changes; **g)** If still not satisfied with the level of details in the model, return to *step [c]*; **h)** Analyse concept clusters in the model that particularises specific areas of the domain. If a group of related concepts has reuse potential, generalise and shape them as a separate reusable domain model (sub-model); **i)** Link back the sub-models to the main model. Make the necessary application-specific extensions to the incorporated generalised domain knowledge; and **j)** Formalise the main model and the sub-models into purposive ontology and reusable domain ontologies, respectively. The ontological terms and definitions are expressed formally in a web-ontology specification language, such as DAML+OIL *(http://www.daml.org)*.

Future domain applications can exploit these reusable domain ontologies and may even churn out new reusable domain ontologies. This matter is discussed in the next section.

[4]The coherency among a set of concepts can be appropriately assessed by performing an ontological analysis [6] involving the concepts and relations in the model.

4. THE REUSE OF DOMAIN ONTOLOGIES

Using an existing domain ontology for supporting another application or to serve as a basis for building another ontology is cost-effective, provided the reusable ontology does not require much customisation effort. We describe two such scenarios that illustrate the reuse of existing domain ontologies and the creation of new versions of existing ontologies.

Purpose 1: Supporting Analysis Specification. We envision applications that allow physicists to partially automate experimental analysis such as skimming, tracking, vertexing and particle reconstruction. As a result, the vocabulary to describe the rudiments of these low-level analyses will need to feature in the purposive ontology built to support these applications.

We begin by sketching the concept model of this purposive ontology. The preliminary study shows that the purposive ontology will provide the vocabulary to describe the EHEP analysis, as in the previous case (refer to Section 3), but in greater detail. In particular, there will be additional analysis variables to be considered, and requires a much 'finer' representation.

All the domain ontologies created earlier are candidates for reuse. However, the *Analysis Variable* ontology will be revised to cater for this new requirement. Low-level analyses also involve tracks and clusters associated with event particles. Like *Particle* ontology, the knowledge about tracks and clusters can be held as separate entities. As we do not anticipate changes to the other existing ontologies, the purposive ontology being developed can make use of those reusable domain ontologies directly.

This developmental activity has generated two new domain ontologies, namely *Track* and *Cluster*; and a new version of *Analysis Variable* ontology, extended from its earlier version (see Figure 3). The distinction between the two versions of the *Analysis Variable* ontology can be made on the basis of additional definitions used to characterise the conceptualisation in the later version. Since no alteration to the existing definitions in this ontology was made, the new version is indeed backward compatible. Our rationale for constructing a new version of this ontology is to ensure that the prospective users who have adopted the original version can continue to rely upon their ontology, and not be overwhelmed with abstraction unrelated to their need.

Purpose 2: Supporting Detector Description. Another planned application aims to provide information about the Belle detectors. The purposive ontology developed for

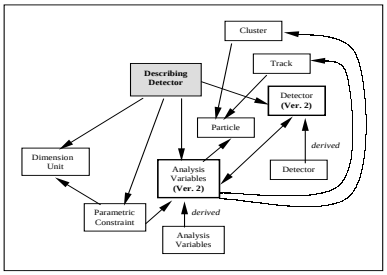

Figure 3: Reusing Existing Domain Ontologies

this application will be used to build a knowledge base about detectors' particle identification capabilities.[5]

The existing *Detector* ontology merely describes the detector and regions where the presence of tracks and clusters are sensed and is referred to by definitions in *Analysis Variable* ontology. The new requirement necessitates definitions in Detector to refer to definitions in *Analysis Variable* instead (see Figure 3). The new version of *Detector* ontology will have to provide the vocabulary to describe the Belle detectors in greater details, including *inverse* relationships between definitions in *Detector* and *Analysis Variable*.

Ontology Development Guideline II. We now recapitulate the operational guidelines for developing purposive and domain ontologies by reusing existing domain ontologies: **a)** Specify purpose and uses of ontology; **b)** Sketch the model of the purposive ontology; **c)** Identify existing reusable domain ontologies that can be used to support the modelling process; **d)** Construct the unsupported portion of the model based on *Guideline I steps [b] - [g]*; **e)** Identify reusable region in the model. If found, convert the related concepts into an independent reusable domain model. It corresponds to *Guideline I step [h]*; **f)** Select reusable ontologies (identified earlier) and make necessary changes to accommodate application needs. Sometimes, the modified ontologies may be redeveloped as new versions of the existing ontologies; and **g)** Rebuild the model of the purposive ontology by linking the sub-models, and formalise the ontologies. It is similar to *Guideline I steps [i] and [j]*.

