HOW COMPUTERISED AGENTS SEE THE WORLD

Robert M. Colomb
Department of Information Systems
Faculty of Computer Science and Information Systems
University of Technology Malaysia[1]

Work performed partly while visiting LADSEB-CNR[2]

Abstract: Agents, which are more or less autonomous programs performing tasks on behalf of users, act by exchange of messages. The content of messages is regulated by agreements called ontologies among the interoperating parties. In order for interoperation involving complex objects to be successful, there are several meta-ontological requirements, notably the ability to identify the object in the appropriate context and the ability to tell which are its parts. These issues of identity and unity are central to the OntoClean meta-ontology and method. This paper shows how they apply to a typical e-commerce application under multiple levels of refinement of more abstract objects into their parts. The point is made that a community of agents must operate in a world which is at least moderated, not a fully open world.

Keywords: Agent, Ontology, Interoperability, Object, ERM model

1. INTRODUCTION

With the advent of the world-wide web, there has been a great deal of activity in the development of agents, which are more or less autonomous programs performing tasks on behalf of users. These tasks are generally intended to involve interoperation among many sites, and generally involve information systems applications, including electronic commerce.

Agents act by exchange of messages. In order for agents to interoperate with web sites or each other, they must operate in a context of agreement as to what the messages and the terms used in them mean. These agreements are often called ontologies [5]. There is a wide variety of ontology. The present work is intended to apply specifically to those characterized as Run-time Interoperation Business Applications in the taxonomy of [8]. A large number of application-specific ontologies of this kind have been developed.

Agent technology is generally implemented in application-specific environments, in which

[1] colomb@utm.my
[2] Corso Stati Uniti, 4; Padova, 13512 Italia

much of the structure needed for the agents to perform their tasks is hard-coded. Intelligent agents which can operate autonomously in an open environment require that all relevant aspects of their environment be declared in the ontology describing the world in which it operates. This paper is an attempt to articulate the necessary structures. To assist with the presentation of a complex topic, the paper is cast in semi-anthropomorphic terms as "how a computerized agent sees the world".

We begin with requirements for seeing individual objects, then proceed to broader and broader constructs, proceeding through classes to the broadest constructs needed for the agent to perform.

2. INDIVIDUAL OBJECTS

If an agent it to do anything, it must be able to see individual objects. We proceed with the aid of a structure-oriented, content-neutral meta-ontology called OntoClean [8] which provides guidelines for constructing good ontologies (good in the sense that they support useful automated reasoning tools), and also provide formalisations of a number of very abstract, very commonly occurring structural relationships. These include part-whole, subsumption, identity and unity [6], among others. These meta-ontologies can be applied to other ontologies, including the Bunge-Wand-Weber (BWW) system [16] as well as to systems of modeling complex objects such as UML.

It is useful to describe the objects seen by agents using Searle's [15] framework of institutional facts. Searle recognizes two kinds of fact, *brute fact*, which is independent of human society, and *institutional fact*, which depends on human society for its existence. An institutional fact is a brute fact which has a social significance. Searle encapsulates the relationship as "brute fact X counts as institutional fact Y in context C". The purchase transaction (the message is a brute fact) between an instance of *customer* and the supplier in respect of a instance of *product* counts as the customer buying the product in the context including: the identifier of the customer is an instance of *customer* in the supplier's database, as is the identifier of the instance of *product*; and that the customer instance is not barred from participating because of poor credit; and that the instance of *product* refers to products which are in stock and to which the supplier holds title.

This characterization in terms of institutional facts is important because the *only* way agents can see anything is by the exchange of messages with other agents. Agents deal only in institutional facts. The other agents can be humans, or robots managing automated warehouses, sensor systems, or the like. This paper is not concerned with the internal workings of the robots, nor with the internal workings of the humans, but with the agents and how they interact. It should be clear then that if agents only see other agents, then the only

kind of fact agents can see are institutional facts. Brute facts are always mediated and interpreted by institutional context, even if the mediation is performed by a robotic system. The agent's actions have effect in the real world because of their institutional context. Even the automated warehouse only has effect in the real world because of how it is built and operated by a human institutional environment. Otherwise it is just a flailing mechanical device.

