How Computerised Agents See the World

Robert M. Colomb

Technical Report 17/02 ISIB-CNR
Padova, Italy, November, 2002

National Research Council

Institute of Biomedical Engineering

ISIB-CNR

Corso Stati Uniti, 4

35127 Padova

Robert M. Colomb

School of Information Technology and Electrical Engineering
The University of Queensland
Queensland 4072 Australia
colomb@itee.uq.edu.au

Work performed partly while visiting LADSEB-CNR; Corso Stati Uniti, 4; Padova, Italia,

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. In particular, it shows that identity and unity are sometimes represented lexically rather than logically, and how a state view of the interoperation follows naturally from its structural specification.

How Computerised Agents See the World

Robert M. Colomb
School of Information Technology and Electrical Engineering
The University of Queensland[1]

Work performed while visiting LADSEB-CNR[2]

E-mail contact **colomb@itee.uq.edu.au**

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. In particular, it shows that identity and unity are sometimes represented lexically rather than logically, and how a state view of the interoperation follows naturally from its structural specification.

Introduction

Ontology and information systems interoperability

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 (Gruber, 1993). A large number of application-specific ontologies have been developed.

An ontology is a data structure designed to support interoperation. It does not generally support an information system directly, but provides either high-level or coordination-specific structures which can help the developers of the interoperating systems bring their structures sufficiently together to interoperate. In particular, there are structure-oriented, content-neutral meta-ontologies such as OntoClean (Guarino & Welty, 2002) which provide 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 (Guarino & Welty, 2000), among others. These meta-ontologies can be applied to other ontologies, including the Bunge-Wand-Weber (BWW) system (Weber, 1997) as well as to systems of modeling complex objects such as UML.

Tools have also recently become available for the principled representation of complex information system models as structurally coordinated refinements of high-level models using category theory (Colomb, Dampney & Johnson, 2001). These high-level models can be at a similar level of abstraction to the ontologies, and therefore can provide a bridge to development of interoperation. The term *refinement* has a history of use in the field of software engineering, where it refers to the addition of

[1] Queensland 4072 Australia colomb@itee.uq.edu.au
[2] Corso Stati Uniti, 4; Padova, 13512 Italia

detail to a specification of either programs or data structures in such a way that the behaviour of the refined specification implies the behaviour of the more abstract specification.

The present paper is intended to show how the meta-properties of OntoClean assist the information system developer in representing the world that the agents being developed will see as they carry out their tasks. It begins with a characterization of the class of problem involved in information systems interoperation as the manipulation of records of institutional facts, which suggests that the usual way ontologies are presented is too general and can usefully be specialized to the domain of records of institutional facts. This view allows us to explicate key issues in information systems interoperation in terms of two central meta-properties, unity and identity, first with a focus on the meta-properties then on their application to interoperability. In particular, it shows how the meta-properties lead to a principled integration of a message view and state view of the interoperation, which gives the requirements analyst a very powerful tool to represent the specifications for interoperating agents.

Representation of identity and unity in a single information system

Single system example

Before we look at the issues of identity and unity for interoperating information systems, we will look at how these are represented in single information systems. Figure 1 shows a fragment of an information system supporting an order-entry/ order fulfillment style application. The model is ER, with the notation showing *many-to-one* relationships as having an arrowhead at the *one* end (*One-to-one* relationships have arrowheads at both ends.) 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 a customer and a product, while *Activity* depends on a *Purchase Transaction* and a *Product*.

It is useful to describe the contents of information systems in Searle's (Searle, 1995) 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.