5. CONCLUDING REMARKS

We have presented guidelines for creating reusable domain ontologies for a scientific user-community. In our environment, the different domain ontologies organise and structure the knowledge of diverse parts of the domain. The domain ontologies are constructed as small reusable knowledge components that can be easily shared across applications. These ontologies are loosely coupled to one another reflecting their association in the real world.

The conceptualisation is guided by a common set of competency questions generated when application-dependent purposive models are conceived. This provides the reason for believing that the same kind of reusable chunks will emerge from the different purposive models, even if the order in which the applications are considered is varied. The concepts in the domain ontologies are captured in some generality to make reuse possible. We are consistent with making specific knowledge more generic.

Sometimes it would be necessary to allow co-existence of different versions of a domain ontology to accommodate different needs in the domain. For example, in Section 4 we exemplified the creation of new versions of existing ontologies. These versions are seen as distinct domain ontologies with dissimilar reuse potential. Herein lays the larger issue of the management of the domain ontologies. A mechanism to control the different versions of domain ontologies is essentially required.

Acknowledgments: We are grateful to our associates in the Physics department, particularly Glenn Moloney and Lyle Winton for providing insights into EHEP experimental analysis.

6. REFERENCES

[1] M. Annamalai, L. Sterling, and G. Moloney. A collaborative framework for distributed scientific groups. In *Proceedings of AAMAS'02 Workshop on Ontologies in Agent Systems*, 2002.

[2] L. Cruz, M. Annamalai, and L. Sterling. Analysing high-energy physics experiments. In *Proceedings of AAMAS'02 Workshop on AgentCities*, 2002.

[3] M. Fernandez, A. Gomez-Perez, and N. Juristo. Methontology: From ontological art towards ontological engineering. In *Proceedings of AAAI'97 Spring Symposium on Ontological Engineering*, 1997.

[4] T. Gruber. A translation approach to portable ontologies. *Knowledge Acquisition*, 5(2):199–220, 1993.

[5] M. Gruninger and M. S. Fox. Methodology for the design and evaluation of ontologies. In *Proceedings of IJCAI'95 Workshop on Basic Ontological Issues in Knowledge Sharing*, 1995.

[6] N. Guarino and C. Welty. Ontological analysis of taxonomic relationships. In *Proceedings of ER'00 Conference on Conceptual Modelling*, 2000.

[7] R. Neches, R. Fikes, T. Finin, T. Gruber, T. Patil, R. Senator, and W. R. Swartout. Enabling technology for knowledge sharing. *AI Magazine*, 12(3):16–36, 1991.

[8] A. T. Schreiber, J. M. Akkermans, A. A. Anjewierden, R. Dehoog, N. R. Rhadbold, W. V. D. Velde, and B. J. Wielinga. *Knowledge Engineering and Management - The Common*KADS *Methodology*. University of Amsterdam, 1998.

[9] S. Staab, R. Studer, H. P. Schnurr, and Y. Sure. Knowledge process and ontologies. *IEEE Intelligent Systems*, pages 26–34, Jan/Feb 2001.

[10] M. Uschold. Building ontologies: Towards a unified methodology. In *Proceedings of the British Computer Society Specialist Group Conference on Expert Systems*, 1996.

[5] An example piece of knowledge may look like this: Aerogel Cerenkov Counter discriminates Kaon over Pion (event variable) with 93% efficiency for momentum greater than $700 MeV/C$.

CO_3L: Compact O_3F Language

Pedro Ramos
ADETTI / ISCTE
"We, the Body and the Mind" Group
Av. Forças Armadas, Lisboa, Portugal
+351 217 903 099
Pedro.Ramos@iscte.pt

Luis Botelho
ADETTI / ISCTE
"We, the Body and the Mind" Group
Av. Forças Armadas, Lisboa, Portugal
+351 217 903 099
Luis.Botelho@iscte.pt

ABSTRACT

This paper presents CO3L, a compact, expressive and easy to use language for ontology representation. CO3L reflects the O3F model of ontology representation proposed elsewhere.

