

Chapter 3

Outsourced types

This chapter¹ addresses the problem of shared data. *Outsourced types* are introduced as an extension to traditional conceptual models that tie local definitions to external sources of information. Outsourced types ensure web applications are developed from the conceptual level with the capability to use and understand information specified in public ontologies, specifications and global standards.

An outsourced type is an abstraction mechanism that allows the conceptual modeler to delegate (or outsource) responsibility for the definition of the internal structure of the type. The modeler can attach requirements for capabilities to the outsourced type that provide the ability to manipulate and query an instance of the outsourced type without having to understand its internal representation.

In the next section a conceptual model is presented that describes a relatively complex case study which shows how outsourced types can be used in a global financial services context. Sections 3.3.1 to 3.3.3 motivate the need for outsourced type descriptions with the main focus on how to provide unique identities for instances of outsourced types in a global context. Section 3.4 introduces a meta-model for outsourced types. In section 3.5 some simple questions that can be used to identify the kinds of objects that may be candidates for outsourcing are suggested. Section 3.6 illustrates how conceptual models can be augmented to allow the generation of relational schemas for the storage of entities with multiple identities.

3.1 Introducing Object Role Modeling

Object Role Modeling (ORM)[83] is a visual conceptual data modeling technique, mainly used for the design of relational databases. It is used to describe object types and their relationships in a particular application domain. The advantages of using ORM are that it has an associated modelling methodology the Conceptual Schema Design Procedure (CSDP) and conceptual query language (ConQuer). It is implementation independent and has a formal semantics. It also has the advantage of being able to include a sample population, shown below the fact types, which

¹Based on work presented in [124]

helps to validate the model and demonstrate how it is used. A model that includes sample data is referred to as a “populated” ORM model or schema.

ORM and CSDP were selected to present the conceptual models in this thesis because of their ability to produce robust and graphic models that “rigorously capture the semantic nuances of an information system” (John Zachman in [83]). Section 1.4.1 discussed how ORM and ontology languages such as OWL have much in common. This similarity is exploited in this thesis by building conceptual models in ORM. This gives the models a formal foundation and the benefits of the CSDP. The validated ORM models can then be easily translated into OWL. This methodology avoids the problems that can be encountered when building conceptual models in languages that use an XML based syntax such as RDF and OWL-S because they cannot be easily understood and validated [33, 119, 136].

Some of the main concepts of ORM are described to assist the reader to use the schema presented in figure 3.1. The ellipses represent entity types or objects in the domain of discourse (e.g. *Request*). The boxes represent the roles these entities play in relationships with other entities.

A fact type consists of one or more roles depending on how many entities are participating in the relationship. A fact type can play a role itself in other fact types (e.g. the fact type *Collateral*), in a similar manner to reification in RDF.

Constraints are depicted in various ways. A double arrow over a role represents an uniqueness constraint over that role (e.g. a *Request* offers delivery from at most one Date), while a solid dot represents a mandatory role constraint (e.g. every *Request* has at least one Quantity representing the borrow or loan amount).

Constraints between roles, such as subset or exclusion constraints, can also be expressed in ORM. For example, an exclusion constraint between two roles states that any instance that participates in one of the roles cannot also participate in the other role.

When a string is put in parenthesis below the name of an entity type (e.g. *id*) this indicates the presence of a value type with a name which is the concatenation of that entity type name and that string. Instances of the value type uniquely identify instances of that entity type (e.g. *Requestid* is a value type providing identification for entity type *Request*).

3.2 Case Study

The case study is drawn from the field of securities² lending. In this domain, the owners of securities can lend those securities in exchange for cash collateral or other securities. Depending on the type of collateral, the owner will be recompensed by interest on the deposit of cash collateral, or a flat fee for non-cash collateral. Borrowers may have any one of several reasons to borrow securities, traditionally the

²Financial instruments or securities, in this context, are those traded on stock exchanges such as the NYSE, the LSE or the ASX.

most common, is to cover short sales³. Lenders usually participate in the transaction to generate additional income from their securities investments.