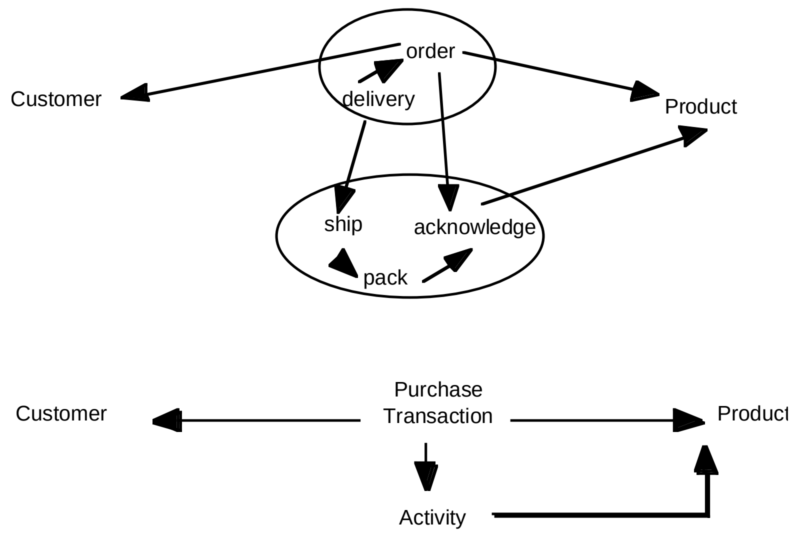

Figure 1: A fragment of a refinement of an order-processing system

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, then pack it, then ship. The relationships between Order and Acknowledge, and between Delivery and Ship, are both refinements of the relationship between Purchase Transaction and Activity. Figure 1 satisfies plausibility criteria for refinements given in (Colomb, Dampney & Johnson, 2001). This refinement illustrates that institutional facts can be viewed at coarser or finer granularities, depending on the purposes of the viewer.

Axioms for identity and unity

We begin our investigation by seeing how the formal criteria for unity and identity of (Guarino and Welty, 2000) (henceforth *GW*) apply.

As we will see below, in the GW system it is not necessary for an entity to have unity in order for it to have identity. A bulk product has identity - we can identify water, eg, without having any representation of the unity of water.

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. *GW* develop their axioms for unity from the mereology of (Simons, 1987). A whole is a collection of parts

$P(x, y)$ is true if x is a part of y (1)

(*GW* introduce an additional parameter, time, which is not at issue in the present work, so has been omitted.)

An object x is an integrated whole if it has a division into parts that is a closed system under a suitable equivalence relation ω called the *unifying relation*. The axiom given is

$$\forall y(P(y, x) \rightarrow \forall z(P(z, x) \leftrightarrow \omega(z, y))) \quad (2)$$

That is to say, every part of x in a certain division is related by the unifying relation to every other part in that division, and to nothing else. (GW give a second axiom, which is intended to exclude trivial unifying relations.)

GW also develop an identity axiom from (Lowe, 1989), which requires another equivalence relation called an identifying relation. A property carries an identity criterion iff

$$\phi(x) \text{ and } \phi(y) \rightarrow (\rho(x, y) \leftrightarrow x = y) \quad (3)$$

(GW also supply additional axioms intended to prevent trivial identifying relations.)

Our intent in this example is to see how these ideas apply to the problem of interoperating information systems. The variables in the preceding formulas therefore are intended to be instantiated by various business objects handled by the information systems. The business objects are all either institutional facts or brute facts associated with institutional facts, therefore the entities of concern are generally representations (brute facts which are records of institutional facts).

Application to the example – individual objects

We first look at a representation of a single entity which is the source of no many-to-one relationships, in the example *Customer* and *Product*. (We will call these universal targets *independent entities*.) The semantics of the ER model are based on set theory, so the principal interpretation of the representation of the entity is as a set of instances. An instance is represented as a tuple of atoms, called *attribute values* (the state of a thing in the BWW system).

The tuple of values can be constructed in a number of ways. One way is to store the tuple directly, in which case the unifying relationship is lexical rather than logical. That is to say, unifying relationship for the tuple of attribute values is "being present in the same tuple". Since we don't yet have an identity criterion, the unifying relationship is represented lexically rather than logically (as a list of attribute values, say, or as a set of attribute name/value pairs).

Another is to construct the tuple as a view (a query, which is equivalent to a logical formula). This is a logical construction, but still based on the most primitive tuples which associate the identifier of the object with the institutional fact represented by a particular attribute value.

So the unifying relationship for a representation of an institutional fact is at least partly lexical, so not entirely logical.