This claim that agents can see only institutional facts, and that institutional facts are what is described in ontologies, has significant consequences for the ontologies used to mediate agents' interactions. The most significant consequence is that the ontologies themselves are only valid within institutional context. (An EDI message which can be used in one industry exchange cannot necessarily be used in another unless the two exchanges are part of a larger institutional context.) There are other consequences which will be drawn our throughout the paper. For the time being it suffices to warrant the focus on the conceptual model of the agent's data structures, since the conceptual model is the specification of the institutional facts with which the agent works.

Returning to the immediate question as to how an agent sees an object, it should be clear that an agent sees anything as a data structure, a complex of interconnected atomic data items, which it can navigate. So at the most basic level, the ontology must include a specification of the data structures the agent will encounter. These might be RDF [14] triples collected into graphs, or CORBA data structures, for example. Note that the specification of the data structures does not determine how they will be transmitted. They could be encoded in XML documents, for example, but XML as such is not relevant to the structures themselves.

If the agent sees one and only one object, then the story ends here. There would be applications where this is the case, but there are many more applications where the agent sees possibly many objects, although perhaps one at a time. If there are more than one object, they must be able to be distinguished, so they need some form of identification. There would generally be a designated subset of the atomic data items whose values identify the object. The OntoClean meta-ontology calls this an *identifying relation.*

If the agent can see more than one object at a time, and the objects are complex, then it must be possible for an agent to tell which parts belong to which whole. So again, there must be a subset of atomic data items which the agent can use to assemble parts into wholes. OntoClean calls this a *unifying relation.*

Suppose for example, our agent can see two objects, represented as RDF triples

1	2	3
4	5	6

(Assume the integers are surrogates for URIs)

There are two kinds of atomic data item here. First, there are six values: 1 ... 6. Then there are the slots in the data structures in which these values are located: first triple subject, predicate, object; second triple subject, predicate, object. In this case the unifying relation is the data structure up to the slots. The identifying relation is the triplets of values ordered by the slots containing them.

We distinguish two general kinds of relations: lexical and logical. A *lexical* relation is essentially a navigable data structure. Our example unifying relation is lexical. A *logical* relation is a pattern of values in slots, like our identifying relation. A logical relation is so called because it can be expressed as a logical predicate.

The agent must know what structures to look at for logical relations and what structures to navigate for lexical relations. For the former, the slots participating in the relation must be labeled in some standard way. The kind of label used to mark the logical relation is part of the ontology.

3. GROUPINGS OF SIMILAR OBJECTS: CLASSES AND ASSOCIATIONS

While there are applications where an agent is programmed to deal with a specific collection of objects, it is more common for an agent's program to have common facilities for dealing with objects of the same kind. There are a number of ways of grouping objects into kinds, but the most widespread in computing is to use the class/instance paradigm. But see eg [11] for an alternative, which merits exploration.

Figure 1 shows a typical order entry ontology as a structure of classes and associations. The arrows represent associations whose target is the class at the end with the arrowhead. There are two versions of the ontology. The upper version is a more refined version of the latter. The objects in the ovals in the upper version are an articulation into parts of the corresponding object in the lower version.

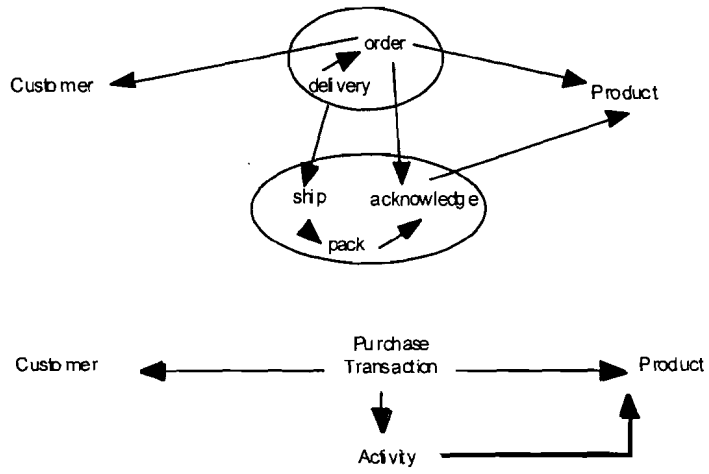

Figure 1: A fragment of a refinement of an order-processing ontology

The lower part of the figure is an abstract model of the process, showing a distinction between the interaction with the customer (*Transaction*) and the processes that the organisation must undertake to fulfill the order (*Activity*). *Purchase Transaction* depends on *Customer* and *Product*, while *Activity* depends on *Purchase Transaction* and *Product*.