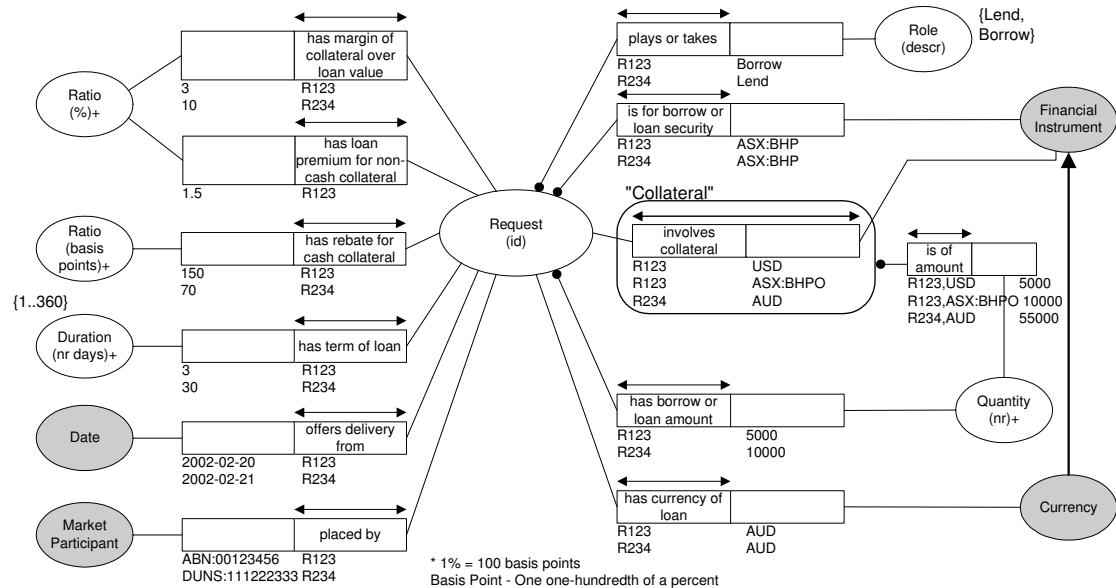


Figure 3.1: Securities lending conceptual model

The model (figure 3.1) shows the information necessary to describe a request that borrowers or lenders could place in a market to indicate a willingness to borrow or lend securities. The request itself is not a web service but provides the *context* for several associated services such as *Placing* and *Retrieving* loan requests. Requests could be constructed in many ways depending on the capabilities of the applications making and taking the request. A stored request can be serialised in XML [46, 17], RDF/RDFS⁴, or OWL⁵ to allow sharing between different applications. A request could be revealed as a web accessible form [50], or it could be revealed gradually as part of an interactive information service.

The shaded ellipses in the figure represent outsourced types. Outsourced types are necessary for this application, designed to operate in a global environment, for several reasons. The Date type (based on ISO 8601 [91]) is outsourced because date representations differ from country to country. For example 10/7/03 represents October 7 in the USA and July 10 in Australia. The Currency type (based on the ISO 4217 standard for currency codes [92]) ensures there is no confusion when applications deal with money outside of their national borders.

Financial instruments are treated as outsourced types because they are subject to many different identification schemes and we want to ensure that all parties are talking about the same thing. This is especially relevant in the context of cross border securities trading where failed trades, due to mistaken identity, cost millions

³Short sale - the sale of a security before it is purchased.

⁴www.w3.org/RDF/

⁵www.w3.org/2001/sw/WebOnt/

of dollars every year [107]. In addition, outsourced types allow capabilities to be attached to the type. This means that instances of the type can use functionality provided by external services, such as retrieval of the name of the issuer or the current market price for the financial instrument type.

The population in the model (figure 3.1) captures two requests. The request (*R123*) is to *Borrow* the *borrow or loan amount* of 5000 of the *Financial instrument* identified as *ASX:BHP* shares⁶. The request offers two types of *Collateral*, the *Currency USD* (US dollars cash) and the *Financial instrument* identified as *ASX:BHPO* shares. The *margin of collateral over loan value* is how much more collateral, over and above the loan value, is required by the lender. The lender's return is the *loan premium for non-cash collateral* of 1.5% on the *ASX:BHPO* shares and the *rebate for cash collateral* of 150 basis points⁷ on the cash value for a 3 day loan.

Over the duration of a securities loan, the value of the loan and the consequent value of the collateral required will fluctuate depending on the value of the financial instrument and the current exchange rate between Australian and US dollars. The runtime re-calculation of the value of the loan means several operations are required to provide this information. A requirement for the capability to retrieve the current market price of shares can be attached to *Financial instrument*, and the capability for exchange rate conversion can be attached to the *Currency* type.

3.3 Outsourced Types

There are three main types of information that can be associated with an outsourced type. These are discussed in the following sections.