In database theory, an instance of an entity is identified by a subset of attributes, whose pattern of values is unique for each tuple (called a *candidate key*). It is very common for the system to be designed with an attribute intended as a candidate key, such as *product number* or *customer identifier*. So in the case of *Product*, an identifying relation can often be represented logically as

$$\rho_1(x, y) = \text{product number}(x) = \text{product number}(y) \qquad (4)$$

Remember that we are referring to the business object, which is characterised by its representation.

However, the identifying relation (4) 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.

There could be a number of such candidate keys. For example, (Neches, 1993) 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 relationship depends on context.

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. 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. Our identity criteria as so far presented do not go deep enough.

In some cases individual instances of products are routinely identified. A car or a computer or a piece of consumer electronics normally has a serial number plate on it, which identifies it as an instance of that type. This identifying plate is the only difference between otherwise indiscernible objects of the same product number.

The citric acid example isn't like that, though. One need for identifying an instance of this product type is to identify a lot of the product subject to a particular transaction or activity. In this business context, the particular lot of the product which the transaction is about is determined quite late in the process, at the time of shipment. Often it is not determined until a particular collection of drums is loaded onto a truck. The drums may not be distinguishable one from another, their only labeling may be with the product type. So the particular instance of the product in this case would seem to be "the drums loaded onto truck VVV1234 for delivery on 20 March, 2002 under delivery manifest number 77883322 to Acme Ltd.".

However, another need is the product in inventory at a particular time. Some products are produced in identified batches (eg pharmaceuticals) or have use-by dates, which can be identified as wholes of an amount of matter. However, many products are simply flows into and out of inventory. The product may be packaged in one form or another like our drums of citric acid, or may be stored in bulk (oil, say).

This gives us a taxonomy. In terms of *GW*, the entity *Customer* is a collection of individuals (having both unity and identity). The entity *Product* is a collection of types, which may or may not consist of individuals. The individuals may be distinguished only by name. But we have seen that the types in *Product* often carry identity (as an instance of the identified type), but not unity so that the lots of product can not be identified other than by the container they may be in. We can call these properties (which carry type identity but not unity) *bulk* properties as distinguished from the *countable* properties of *GW* (which carry both identity and unity so yield individuals).

Note that unity can appear at some levels of granularity and disappear at others. Following *GW*’s example of apple in the spirit of the purchasing application, the product type may be “Apple – Granny Smith in cases of 80”. The cases carry no identification. So at the level of maintenance of inventory, shipping and receiving, the product is bulk. However, a particular customer (a restaurant, say) may have ordered a small number of cases in a particular transaction. The cases may be able to be identified now (by location in the truck, say) during the delivery, but then go into the restaurant’s cool store and merge with other cases in bulk again. When a case is opened, the apples in it are not identified, but an apple served to a particular customer of the restaurant may be identified, again by what amounts to packaging. To this customer, the apple carries identity not only of type, but also individuality, since it is the (only) apple served them at that time. It also has a topological unity, so is an individual in this context.

Dependent entities

In information systems applications entities like *Purchase Transaction* which are the source of mandatory many-to-one relationships (we will call *dependent* entities) pose a somewhat more complex problem. The representation of these entities (still referring to business objects) will generally include identification of the business objects represented by entities which are targets of the mandatory many-to-one relationships. Semantically, the independent entities define the most stable aspects of the operation since they must have instances in order for the other entities to have instances. The dependent entities represent records of the more dynamic aspects of the operation. The institutional facts referred to by the populations of the dependent entities 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 entities 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 (Colomb, Dampney & Johnson, 2001), and is an instance of the category theoretic concept of a *commuting diagram*.)

If each of the relationships 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 relationships 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.

Application to the example – classes

Instances have now been identified as instances of their respective entities. This leads us to ask how the entities are identified. Entities represent classes of business objects, and a class of business objects is often the subject of discourse.

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”.

So a class of institutional fact is represented by an entity, which formally is a set whose members are instances, but the normal practice is to define the entity intentionally rather than extensionally. The same entity can have many different populations – indeed much of the code in an information system is devoted to updating the populations of entities. That is to say, much of the code is devoted to updating the representations of members of classes of business objects.