The proposed language enables the representation of basic ontological entities, their relationships, and arbitrary axioms of the domain. Ontological entities include classes, properties, methods, facets and types. Relationships include n-ary associations and inheritance. Axioms may be used to capture complex constraints and relations between entities, to define relational and functional methods, and to represent the effects of the execution of action methods in the world.

CO3L is based on the language of the first order logic, with explicit world states; the relational operator *State/1*, which is true of world state designators; the functional operator *Do/2*, which returns the world state designator resulting of the execution of an action in a given world state; and the modal operator *Holds/2* which is true of propositions satisfied in given world states.

1. INTRODUCTION

In open agent systems it is important to ensure interoperability at the several levels of the so-called communication stack. One such level pertains ontology technology and its uses. Often, agents need to dynamically use ontological information for the most diverse tasks, including service discovery [2] and message translation. Unfortunately there is no consensus regarding the requirements of ontology services for open agent systems. Some need just a simple way of describing the classes of the domain. Others need more than classes; they need to know the kinds of actions in a given domain. Yet others need detailed descriptions of the existing relations between input and output parameters of dynamic computations, or even of descriptions of the effects of performing domain actions.

Each of the above requirements has been handled to different degrees by the most common approaches to ontology representation and use. This has resulted in a heterogeneous landscape of ontology technologies that has not contributed to improve interoperability.

One way to go about this problem is by developing general, more abstract frameworks of which, the existing approaches can be seen as particular cases. This paper contributes to the advancement of one such a framework – O3F, the object-oriented ontology framework.

O3F [10], is a general framework for ontology representation that can be used to capture ontological information originally described in several formalisms including DAML+OIL [4], OWL [11], Ontolingua [5] and OKBC [3]. Additionally, O3F has several advantages in relation to these other approaches, namely the possibility of representing arbitrary axioms, which can only be found in Ontolingua and OKBC, and the possibility of declaratively representing action methods and their effects in the world, which is not offered by any other approach. Besides, [10] presents a significant advance in terms of the formal specification of several kinds of translation relations between ontologies, which were based on the concept of basic mapping.

Given the above, O3F is a natural candidate for ontology representation in web and agent based applications. However, the ontology representation language proposed in [10] uses long cumbersome expressions. This problem is a considerable handicap of the O3F framework because it will impair its effective use and dissemination in the agent and web communities.

The difficulty of using and understanding the proposed representation language derives from two main facts. First, it was directly shaped from the UML class diagram representing the O3F ontology representation model. Second, it straightforwardly adapts constructs used in the agent content language proposed in [1], which are not necessarily the best for an ontology representation language.

This paper presents a solution for the difficulties encountered in the ontology representation language proposed for the O3F representation framework, by attacking the two mentioned problems. The proposed solution preserves the full expressiveness of the language while drastically improving its readability. Furthermore, the representation power of the original O3F framework is also not affected.

Section 2 briefly describes the O3F original approach with special emphasis on the originally used representation language. Section 3 describes our approach to improve the readability and usability of the representation language. First, we show how it is possible to deviate from the original language constructs used in [10] without losing generality or expressiveness. Then, we present CO3L, the new ontology representation language for O3F. Section 4 analyses our approach in the scope of related work. Finally, section 5 presents conclusions and future work.

2. O_3F FRAMEWORK

O3F [10] is a general object oriented framework for ontology representation. An ontology is composed of a set of basic entities, simple relationships between them and arbitrary domain-dependent axioms. Basic entities are classes, properties, facets, methods and types.

As in DAML+OIL, properties, methods and axioms have autonomous existence separated from the classes with which they might be related. The independence of properties from the classes with which they may be related enables O3F to capture ontologies originally represented in DAML+OIL and OWL.

The possibility to represent arbitrary axioms using first order logic enables O3F to capture ontologies originally represented in Ontolingua. Finally, the possibility to represent procedural methods is an advantage of O3F over their direct competitors.

Using the approach described in [10], for instance, it is possible to specify the class named Restaurant, the properties named *Name* and *Price*.

```
(instance (Class :name Restaurant) Class)
(instance (Property :name Name) Property)
(instance (Property :name Price) Property)
```

It is also possible to associate the attributes *RestauratName* and *MaxPrice* to the class *Restaurant*. Originally, in O3F, associations between classes were represented through instances of the special class *O3FRelation. O3FRelation* is an O3F class used in the language to represent all the O3F relations (e.g., Archetype_Attribute, Argument_Attribute).