3.3.1 Information Sources

A web-based application should be able to provide clarification of the terms it uses by referring to definitions, ontologies and other sources of information. The association of other sources of information with the outsourced type is primarily for the benefit of inference tools and mediators [55], it allows modelers to explicitly provide the semantics of the terms used in this context. As application terminology is often unique to a particular designer or software provider, support for dynamic runtime disambiguation and inferencing must be provided by the service itself. In addition, the use of common type systems such as external standards for the internal representation of data provides interoperability with all the other programs using the same standards. References to type definitions and alternative sources of information will enable inference engines to disambiguate the terms used in this context and is a practical step towards the goal of ad hoc interoperability.

The outsourced type *Date* demonstrates how external sources of information can be used. The outsourced type definition can refer directly to the ISO 8601 Date

⁶ASX = Australian Stock Exchange, www.asx.com.au

⁷Basis Point - One one-hundredth of a percent

specification, or it can take advantage of the XML Schema Datatypes specification or schemas developed for use with RDF such as the Dublin Core (DC) element set⁸ both of which are based on ISO8601. For those applications that are not aware of the ISO standard or its syntax (ccyy-mm-dd), the outsourced type definition can also specify a requirement for the capability to translate dates to and from ISO format.

3.3.2 Service Descriptions and Capability Oriented Modeling

The lack of prior agreements between interaction partners means that there may be mismatches between the data required for a web service invocation and the data available. In this case capability descriptions can be used to describe the operational capabilities that may be required for the outsourced type in this context. Some of the operational capabilities will be generic, such as *creating instances*, *comparison*, *format translation* and *getting* and *setting* values, others will be specific to the type, such as retrieving the current market price from a financial instrument. A service that provides a capability may also describe non-functional requirements and constraints. A description of these can be found in [127]. At present, it is not possible to specify this kind of capability requirement in ORM models [158]. Outsourced type definitions allow the specification of capability and non-functional requirements at a conceptual level, where the description is in terms of *what* is required rather than *who* by, or *how* the operations should be provided (implementation).

The delegation of capabilities or tasks to external services is a natural extension of the object oriented and component programming paradigms. This kind of modularisation allows applications to concentrate on core competencies and delegate less relevant concerns to specialist services in non-core areas. At the conceptual level the modeler is only responsible for providing a description of the capabilities that may be required. At runtime, the choice of which service implementation to provide the capability should be based on how well the service fulfils the requirements and non-functional constraints specified.

3.3.3 Identification Schemes

In the global environment there is often more than one externally controlled identification scheme and entities may have valid identities in several schemes. In the past, each application could provide its own unique identification mechanism for the local application context but as applications globalize it is difficult to maintain uniqueness constraints over relationships when the same entity may have several valid identifiers in different schemes. Here the issue of entity or object identity is addressed rather than the issue of personal identity on the web covered by products such as the Microsoft Passport and frameworks such as [126, 89]. In UML models

⁸dublincore.org/documents/1999/07/02/dces

in particular, the identity of entities and objects is implicit rather than explicit as in ORM. The advantage of providing explicit identification is the ability to identify specific instances in terms that are meaningful outside of the system being modeled. An extension to the UML meta-model to allow explicit identity has been proposed in [44].

In ORM each instance of an entity must be able to be uniquely identified. An application that has outsourced entity types must be able to use different identification schemes *while ensuring that different identities for the same instance are recognised*. The use of an identity in combination with its scheme means the same identifier in a different scheme identifies a different instance. The use of external identification schemes means the responsibility for enforcing constraints such as a constraint that *identities within a scheme must be unique*, is delegated to the scheme's controller.

Each identification scheme will be owned or controlled by a different organization and each will have different rules and jurisdiction over the target group of entities to be identified. The coverage of identification schemes may range from fully disjoint to fully overlapping, depending on the aspects/properties/roles of the entity that the scheme is interested in [79]. Similarly the scope of schemes may not be equivalent. For example, the Australian ABN scheme [8] simply states that an Australian business exists and is registered (to pay tax), whereas a D-U-N-S⁹ implies some rating or assessment has taken place before the identifier is issued. Identification schemes themselves may have properties such as trustworthiness, accessibility, reliability and quality.

It may be necessary for some applications to limit the identification schemes that can be used; either by limiting the number of schemes, or by specifying acceptable schemes by name. The limitations on schemes could also be instance dependent. For example, if a Person is involved in a relationship "pays tax" and the person is from USA then only accept USA TFNs or USA SSNs to identify that Person.

In some cases it may not be possible to determine if an instance has an identity in a particular scheme, due to limitations on access to information without payment of a subscription or because of security concerns. Security is also an issue when it is necessary to pass identity information to downstream services, there may be user constraints on whom the information can be passed to, or how the information can be used.