The upper part of the Figure shows a refinement of the lower model, showing both *Purchase Transaction* and *Activity* as processes with parts (*systems* in the BWW model). *Purchase Transaction* is an *Order* followed by a *Delivery*, while the *Activity* process is first to acknowledge the order (*Acknowledge*), then pack it (*Pack*), then ship (*Ship*). The associations between *Order* and *Acknowledge*, and between *Delivery* and *Ship*, are both refinements of the association between *Purchase Transaction* and *Activity*. Figure 1 satisfies guidelines for refinements given in [2]. This refinement illustrates that institutional facts can be viewed at coarser or finer granularities, depending on the purposes of the viewer.

Using the class data model, we can now speak of properties applying to classes. Each instance of a class is associated with instances of other classes by means of properties. The identifying and unifying relations can now be expressed in terms of properties, with each instance of a class held together by a unifying relation involving the same properties, and each instance identified by an identifying relation involving the same properties. Now, the agent can navigate the classes and properties. To know which properties are identifying or unifying it needs some kind of label. For example, if the data structures are relations, the properties are attributes, and some properties are designated as parts of the primary key. The agent knows how to look for "primary key" labels on properties in the data schemas. We can think of these labels as properties of properties, or *metaproperties*. The agent knows how to look for properties with the metaproperties unity and identity.

Thinking at the class level allows a richer taxonomy of structures.

It is not necessary for a class to have unity. A *bulk* class lacks unity. Value measured in some currency units is a good example of a bulk institutional fact. If we have a class "Total of Account", it consists of an amount of value, which can be subdivided, so can be said to have parts. But if I add 10 units to it and you later remove 10 units from it, there is no sense in which the 10 units you remove is the "same" 10 units I added. There is no unifying relation maintaining a boundary around the 10 units.

Neither is it necessary for a class to have identity. A class can consist of a number of indistinguishable objects (say an inventory of boxes of pens). We can remove some pens and return exactly those pens, but can't record anything other than the total number. This sort of class is also a bulk class.

However, a structure composed of parts can't have a single identity unless it has unity. We must be able to tell which parts belong to a particular whole. A class with both unity and identity is called *countable*, and consists of individuals.

So our agent needs the metaproperties unity and identity for each class not only to manipulate individual instances, but also to determine whether the class has individual instances at all. It will deal with countable classes differently from bulk classes.

4. APPLICATION TO THE EXAMPLE – INDIVIDUAL OBJECTS

In ontologies where there are classes and associations, identity in particular can be complex.

We first look at a representation of a single class which is the source of no many-to-one associations, in the example *Customer* and *Product*. (We will call these universal targets *independent classes*.) Assume we have an identifying relation for the Product class consisting of an internal product ID.

An identifying relation in terms of internal product ID is not sufficient. It works well for the internal working of the organisation, but is not generally a satisfactory way of representing identity outside the particular organisational context. A customer is looking for citric acid 99.9% purity in 200 litre drums, not product CA999-200. A customer knows their own name, address, and other details, not generally their customer number. Even though the product identifier is in practice used as the primary key, there must always be a candidate key composed of attributes whose values are known by all parties in potential interactions with the information system. (The situation is in fact more complicated than this, but we will defer consideration of the additional complexity until we treat interoperating agents, below.)

There could be a number of such candidate keys. For example, [12] describes a system where electronic parts have besides their distributor-specific product identifier a global identifier called *Federal Supply Clauses* which can be used by agencies of the United States Government, but not by other purchasers. So the identifying relation depends on context. This

is another consequence of the nature of the product as an institutional fact.

A significant point emerges here. What the information systems designer thinks of as an instance can be from the perspective of the ontology a type, not an instance. That is to say, our agent can encounter a class whose instances are themselves classes. The information system designer's concept of an instance of *Customer* is also an instance in the ontology, but the same is not true of *Product.* In particular, product number CA999-200 or "citric acid 99.9% purity in 200 litre drums" both identify product types. There can be an amount of the product in various places in the warehouse, in various stages of manufacture, and in various shipments to customers, not to mention waste or spoilage. Instances of the countable class Product are themselves classes, either bulk or countable. (A product whose instances carry identity plates but are otherwise indistinguishable, like consumer electronics products, is a countable class.) It would be useful to have a metaproperty on a class indicating whether its instances were themselves classes.