In practice, systems designers identify entities in several different ways. If the primary design vehicle is the ER model, the entity is represented simply as a name on a graphical element, 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 entities 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 an entity, the programmer needs to know which columns contain the preferred candidate key and which columns contain the attributes and foreign keys associated with that entity. It is also possible to represent a conceptual model in a single table with four columns: *entity*, *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 entity. (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 entity in order to retrieve its instances.

So in practice the unifying relation for an entity is something like an SQL statement (the statement, not the result), and the identifying relation is either the name of the entity or the names of columns which are known by the programmer to constitute the representation of the entity. Entities are therefore identified lexically.

Application to the example – the system as a whole

Finally, we take another step upwards in scale and look at the system as a whole. This system is a single system, which could be one of many operated by the same organisation, so would have a name, say *Acme Corp Ordering System* (henceforth ACOS).

Either system in Figure 1 has a unifying relation, namely "is a part of ACOS', but that unifying relation is not represented in the structure. The unity of the system is maintained by its operational environment: it is located on a particular cluster of servers operated by a particular company, compiled from modules held under a certain version in a certain repository, and so on. As an isolated system, it has no practical need for its unity to be represented internally.

However, it is not difficult to represent a unifying relation. Following Dampney (2001), we can supply an additional entity ID, called the identifying entity, with exactly one instance and require a (unique) many-to-one relationship id from each entity in the system to the identifying entity. We can parameterise the identifying entity by the name of the system, represented as ID_x. Using the category theoretic framework, the identifying entity is a terminal object. The unifying relation of the refined model is a refinement of the unifying relation of the abstract model. In this case, we have

$$\omega_x(z, y) = \exists z,y \text{ (id}(z, ID_x) \text{ and id}(y, ID_x)) \qquad (5)$$

where the unifying relation is also parameterised by the name of the system.

Given the unifying relation (5), we can represent an identity criterion for "is system ACOS" as

$$\exists x\ ID_x \text{ is a terminal object for system ACOS} \qquad (6)$$

Because they are both parameterised by the name of the system, neither the unifying nor identifying relations are logical.

This is all a bit academic, of course, since we have already established that in the isolated system we have taken ACOS to be, there is no practical need to represent either unity or identity. However, as we see below, unity and identity become very important when we start seeing ACOS as a member of a community of interoperating systems.

Interoperating systems

An order processing system would naturally interoperate in an electronic commerce environment with a purchasing system operated by another organisation. The application of the concepts of identity and unity in the previous section were in the context of a single information system, where the complex information structures tend to blend in one with another. This is because the main issue in conceptual modeling for a single information system is the relationship among atomic elements throughout the system. 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 relationships of their elements with other elements in either system.

First, neither system would generally expose itself in its entirety to the other. Each system 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 system (left) and a supplier's order processing system (right). Both systems interoperate via *Transaction*. Formally, what has happened is that each system presents a view of itself to its partners. Following the category-theoretic framework, a view is a functor (graph homomorphism) from the conceptual model of the view into the conceptual model of the entire system (Johnson & Dampney, 2001). If the underlying system has a terminal object, 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 terminal objects.)