There are two conceptual modeling issues related to multiple identities that are a consequence of outsourcing types in a global context. The first issue is that there may not be an *exhaustive* list of all the possible identification schemes to start with, hence the model will not "know" about new schemes introduced after the modeling process is complete. Consequently a form of schema evolution is required to introduce new identity schemes. In terms of our meta-model (figure 3.2), an evolution mechanism for the introduction of new identification schemes would require information about the organization that controls, owns or publishes the scheme; whether or not its

⁹www.dnb.com

origin is based upon some other published information, such as a standard or law, and a source of information about the scheme itself.

The second modeling issue concerns ensuring the *uniqueness constraints* described in the base model are not violated when using instances of outsourced types. There are several options to prevent constraint violations at the implementation level. One option is to choose one or more of the external identification schemes to act as the canonical representation within the local application. A *cover* constraint is a requirement that all possible instances can be identified by the selected identification scheme or schemes. Therefore the schemes should be selected to maximise the coverage of instances and minimise the possibility of overlapping identities.

In terms of the case study, we could choose the ISO 15022 standard for International Security Identification Numbers (ISIN) [157] as it is an internationally agreed standard for the identification of securities. In practice it cannot be used on its own yet as it is not implemented in all countries. This means the ISIN would have to be used along with all the identification schemes used in all the countries not yet covered by the ISIN scheme.

Another option is to create an internal *canonical* identification scheme, and map identities from multiple external schemes to the canonical identifier, typically with the help of external identity conversion services. In most applications, including the case study, it will be necessary to record which identification scheme a client prefers to enable the reverse mapping back to the representation the client understands. A mechanism to do this is discussed more fully in section 3.6.

The advantage of the locally defined canonical identifier approach is that complete coverage can be guaranteed, at least internally. Complete coverage ensures the single canonical identification of any instance; and any instance previously unknown can be assigned a new identifier in the local scheme. The use of a single (locally) controlled canonical identifier also ensures that uniqueness and other constraints relating to the roles an outsourced type plays in a schema can be checked and validated locally.

The identification of Financial Instruments is just one example of the overlapping and duplication of identification schemes in the global context. In the first case above, several existing schemes could be chosen to provide complete coverage. However it will be difficult to find a manageably small number of schemes so that this can be done without overlapping identities. For this reason we tend to prefer the second option of a single canonical representation. The single canonical representation can be managed by a single party that has the potential ability to: ensure uniqueness constraints are preserved, include new identification schemes, and generate new identifiers when appropriate.

3.4 Meta Model

Figure 3.2 is a populated ORM meta-schema for outsourced types. Outsourced object types are a subtype of ORM Unnested Entity Types as defined in the ORM

meta-schema [82]. They participate in three relationships which match the three types of information identified in section 3.3.