Since agents see only institutional facts, agents see a simpler world than humans. The analysis of identity and unity in [6], [7] is cast mostly in terms of human perception. In general, a human perceives something in a superfluity of qualities. For example, a jar of citric acid is seen as a transparent container filled with a white powder, having a certain weight, texture, taste and so on. Besides these immediate qualities, the object behaves in certain ways under various tests. The container breaks if struck by a hammer. The powder dissolves in water. The solution fizzes if sodium bicarbonate is added. This analysis of identity and unity includes the ancient problem of how to select from these qualities those qualities which serve to identify the type to which the object belongs and to hold together the parts of complex objects.

The fact that agents see only institutional facts greatly simplifies the problem, since an institutional fact exists only in representations, and the identity and unity criteria are explicitly specified aspects of the representations. The complexity of identity and unity we show in the present work is extreme simplicity in the general context.

5. DEPENDENT CLASSES

In information systems applications classes like *Purchase Transaction* which are the source of mandatory many-to-one associations (we will call *dependent* classes) pose a somewhat more complex problem. The representation of these classes (still referring to business objects) will generally include identification of the business objects represented by classes which are targets of the mandatory many-to-one associations. Semantically, the independent classes define the most stable aspects of the operation since they must have instances in order for the other classes to have

instances. The dependent classes represent records of the more dynamic aspects of the operation. The institutional facts referred to by the populations of the dependent classes have as some of their properties the identifiers of the records of other institutional facts. These other institutional facts exist for the speech act creating the present institutional fact to be validly framed, so these records are important properties. An invoice may require a purchase order, a delivery advice and a delivery acknowledgement, for example.

So, although it is common for invoices to be identified internally by say an invoice number, a convenient way to obtain an identifying relation for wider contexts is to build on the identifying relations for the objects represented by the independent classes which are appropriate to the particular context. (This is essentially the formal mechanism of *weak entities* in ER modeling.)

In our example, *Purchase Transaction* is dependent on both *Product* and *Customer*. The third dependency, on *Activity*, is subject to an additional integrity constraint whereby *Purchase Transaction* is transitively also dependent on *Product*, but the product instance reached in either path must be the same, so this dependency does not add anything. (This kind of integrity constraint is called *the principle of consistent dependency* in [1], and is an instance of the category theoretic concept of a *commuting diagram*.)

If each of the associations between *Purchase Transaction* and *Product* and respectively *Customer* were one-to-one, then the concatenation of the identifiers of *Product* and *Customer* appropriate to the context would be sufficient to identify an instance of *Transaction*. In general, however, the associations are many-to-one, so that for each pair of *Customer* and *Purchase Transaction* instances, there can be many instances of *Transaction*. An additional local candidate key is also needed. Internally one might use a sequence number and externally for example *Date*.

In this example, the instances of *Purchase Transaction* are all individuals. Given the need for accounting and audit, it is hard to see how an instance of *Purchase Transaction* could be a bulk property.

So if the data representation used by our agents permits more than one way to represent a property, the different kinds of representations all need to support the metaproperties.

6. APPLICATION TO THE EXAMPLE – CLASSES

Instances have now been identified as instances of their respective classes. This leads us to ask how the classes themselves are identified. If our agent encounters an object it needs to be able to figure out what class(es) it belongs to.

Institutional facts are completely characterised by representations of their records, so have a

limited set of properties. In particular, there is no way to distinguish an invoice from a credit note without the record of the class of institutional fact. So it is hard to see that the class of invoices is anything other than the subclass of institutional fact whose representations contain an attribute called "type" which has the value "invoice".

In practice, systems designers identify classes in several different ways. If the primary design vehicle is the ER model, the class is represented simply as a name on a graphical element (entity), which has graphical connections to representations of names of attribute value sets and graphical representations of relationships with other entities. Often, the representation is a translation of the conceptual representation to a relational database table scheme, where the entity name becomes the table name, and the relationships and attributes column names, the former labeled foreign keys.

Other ways also are used. Sometimes several classes are combined, possibly redundantly, in a single table (universal relation) – often as a view. In this case, to identify an instance of the representation of a class, the programmer needs to know which columns contain the preferred candidate key and which columns contain the attributes and foreign keys associated with that class. It is also possible to represent a conceptual model in a single table with four columns: *class*, *tuple*, *attribute*, *and value*. Each row of this table contains a single value of a single attribute of a single tuple instance of a single class. (The tuple in this case is often identified by an arbitrary number which is not itself the value of an attribute.) Here, the programmer needs to know the name of the class in order to retrieve its instances.