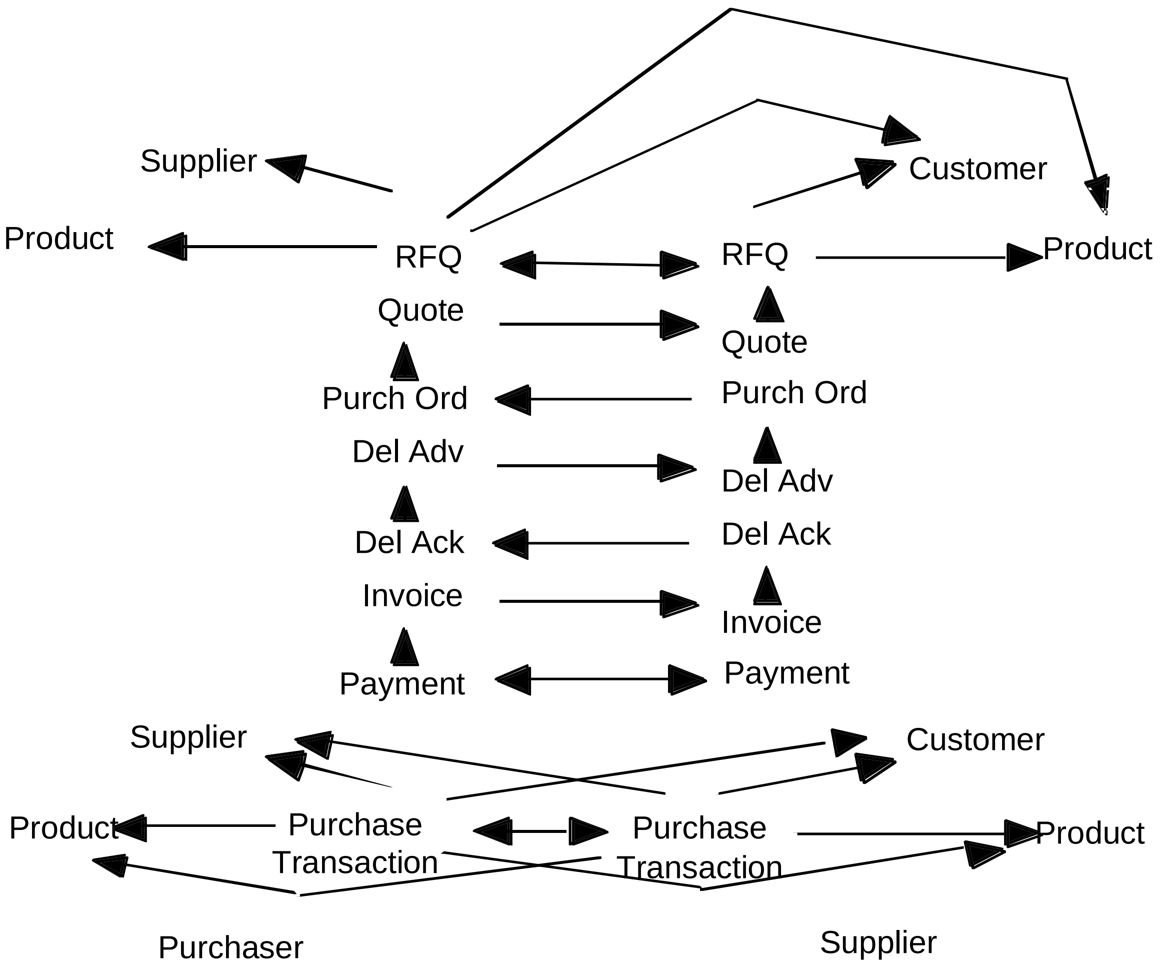

Figure 2: Purchasing and Supplier system interoperating

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 relationships involving *Product* on each side.

Product in each system is an independent entity. 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.

The relationships involving *Supplier* and *Customer* raise additional issues. It is normal for the Purchaser system to have an entity *Supplier* (as in Figure 1), and for the Supplier system to have an analogous entity *Customer*. However, what is an instance of *Supplier* for the Purchaser is the entire system of the Supplier, and what is an instance of *Customer* for the Suppler is the entire system of the Purchaser. As we have seen in the previous section, in isolated systems neither the unifying nor identifying relations for the whole system are generally represented. For interoperation, however, they must be. So the terminal object for each system must be mapped into an instance of an entity of the other.

Purchase Transaction in the lower part of Figure 2 is different from the entity *Purchase Transaction* in Figure 1. In the present case it is the exchange of messages which effect the interoperation between the two systems. 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* entities 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 relationship with an earlier message. The first message (*RFQ*) is shown with a one-to-one relationship between the parties – the relationship 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 relationship 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 relationships 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* relationships, taking advantage of the identification of the Purchasing system as an instance of *Customer* and the Supplier system 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* entity.

The other parts as they come into existence can be identified by their possibly indirect relationship 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 relationship 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 entities 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, as illustrated in Figure 3. Each successive state is reached on receipt of the next expected message, or on the decision by the organisation that the message will never come, therefore the transaction is finished, in the sense that no more parts will come into existence, but failed (*Failed Quote*, *Unable to Ship*, *Bad Debt*). The state where a whole transaction exists is *Complete*. The other states are all non-terminal, awaiting either a message from the other party or a report of an action by the owning party.