The class *O3FRelation* has the attributes *name*, which is the name of a specific relation; *arguments*, which is the set of archetypes that are associated in a given relation; and *attributes*, which is the private attributes that belong to a relation. Each element in the set of arguments of a relation is an instance of the class RArgs (Relation Arguments), which is composed of the class of the archetype being associated (e.g., *Class* or *Property*), the key attribute of the archetype being associated (e.g., *name*) and the name of that archetype being associated (e.g., *Restaurant*). The following statement says that the property named *Name* is associated with the class *Restaurant*.

```
(instance
   (O3FRelation
    :name Archetype_Attribute
    :arguments (set
      (RArg
       :class Property
       :key name
       :object Name)
      (RArg
       :class Class
       :key name
       :object Restaurant))
    :attributes (set
      (Attribute
       :name Name
       :value RestaurantName)))
   O3FRelation)
```

An association between an archetype and a property defines an attribute of the archetype. The above association between the property *Name* and the Class *Restaurant* defines the attribute *RestaurantName* of the class *Restaurant*.

The example discussed in this section makes it clear that the specification of an ontology using the proposed language is a cumbersome process which results in a large set of long, difficult to read expressions. Since this process may have to be handwritten, it is better to make it easier and shorter. The next section shows our proposal to achieve this goal.

3. THE NEW O_3F LANGUAGE

In this section we show how the O3F class architecture must be re-interpreted before it can be used as the basis for the design of the ontology representation language.

3.1 Representing O_3F Basic Entities

Class, Property, Method, Type, Facet and other entities are classes of the O3F class diagram. Particular domain classes, properties, methods, types and facets are instances of those classes. Using the language proposed in [10], those instances are represented through the two-place relation *instance*. Here we propose to use one reserved word for specifying instances of each of these classes. The relations *Class*, *Property*, *Method*, *Type*, *Facet,* and *Axiom* will be used to specify the classes, the properties, the methods, the types the facets and the *Axioms* of the ontologies.

The statement (Ontology RestaurantOntology 'Agentcities Consortium') asserts the fact that RestaurantOntology is an ontology whose author is the Agentcities Consortium. The statement (Class Restaurant RestaurantOntology) means that Restaurant is a class of the RestaurantOntology). As another example, the statement (Property Price RestaurantOntology) means Price is the name of a property of the ontology RestaurantOntology. From now on the ontology will be omitted from the arguments list of the language statements, so that the explanations can stay focused.

3.2 O_3F Class Diagram Re-Cast

The main problem with the interpretation of the O3F framework is the way associations are represented. Originally, all associations were represented using the special purpose class *O3FRelation* (see section 2). This policy, together with the way instances were specified using the two-place predicate *Instance/2* lead to lengthier, more complex and cumbersome expressions.

Here we propose to use one specific relation for each of the possible associations of the model. For example, in order to represent the attribute *RestaurantName* of the Class Restaurant it is not necessary to have a language we use the attribute statement (Attribute Restaurant RestaurantName), which contrasts with the long expression in section 2.

3.3 Arbitrary Axioms

Besides the relations specified in sections 3.1 and 3.2, it is also necessary to define a set of primitive operators used in the definition of arbitrary axioms.

Often the declaration of classes, properties, methods and relationships between them is not enough to capture the relations of the domain.

In order to represent arbitrary axioms in first order logic, it is necessary to have the usual logical connectives and quantifiers. This is not new when compared with other ontology representation languages such as KIF [6] in Ontolingua.

In order to define the effects of action methods (i.e., methods whose execution changes the state of the world), it is necessary to talk about states of the world, and to capture the execution of methods in the world along with the changes they produce.

We introduce the following operators:

State/1 is a relational operator used to declare an identifier for a world state. State(<State Identifier>) means <State Identifier> is a state identifier.

Holds/2 is a modal operator used to say that a given proposition holds in a given state of the world. Holds(<Proposition>, <State Identifier>) means <Proposition> is true in the state of the world identified by <State Identifier>. In the current approach, *Holds/2* is a modal operator instead of a relational operator as in [10], in order to avoid the reification of relations, logical operators and quantifiers to functions.

Do/2 is a functional operator used to represent the state that results of the execution of a given method in a given state of the world. Methods are represented by terms called action identifiers. Do(<Method>, <State Identifier>) represents the sate of the world that results of the execution of the method <Method> in the state identified by <State Identifier>.