So in practice the unifying relation for a class is something like an SQL statement (the statement, not the result), and the identifying relation is either the name of the class or the names of columns which are known by the programmer of the agent to constitute the representation of the class. Classes are therefore often identified lexically. If, however, the objects are represented using RDFS, an object will have a property "rdf:type" whose values are the classes of which it is an instance, so will have a logical identifying relation.

OntoClean has a metaproperty for this. A property identifying the class of which an individual is an instance is called *rigid*. Our agent needs to be able to look for properties with the metaproperty *rigid*.

7. APPLICATION TO THE EXAMPLE - UNITY

Figure 1 shows in its upper half a refinement of the model in the lower half. The class *Purchase Transaction* has been refined into two parts, *order* and *delivery*; while the class *Activity* has been refined into three, *acknowledge*, *pack* and *ship*. How do we identify instances of these new classes?

We have looked at identification of instances of *Purchase Transaction* and *Activity*. One general way to identify parts of a whole is to use the identifier of the whole with the addition

of a local part identifier, as in weak entities in the ER system. That method won't work in this case, since the whole is not represented. If we think of the lower part as being a representation of a strategic view of the ontology and the upper part as being a representation of the ontology's tactical implementation, then in developing the ontology, the lower half is replaced by the upper half when the system is implemented. When we are thinking about *order* and *delivery* there is no longer any class *Purchase Transaction.*

In operation, the system will generate linked instances of each of the part classes. If we happen to want to look at combinations of instances of *order* and *delivery* as instances of the more general *Purchase Transaction*, we will create them by a query. So the whole only comes into existence when all of its parts do. Further, we might be interested in partially completed wholes. In these cases, only some of the parts exist, so the whole does not exist at all. So we can't identify the parts with reference to the whole.

What we can do is take advantage of the fact that the instances of the part classes are linked by foreign keys. Every instance of *delivery* is linked to an instance of *order*, every instance of *pack* is linked to an instance of *acknowledge*, and every instance of *ship* is linked to an instance of *pack*, which is in turn linked to an instance of *acknowledge.* If we identify the instances of the parts in some way, we can derive an identification of the whole. Every instance of *Purchase Transaction* includes an instance of *order*, as does every instance of partially completed *Purchase Transaction.* Similarly, every instance of *Activity* includes an instance of *acknowledgement*, as does every instance of partially completed *Activity.* So one way to identify the whole is by one of its parts which always exists: *Purchase Transaction* by *order* and *Activity* by *acknowledgement.* Identifying a whole by one of its parts is called *metonymy.*

If we identify the whole in this way, then the unifying relation for the whole becomes a query on the part classes where the identifiers are the same.

The agent needs to be able to see some kind of representation of the whole in order to look for its identifying and unifying relations via the relevant metaproperties. One way to do this is to include an object in the ontology which names the whole and serves as a point of attachment for the queries. This object does not need to have instances of its own, and is not a superclass of the concrete part classes. We can think of it as a class-like object called *phantom whole*, and it needs a metaproperty to label it as such.

8. SUPPORTING INTEROPERATION

An order processing agent would naturally interoperate with a purchasing agent operated by another organisation in an electronic commerce environment under the framework of some sort of institutional structure which gives context to the institutional facts involved. The application of the concepts of identity and unity in the previous section were in the context of

a single agent, where the complex information structures tend to blend in one with another. This is because the main issue in conceptual modeling for a single agent is the association among atomic elements throughout the agent. Interoperation involves the sending of messages consisting of complex objects or the sharing of complex objects. The integrity of the objects needs to be emphasized, although still in context with the associations of their elements with other elements in either agent.

First, neither agent would generally expose itself in its entirety to the other. Each agent in the interoperation would see something like in Figure 2. As with Figure 1, the upper system is a refinement of the lower. The lower system shows a purchasing agent (left) and a supplier's order processing agent (right). Both agents interoperate via *Transaction*. Formally, what has happened is that each agent presents a view of itself to its partners. A view is a graph homomorphism from the conceptual model of the view into the conceptual model of the entire system [10]. If the underlying system has an identifying relation for the system as a whole, then the view can have one, too, so that the view is unified and identified in the same way as the underlying system. (If necessary, the two can be differentiated by differentiating the names used.)