From a data perspective, the successive states look very much like successive subtypes (Dampney, 1987). Each state includes all the data of the previous state, plus more. Each non-terminal state includes a disjunction of possible continuations. Each continuation implies the disjunction. Thus the states are subtypes both from a set-theoretic (as one finds in databases and programming languages) and a proof-theoretic (as one finds in description logics) perspective.

However, the names of the states do not follow the semantic abstraction hierarchy normally used in exposition of subsumption hierarchies (thing, physical object, person, student, student in my class, etc.). In this kind of application, the states are named in a metonymic fashion. There is no requirement that the 'normal' natural language semantics of the names of subtypes be in subtype relationships in 'normal' natural language.

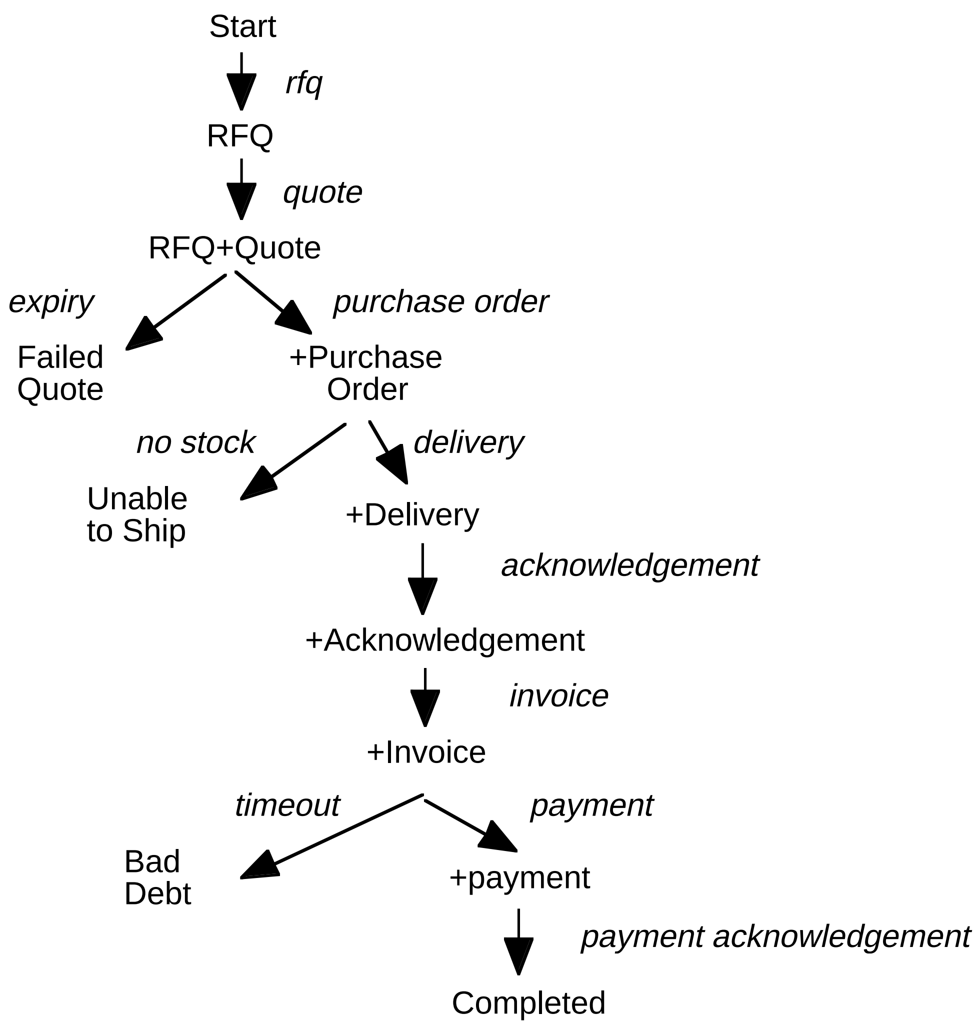

Figure 3: A state view of the coming-into-existence of parts of a transaction

If we take such a subtype view, we can take advantage of the distribution of identity over subtypes given by (Guarino & Welty, 2000a). Identity of a completed transaction can be supplied by *RFQ* or *+Purchase Order*, and carried down through the hierarchy. The failed transactions also can get their identity in this way – *Failed Quote* from *RFQ, Unable to Ship* and *Bad Debt* from *+Purchase Order*, for example. So also can the various incomplete but still active states.