Given these operators it is possible to represent the effects of the execution of action methods and it is also possible to represent constraints over states of the world. The axioms written below represent the method BookFlight used to book a plane ticket in a travel agency. BookFlight takes the flight number (f), the date of departure (d), the flight class (c) and the identification of the client (x).

$\forall s, f, d, x\ \exists t\ State(s) \wedge Holds(\neg(t \in Ticket) \wedge x \in Client \wedge <c , f> \in Flight\_Classes, s) \Rightarrow$
  Holds($t \in Ticket \wedge <x , t> \in Client\_Ticket \wedge$
  $<t , <c , f >> \in Flight\_Classes\_Ticket \wedge t.Date = d,$
  Do(Ticket.BookFlight(f, d, c, x), s))

There is a t for which if it is not a ticket in situation s, it becomes a ticket ($t \in Ticket$) in the situation resulting of booking a ticket.
Moreover, the booked ticket becomes associated with the client specified in the reservation ($<x , t> \in Client\_Ticket$); the booked ticket becomes associated with the pair of the class and flight specified in the reservation ($<t, <c , f >> \in$ Flight_Classes_Ticket); and the date of the ticket becomes the date specified in the reservation (t.Date = d).

The axioms of the ontology are expressions in which the logic and other general-purpose symbols such as quantifiers and reserved operators belong to the language, and the domain symbols such as classes (e.g., Ticket) and associations (e.g., Client_Ticket, Flight_Classes) must belong to the ontology.

Here, we have used the notation commonly used in textbooks about logic. This is not the actual syntax of CO3L, since CO3L is an abstract language (section3.4).

3.4 Compact O_3F Language

This section presents examples of the abstract syntax of the O3F Representation Language. Basically, this is a first order logic, with modal operator *Holds/2*, and a set of reserved relational and functional operators. In [8] we present a simple ontology example using the UML notation. The example is a fragment of the Travel Agent scenario described in the OAS'03 Challenge Problem.

The provided abstract syntax defines an abstract language that may be instantiated through a variety of concrete syntaxes such as the S-Expression, the usual prefix first order logic syntax, and an XML syntax commonly used in web applications. The top-level symbol representing Compact O3F Language statements is OntologicalProposition, which can be an ontology declaration, an ontological entity declaration, an ontology relationship, an arbitrary axiom[1], or an instance definition.

Frame Name	OntologicalProposition	Abstract
Kind of	CO3LExpression	
Description	Top level language statement	

The frame OntologicalProposition is an abstract concept in the sense that it cannot be instantiated. OntologicalProposition is a CO3LExpression, thus the slot **kind-of** is CO3LExpression, which is the top most CO3L expression.

Frame Name	EntityDeclaration	Abstract
Kind of	OntologicalProposition, AtomicProposition	
Description	Declares the existence of ontological entities such as classes, properties, methods, types	

An entity declaration is also abstract. Only its sub-frames (e.g., ClassDeclaration, PropertyDeclaration, MethodDeclaration) can have concrete instances.

Frame Name	OntologyRelationship	Abstract
Kind of	OntologicalProposition, AtomicProposition	
Description	Declares the existence of a relationship between two or more ontological entities (e.g., the relation between a property and its type)	

Simple ontological relationships are represented through atomic propositions. That's why the frame OntologyRelationship is also a kind of AtomicProposition.

The full language description involves several other frames. However, space constraints preclude their presentation. The complete specification can be found in [7].

The presented abstract grammar enables the definition of diverse concrete syntaxes as appropriate for the application at hand. Our agents have been using S-Expression syntax, but others can also be defined, in particular XML syntax.

[1] An axiom is certainly a relationship between entities of the ontology. This distinction highlights the difference between the simple relations usually captured in ontology representation languages, and the complex relationships that can be expressed by axioms such as the one in this section.

4. RELATED WORK

In the same vein as RDF / RDF Schema [12] and DAML+OIL [4], CO3L is also an abstract language for which several concrete syntaxes may be defined including S-Expression and XML syntaxes.

As in DAML+OIL [4] and OWL [11], properties have autonomous existence outside the scope of classes. This allows defining hierarchic and other relations between properties. The same holds for methods, which may also exist outside the scope of classes.