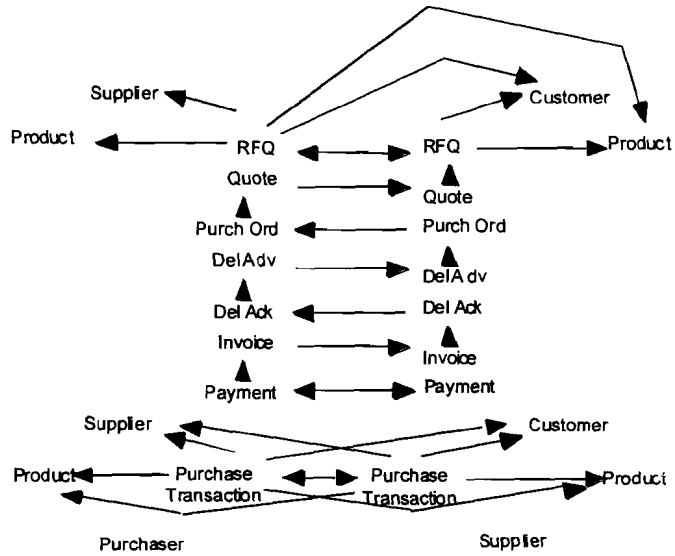

Figure 2: Purchasing and Supplier agent interoperating

Still concentrating on the lower (abstract) system, note that on each side, *Purchase Transaction* is dependent on both *Supplier* and *Product* on the Purchaser side, and also on *Customer* and *Product* on the Supplier side. We focus first on the associations involving *Product* on each side.

Product in each agent is an independent class. In our (simplified) system, a transaction involves an instance of *Product* on the Supplier side and an instance of *Product* on the Purchaser side. In fact, it is physically the same product on each side, moving say from one

warehouse to another. There is therefore an issue of identity – the identifying relations for *Product* on each side need to be correlated, generally by a one-to-one mapping. The mapping need be neither injective nor surjective – one company may purchase only some its products from a given supplier, and purchase only some of the products offered by any given supplier. This mapping implicitly provides an extensional unifying relation for the subtype of products shared between a given customer and supplier.

In practice the problem of identification is much more complicated. We have assumed that the purchaser and supplier agree on the specification of the product in question. However, the supplier may have a richer subdivision of the product type than the purchaser. They may supply computers with different colour casings, to which the purchaser is indifferent, but each different colour is a different product type. Similarly, the purchaser's taxonomy can be richer. The purchaser may specify computers of a certain specification with a green casing, but the supplier does not specify colour. Even worse, the two may have different taxonomies. The purchaser may distinguish colour while the supplier distinguishes material of the casing.

Further, the identification criteria must scale. A pairwise correspondence will serve two parties, but if there are hundreds of players in an e-commerce exchange, the resulting equivalence classes are extremely cumbersome. A much more practical approach is to establish an exchange-wide nomenclature something like ISBN. This requires that the exchange have players who can make the speech acts to declare that a given object is given the appropriate identity. There are many design issues here: can an object be given more than one identity (consider mobile telephone numbers)? Does this matter (consider US Social Security accounts)?

Resolution of these issues is essential to successful interoperation, but is outside the scope of the present paper. In practice, for example, the agents can report back to their human organisations and allow the humans to negotiate. The present concern is the prior question as to how the unity and identity are represented across organisations.

Additional issues are raised in the associations involving *Supplier* and *Customer*. It is normal for the Purchaser agent to have a class *Supplier* (as in Figure 1), and for the Supplier agent to have an analogous class *Customer*. However, what is an instance of *Supplier* for the Purchaser is the entire agent of the Supplier, and what is an instance of *Customer* for the Supplier is the entire agent of the Purchaser. In isolated systems neither the unifying nor identifying relations for the whole system are generally represented. For interoperation, however, they must be. So the name of each agent must be mapped into an instance of a class of the other. The scaling problems mentioned above apply here, too.

Purchase Transaction in the lower part of Figure 2 is different from the class *Purchase Transaction* in Figure 1. In the present case it is the exchange of messages which effect the interoperation between the two agents. In the former case, it is the representation of the

speech acts in the supplier side which implement the interoperation. The two are closely related, hence the use of the same name. In particular, the identification of the unrefined *Purchase Transaction* of Figure 2 is similar to the identification of the *Purchase Transaction* of Figure 1 discussed in the previous section.