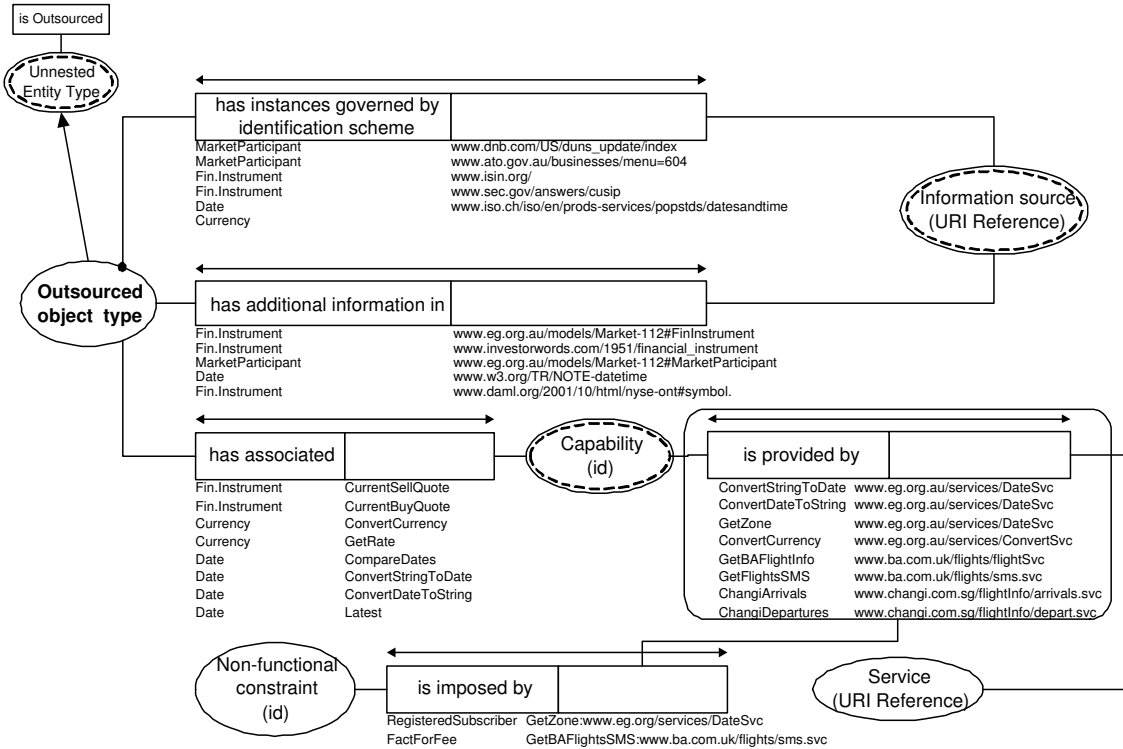


Figure 3.2: Meta-model of outsourced object types

The first is the mandatory role *has instances governed by identification scheme*. The mandatory role constraint means that there must be at least one way of identifying the scheme that issued the outsourced type’s instance identifier. This rule makes explicit the entity identification rules inherited from ORM while recognising that instance identifiers may come from different identification schemes. For example, market participants may be identified by a Dun and Bradstreet D-U-N-S number as described in “www.dnb.com/US/duns_update/index”, and/or an Australian Business Number (ABN) as described in “www.ato.gov.au/business/menu.604”.

The second role named *has additional information in* allows other information to be associated with the type, such as a definition in a thesaurus or ontology. The information is intended to be used to aid the comprehension of the syntax and semantics of the type and its roles in this context. For example the information source “www.investorwords.com/1951/financial_instrument” provides relevant information about Financial Instruments.

Both of these two roles are played in a relationship with an Information source. The double line around Information source identifies it as an *external type* which means it is described in another ORM model. Figure 3.3 contains the Information source schema.