Besides a predefined set of facets, CO3L allows the definition of new facets if desired. As far as we are aware of, only Ontolingua [5] and OKBC [3] have this feature.

Unlike other well-known ontology language, CO3L allows the declaration of methods, their arguments and return values (in case of functional methods). This feature has been proposed has a desired extension to DAML+OIL in [9] but was never officially integrated as part of DAML+OIL. Ontolingua allows the definition of functions and relations but can't associate them to classes.

As with Ontolingua [5] and OKBC [3], CO3L allows the definition of arbitrary axioms. However, unlike them, CO3L allows the definition of action methods (i.e., methods whose execution changes the state of the world) through state constraint and state change axioms. This is possible only because CO3L has a set of operators used to capture world state and world state changes.

5. CONCLUSIONS AND FUTURE WORK

This paper proposes a compact version of the ontology representation language of O3F. Since O3F is a reasonably comprehensive ontology representation framework capable of representing ontologies originally expressed in a variety of other frameworks such as DAML+OIL, OWL, and Ontolingua, the proposal of an expressive and yet simple to use language for O3F can be a useful contribution to advance the state of the art.

Some of the properties of the framework exceed the capabilities of common ontology representation approaches. In particular, the O3F framework allows to represent all commonly used ontological entities such as classes, properties and facets; it also represents some concepts that are not so common such as arbitrary axioms; and it represents other concepts that are not represented in other approaches such as action method definition.

Moreover, since the compact language proposed in this paper is much more usable and readable than the original proposal, it is at least likely that our work will contribute to facilitate the ontology specification process.

Finally, since we have provided an abstract grammar for the proposed language, it makes it possible to easily define diverse concrete syntaxes, which may be more adequate for the application at hand. For instance, the XML syntax of O3F will certainly be welcome in the web community.

The first future step is the development of a user-friendly editor for O3F to facilitate the definition and maintenance of ontologies using the O3F framework.

One would also profit from the existence of plug-in parsers and generators for other ontology representation languages such as DAML+OIL, OWL and Ontolingua. This way, it would be possible to import and export O3F ontologies from and to other formats.

Most ontology representation frameworks, such as description logic based approaches, have been proposed with the goal of enabling the development of decidable theorem provers. In general, it is necessary to trade off expressiveness for decidability. One of the future steps of our work will be to define subsets of the O3F representation language with different types of decidability.

6. ACKNOWLEDGMENTS

The research is supported by UNIDE/ISCTE.

7. REFERENCES

[1] Botelho, L.B.; Antunes, N.; Mohmed, E.; and Ramos, P. "Greeks and Trojans Together". In Proc. of the AAMAS 2002 Workshop "Ontologies in Agent Systems". 2002

[2] Botelho, L.M.; Mendes, H.; Figueiredo, P.; and Marinheiro, R. "Send Fredo off to do this, send Fredo off to do that". Submitted to the International Workshop on Cooperative Information Agents (CIA-2003). 2003

[3] Chaudhri, V.K.; Farquhar, A.; Fikes, R.; Karp, P.D.; and Rice, J.P. "Open Knowledge Base Connectivity 2.0.31". http://www.ai.sri.com/~okbc/spec.html. 1998

[4] DARPA Agent Markup Language. "Reference description of the DAML+OIL (March 2001) ontology markup language". 2001

[5] Farquhar, A.; Fikes, R.; and Rice, J. "Tools for Assembling Modular Ontologies in Ontolingua". In Proc. of the Fourteenth National Conference on Artificial Intelligence (AAAI'97). 1997

[6] Genesereth, M.R.; and Fikes, R., eds. Knowledge Interchange Format. *Draft proposed American National Standard* (dpANS). NCITS.T2/98-004 . 1998

[7] http://we-b-mind.org/o3f/language/abstract_spec_v1.htm

[8] http://we-b-mind.org/o3f/ontologies/travel_agency.htm

[9] Mota, L. "Extension to DAML+OIL: representation of methods". 3rd Agentcities.net Information Day. 2003

[10] Mota, L.; Botelho, L.M.; Mendes, H.; and Lopes, A. O_3F: an Object Oriented Ontology Framework. Proc. of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS2003), 2003, to appear

[11] W3C. OWL Web Ontology Language Overview. W3C Working Draft 31 March 2003. http://www.w3.org/TR/owl-features/. 2003

[12] W3C. Resource Description Framework (RDF). http://www.w3.org/RDF/