In this way, the representation of *Purchase Transaction* as a whole with (temporal) parts has enabled us to derive a representation of the state spaces of each of the participating processes.

Having dealt with unity and identity, we can return to the question of existence for instances of the refinement into parts of *Transaction*. *GW* adapt an axiom from (Simons, 1987) that if x is a part of y then both x and y must exist (actual existence of parts).

Our application has a whole under construction. The entire function of the information system supporting the e-commerce exchange is to progress from one incomplete state to another. The axiom *actual existence of parts* leads us to consider the various failed transactions as wholes in their own right, as distinct from permanently incomplete transactions. In this view, in-process transactions are wholes which are subject to spontaneous change when further events happen. The state view is then part of the specification of the application, not of the ontology.

Alternatively, it is possible to consider expanding the ontology to include in-process states. The problem here is that it is difficult to do this generally, because there would be many different styles of state-transition diagrams because there are many different sorts of occurrences. A system like the e-commerce purchasing application has time measured by a finite collection of pre-defined temporal parts which can occur in restricted sequences, so the collection of completed and in-process occurrences are well-defined. On the other extreme, we have a person's life, where the transition possibilities are largely indeterminate. It may be too difficult for a very general ontology to cover all possibilities.

Discussion

We have analysed an electronic commerce sort of application storing institutional facts from the point of view of unity and identity, taking into account the refinement into parts of its entities. We have shown that the identity relations are partly logical and partly lexical (relying on a priori names included in the representations in some way). The unifying relations are therefore also partly logical and partly lexical. Unifying relations are needed for grouping instances into entities, and for the systems as wholes, since each system in the e-commerce interaction is viewed as an instance of an entity for the other. In particular, unifying relations are needed to bound the complex objects exchanges or shared. Furthermore, an attribute to be used in unity and identity relations must have a scope that encompasses all relevant parties.

For refinement into parts, we have seen that identity relations for the parts can be constructed from the functional relationships among them and from the inter-system mappings needed to define identity of objects across systems. These same functional relationships can be used as unifying relations for the assemblies of parts. A state view of the process of construction of the wholes from parts leads to a subtype view of the states of the construction process. The axiom of actual existence of parts leads to recognition of a number of permanently incomplete constructions as completed failed constructions of various types.

These meta-ontological concepts assist in the design of the information structures needed in the exchange of complex objects among organizations, which is how interoperation among systems is effected.

References

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

Dampney, C.N.G. (1987) “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 pp 143-164.

Dampney, C.N.G. (2001) Personal communication.

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

Guarino, N. and Welty, C. (2002). “Evaluating Ontological Decisions with OntoClean” *Communications of the ACM*, 45(2): 61-65.

Guarino, N. and Welty, C. (2000) “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.

Guarino, N and Welty, C. (2000a) Ontological analysis of taxonomic relationships, in Laender, A and Storey, V (eds) *Proceedings of ER-2000: The International Conference on Conceptual Modelling,* October, 2000 Springer-Verlag Lecture Notes in Computer Science No. 1920.

Johnson, M. and Dampney, C.N.G. (2001) “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.

Lowe, E.J. (1989) “What is a Criterion of Identity?” The Philosophical Quarterly, 39 1-21.

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

Searle, John R. (1995) *the Construction of Social Reality* New York:The Free Press.

Simons, P. (1987) *Parts: a study in ontology* Oxford University Press.

Weber, R. (1997) *Ontological Foundations of Information Systems* Coopers & Lybrand Accounting Research Methodology. Monograph No. 4. Melbourne.