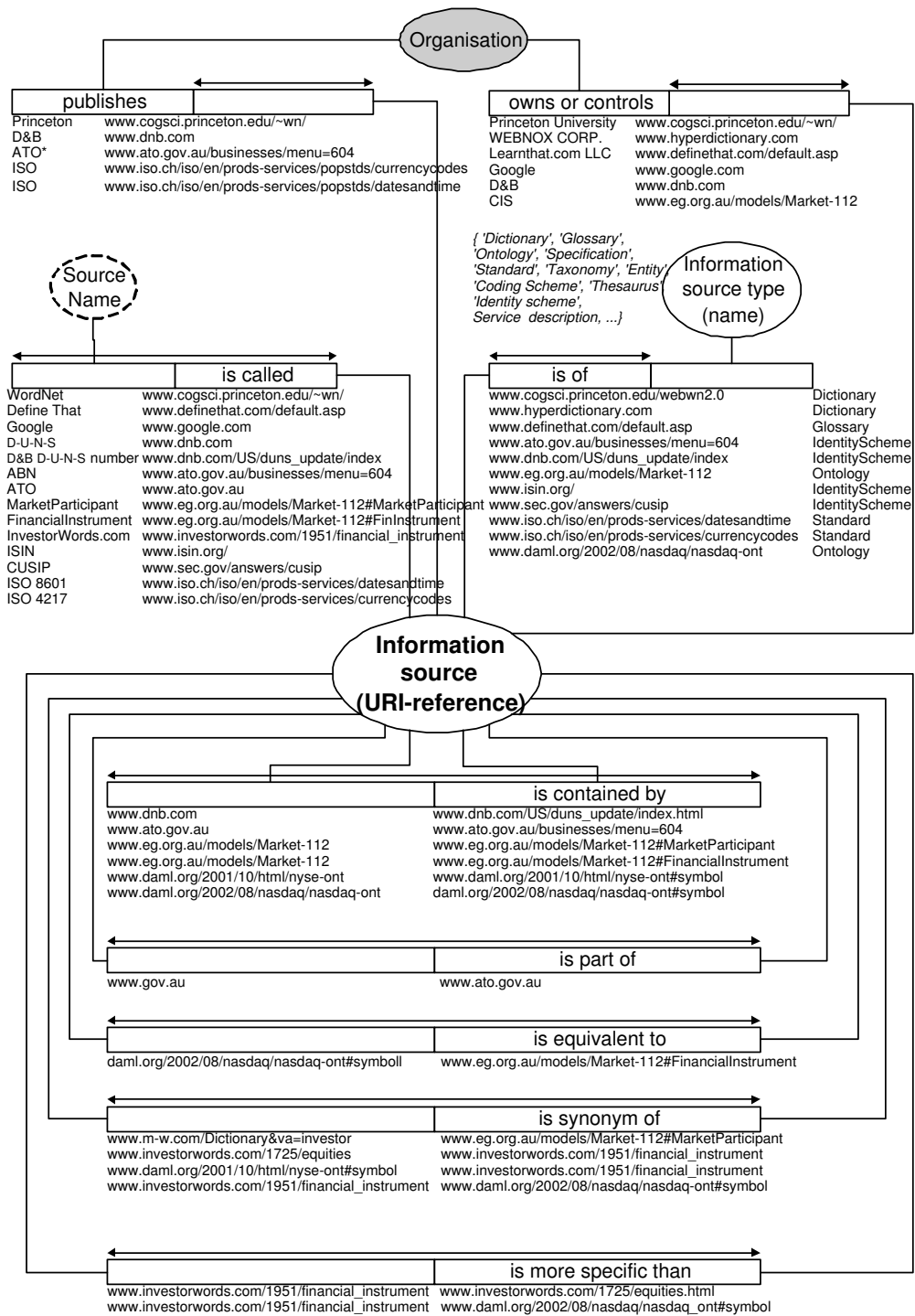


Figure 3.3: Information sources for outsourced object types

The third role played by outsourced types in figure 3.2, named *has associated*, is attached to another external type Capability. Capabilities are introduced in chapter 4 and the model is shown in figure 4.1. The *has associated* relationship allows various

requirements for downstream capabilities to be associated with an outsourced type and its instances.

Capabilities are *provided by* services. A “Provider” may impose non-functional constraints on the delivery of capabilities. The *provided by* relationship is a many to many relationship this means that a capability *can* be delivered by the named service but other services may also be able to deliver this capability. For example, one of the capabilities that can be associated with an instance of a financial instrument is identified as “CurrentSellQuote”. This capability retrieves the price sellers want to receive when they sell their securities. This capability can be delivered by the service at “www.ex.org.au/services/MarketDataSvc” or it may be provided by other services. The many to many relationship also means that “www.ex.org.au/services/MarketDataSvc” may also provide other capabilities.

Figure 3.3 describes the roles an Information source can play. The model has roles to represent the common conceptual *mappings* referred to in section 1.4.1, such as synonymy, equivalence and hypernymy. These roles are played in relation to other Information sources.

An Information source is identified by a URI Reference. This web based identity scheme may exclude some information sources that do not have namespaces. This appears to detract from the applicability of the model but as we are dealing with web based applications a machine accessible unambiguous identification scheme is preferable in this context.

Information sources play four other roles. They are *published* by an Organisation and *owned or controlled* by some Organisation. Information sources are often known by their common names rather than their URI. The mapping between names and the URI is the purpose of the *is called* role.

In some circumstances it may be that the purpose or role of an Information source can be further described by reference to its *Information source type*. For example, an Information source that *is contained by* an Information source that *is of* Information source type Dictionary will be a definition. The value constraint attached to Information source type gives an indication of the kind of values that may be used such as dictionary, standard, specification, identity scheme and ontology and so on.

The two models in figures 3.2 and 3.3 describe the meta model for outsourced types. The meta model describes the abstract syntax of the proposed extension to conceptual models. An abstract syntax captures concepts and their inter-relations but does not deal with aspects of concrete representation. A concrete representation appropriate for web application descriptions is necessary but outside the scope of this work.

A translation of the Information source model is provided in appendix A to give an indication of how an ORM model can be translated into a web accessible OWL ontology.

3.5 Identifying Outsourced Types

Although ORM already has the *external type* construct to represent information found in another ORM model, outsourced types allow ORM models to include external descriptions from many other sources.

The Conceptual Schema Design Procedure (CSDP) is a seven step method for creating a conceptual schema. Steps 1 to 3 are concerned with identifying entity types, value types and fact types. Steps 4 to 7 of the CSDP are when the constraints on entity types and the roles they play in relationships, are identified and incorporated into the schema diagram. The decision to use outsourced types can be taken when all the elements in the domain model have been identified, therefore the decision can be made after step 3 of the CSDP.