We consider now the refinement of the *Purchase Transaction* classes in the upper part of Figure 2. A business transaction is an institutional fact normally created in a series of speech acts organised into a process, using a semantic protocol like one of the EDI (Electronic Data Interchange) standards. This means that to carry out the interoperation, the abstract system needs to be refined, so that *Purchase Transaction* has several parts. In the example of Figure 2, we have the interaction as proceeding from a request for quotation (*RFQ*) issued by the purchaser, through *Quote* by the supplier, *Purchase Order* by the purchaser, *Delivery Advice* by the supplier, *Delivery Acknowledgement* by the purchaser, *Invoice* by the supplier to *Payment* by the purchaser. In practice, of course, the interaction can be much more complex, involving many more exchanges of different types, and the sequence need not be linear.

In the example, each side keeps copies of all messages, very likely linked to other aspects of their respective systems outside the view. Each message participates in a many-to-one association with an earlier message. The first message (*RFQ*) is shown with a one-to-one association between the parties – the association from Purchaser to Supplier represents acknowledgement by the supplier that a communication sequence has been established. The last message (*Payment*) has a similar one-to-one association representing acknowledgement of the closure of the interaction.

The refinement is conceived of as an articulation of the whole of *Purchase Transaction* into parts. We therefore need to consider the unifying and identifying relations required.

An instantiation of *Purchase Transaction* is as a process, so its parts come into existence one by one over possibly considerable time. (We do not need to take clock time into account, simply sequence.) Further, in the example the whole is not represented in the refinement, only the parts. A whole transaction instance is represented by the assembly of all of its parts. So besides unity and identity, we need to consider existence. We focus first on identity and unity.

Atomic parts of *Purchase Transaction* are instances of *RFQ*, *Quote*, and so on. The first part to come into existence is *RFQ*, and it can be identified in the same way as we have discussed for the unrefined *Transaction* of Figure 1, as an instance logically by the associations with either *Customer* and *Product* or *Supplier* and *Product* for Supplier and Purchaser respectively, together with an agreed attribute like *Date*. Since there needs to be agreement between the parties on the identification of RFQ, there needs to be an intersystem identity relation, which can be supplied by including all three of *Product*, *Supplier* and *Customer* associations, taking advantage of the identification of the Purchasing agent as an instance of *Customer* and the Supplier agent as an instance of *Supplier*. Of course the identity

relation for the *RFQ* instance also includes the lexical identification of it as an instance of the *RFQ* class.

The other parts as they come into existence can be identified by their possibly indirect association with *RFQ*, if necessary including a further local identifier based on *Date* or *Message-ID* or some other attribute whose scope includes the contexts of both parties.

Dependence on *RFQ* provides a convenient unifying relation for the whole transaction. However, the unifying relation needs to exclude the objects on which parts depend, such as *Customer*, *Supplier*, and *Product*, while keeping the association instances which form a record of the satisfaction of the contextual conditions for the speech acts. In other words, in keeping track of the parts of an instance of *Purchase Transaction*, we need to record the instances of *Customer*, *Supplier* and *Product* on which the instance depends, but the *Customer*, *Supplier* and *Product* instances themselves are not parts of the instance of *Purchase Transaction*. The unifying relation is what maintains the boundary of the *Purchase Transaction* object.

We now consider how to identify the whole of *Purchase Transaction*, assuming we have a complete collection of parts. One obvious way is to employ metonymy (naming a whole by one of its parts), using the identifier of RFQ as the identifying relation of the whole. However, the various parts of the transaction come into existence over time, and in practice may never come into existence. At any given time, the populations of the classes refining *Purchase Transaction* will contain all sorts of incomplete transactions. Some of these incomplete transactions may be stopped, for example it often occurs that a purchase order is not ever issued in respect of a quote, and it sometimes happens that a customer does not pay. It therefore may suit the organisations to refrain from identifying a whole transaction until a part comes into existence which usually leads to completion, say *Purchase Order*. But of course the identification of a not-completed transaction does not guarantee that it will ever be completed.

The fact that a transaction's parts come into existence over time, and in fact may never come into existence, suggests a state view of the process, which requires further research.

9. FURTHER ASPECTS OF INTEROPERATION

An autonomous agent lives in a world consisting of other agents. It needs to know who these other agents are. The agents do things, using exchanges of messages. They need to know what actions are possible, and what their preconditions are. Further, they will perform complex actions by composing more elementary actions. They need to know what an action does in a way that relates to preconditions for other actions. Working out ways to express these things is the subject of the WSDL [17] and OWL-S [13], among others. Space precludes

a detailed discussion.