Three questions help to decide whether an entity type should be defined as an outsourced type. At the conceptual level the concerns are to manage complexity, to reuse external definitions when they are appropriate and to prepare for interaction as a web-based application.

Elementary types: The first question asks whether the object type represents an elementary or complex type. Elementary objects representing a number such as *Quantity* in figure 3.1 will not usually be treated as outsourced types. However, to promote interoperability elementary types could be defined in terms of an external definition such as XML Schema Part 2: Datatypes¹⁰.

Imported definitions: The second question asks whether there already exists a definition for this object type. These definitions can come from any model, schema or definition that is not the current model.

At the beginning of our investigation, it was assumed all imported definitions should be treated as external, that is, supplied with an outsourced type definition for onward exposure to clients of the application being modeled. However as the meta-model has been developed, this requirement has become blurred. On the one hand an imported element could be treated as an outsourced type only if it is also exposed to users. On the other hand, outsourced types can be used as a guide for software developers in the way an element should be implemented because the more external information is reused in local models, the greater will be the basis for shared understanding and interoperability. Reuse of publicly accessible information provides cost effectiveness, consistency and interoperability both within an organisational context and in the global world of web applications.

Exported definitions: The final question asks if the object type is to be exported or exposed to clients by applications using this model. Exposure can either be direct or indirect. Direct exposure happens when objects are shared as part of the normal operation of an application, such as function names, parameters, return values or message components. Indirect exposure happens when up or downstream services require clarification of the semantics of the terms used by the application. For example a user of the securities lending request, may ask “what is a financial

¹⁰www.w3.org/XML/Schema

instrument?”, and the application could return the URI of a dictionary definition, or an ontology entry, or a list of alternative terms.

3.6 Realisation

Outsourced types provide a means to describe entities in terms of external information, however web-based applications will need to store this information locally. This section bridges the gap between the extension to the conceptual model and databases to store information about outsourced types. It illustrates how traditional ORM models must be modified to recognise multiple identity schemes.

The existence of multiple external identity schemes is not evident from the representation of outsourced types in the local ORM model. This means the relational mapping procedure (Rmap) [83], used to convert an ORM model to a database schema, will be unaware of the multiple identities that may be associated with an instance of an outsourced type. The following examples illustrate how the model must be augmented as a precursor to Rmap. The augmentation procedure substitutes a canonical identifier for each external identifier and its identification scheme, while retaining information about which scheme is being used in each role. There are many cases that could be considered, each representing the different relationships (and their constraints) that an outsourced type can participate in with other outsourced and non-outsourced types. This section illustrates two sample cases that represent different uniqueness constraints on *binary* relationships between an outsourced and a non-outsourced entity type.

The augmentation process is done in two stages. In the first stage, a canonical identifier is introduced in place of an external identifier and its associated identification scheme. To retain knowledge about the relationship between: the canonical identifier, the external identifier, and its associated identification scheme, a ternary relationship (henceforth called the identity catalogue) is created. The identity catalogue is attached to the outsourced type, which now uses the canonical identifier as its primary reference.

The identity catalogue is used to record *all* the identifiers and schemes that are known or can be found for a particular canonical identifier. There are two ways to build this list of schemes and their identifiers. The first option is to actively search for alternative identities and schemes when the database table is first created; the second option is to check and add new schemes when they are introduced to the application. The options represent a trade-off between the possibly lengthy time taken to populate the table when it is created and quick updates; or a longer time taken to make updates while new information is checked and incorporated into the table.

In the second stage, a new binary relation is introduced to make explicit the association of the canonical identifier and the identification scheme used in a specific role. The association of this new relation with the entities in the model depends

upon the uniqueness constraints governing the whole relationship. This is illustrated below in relation to the one to many and many to many relationships.

Configuring a One to Many relationship. Figure 3.4(a) is in two parts, the top section shows the original relationships between an outsourced type (Person) and local entity types (Event and Country). A problem with this model is the identification scheme for *Person* is a combination of the name of an identification scheme and a value in that scheme. In the example, a Person may only represent one country but cannot determine from the identifiers, TFN:z024 and SSN:z021, that these really are different people. For this reason a canonical identifier (cid) is created for each outsourced entity instance and the cid is substituted for the composite identification in the original *participates in* and *represents* roles.