However, we must note that the players, messages, preconditions, postconditions and other structural aspects of the ontology of the agent world need to be defined across the world. So we need not only the structural primitives of WSDL and OWL-S, but we need to standardize the concepts used in the WSDL and OWL-S descriptions of the agents. Otherwise the agents will not understand each other.

10. CONCLUSION

We have analysed how an agent sees. The first point made was that an agent sees only institutional facts, and what an agent does is perform speech acts which create institutional facts. The agent needs to see its counterparts, but also itself.

The agent's world consists of other agents, possible actions, preconditions and postconditions, and classes of objects of various kinds, held together by unifying relations and distinguished by identifying relations which must be agreed at the level of the world as a whole.

So an autonomous agent must live in a semi-closed world. The organizations responsible for the agents must participate in the agreements establishing the ontology representing the world, and each agent must commit to that ontology. If the agents are allowed to do anything interesting, like buying and selling, then the commitment of an agent to the ontology must be certified in some way, so that the system is not fully open, but is at least moderated.

REFERENCES

[1] R.M. Colomb, C.N.G. Dampney, and M. Johnson, Category-Theoretic Fibration as an Abstraction Mechanism in Information Systems, *Acta Informatica* 38, (2001) 11-44.

[2] R.M. Colomb and C.N.G. Dampney (2005) "An Approach to Ontology for Institutional Facts in the Semantic Web" *Information and Software Technology*, Volume 47, Issue 12, 1 September 2005, Pages 775-783

[3] C.N.G. Dampney, Specifying a semantically adequate structure for information systems and databases, Proceedings of the 6th. International Conference on the Entity-Relationship Approach, New York, 9-11 Nov., 1987; North Holland Publishing Company (1987) 143-164.

[4] C.N.G. Dampney, Personal communication (2001)

[5] T. R. Gruber, Toward Principles for the Design of Ontologies Used for Knowledge Sharing, Technical Report KSL 93-04, Knowledge Systems Laboratory, Stanford University 1993.

[6] N. Guarino and C. Welty, Identity, Unity and Individuality: Towards a Formal Toolkit for Ontological Analysis, in: W. Horn (ed) Proceedings of ECAI-2000: The European Conference on Artificial Intelligence IOS Press, Amsterdam, 2000.

[7] Guarino, N and Welty, C., Ontological analysis of taxonomic relationships, in A. Laender and V. Storey (eds) Proceedings of ER-2000: The International Conference on Conceptual Modelling, October, 2000 Lecture Notes in Computer Science Vol. 1920, Springer, Berlin, 2000.

[8] N. Guarino and C. Welty Evaluating Ontological Decisions with OntoClean, Communications of the ACM, 45(2) (2002) 61-65.

[9] L. Hart, P. Emery, R. Colomb, K.y Raymond, D. Chang, Y. Ye, E. Kendall & M. Dutra (2004) "Usage Scenarios and Goals For Ontology Definition Metamodel" in Zhou, X., Su, S., Papazoglou, M., Orlowska, M. and Jeffery, K. (eds.) *Web Information Systems Engineering Conference* (WISE'04) 22-24 November, 2004, Brisbane, Australia. Springer LNCS 3306 596-607.

[10] M. Johnson and C.N.G. Dampney, On Category Theory as (meta) Ontology for Information Systems Research, International Conference On Formal Ontology In Information Systems (FOIS-2001) October 17-19, 2001, Ogunquit, Maine ACM Press, New York, 2001.

[11] Lakoff, George (1987) Women, Fire and Dangerous Things: what categories reveal about the mind University of Chicago Press.

[12] A-L Neches, Government Application of FAST Technology, Tasks 1, 3 & 4 Semiannual Technical Report, Information Sciences Institute, University of Southern California, USA, April 1993.

[13] OWL-S 1.1 Release http://www.daml.org/services/owl-s/1.1/

[14] *Resource Description Framework (RDF)* www.w3.org/RDF/

[15] J. R. Searle, The Construction of Social Reality, The Free Press, New York, 1995.

[16] R. Weber, Ontological Foundations of Information Systems Coopers & Lybrand Accounting Research Methodology. Monograph No. 4. Melbourne, 1997.

[17] *Web Services Descriptin Language* (WSDL) 1.1 http://www.w3.org/TR/wsdl