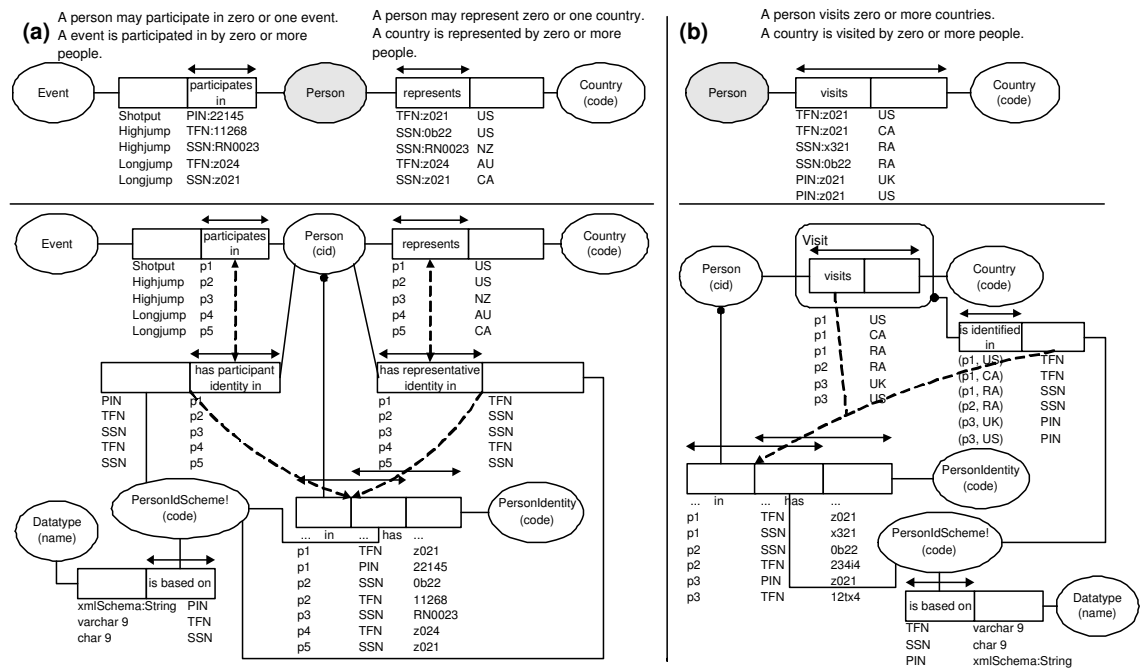


Figure 3.4: One to Many and Many to Many relationships

The lower part of the figure is extended with an identity catalogue, to make the relationships between the cid, and all its related identification schemes and identities explicit. The identity catalogue is subject to two uniqueness constraints. The first is over the external identifier and its identification scheme. This constraint states that each identifier is unique within its identification scheme. The second constraint states that each combination of cid and identification scheme is unique. The combination of these two constraints means that a person with identities in different identification schemes can be uniquely identified by a single cid.

The next stage in the translation procedure is to make explicit the relationship between the cid and the identification scheme used in a particular role. The equality constraint between *participates in* and *has participant identity in* ensures that when a cid appears in the *participates in* relation, then this cid and the identification

scheme used for this role are recorded in the new relation *has participant identity in*. This relation records which specific scheme, amongst potentially many schemes, is relevant for the *participates in* relation. Similarly a new relation, *has representative identity in*, is constructed to represent the relevant identities and schemes for the *represents* relation. As both *participates in* and *represents* are one to many relationships with each *Person* only appearing once in each population the new relationships are attached to *Person*.

Configuring a Many to Many relationship. Figure 3.4(b) shows a visitation relationship between People and Countries. The primary concern in this case is to ensure that each row of the visits/visited by relationship is unique. In the original schema we would be unable to enforce this constraint given the current information. For example, *TFN:z021* and *SSN:x321* are different identity codes, but are alternative identifiers for the same Person instance (p1).

Although there are alternative ways to prepare the many to many conversion for Rmap, in this case it was decided to introduce an objectified relation *Visit* to represent the necessary information. The uniqueness constraint over the *Visit* roles ensures the original uniqueness requirement (that there are no duplicate rows) is maintained. A join subset constraint is placed over the roles involving a *Person* and a *PersonIdScheme* with the corresponding roles in the identity catalogue. This constraint means that for every fact in which a certain Person is used in conjunction with a Person Id Scheme, we should have information about this combination in the identity catalogue.

3.7 Discussion

In this chapter outsourced types were introduced as an extension to classical conceptual models. Outsourced types are a practical step towards achieving the goal of automated ad hoc interaction between web-based applications.

Outsourced types provide several benefits. Firstly, they allow designers and developers to *reuse* existing definitions from local and global sources. Outsourced type definitions can also be reused as they are defined at a conceptual level outside of the scope of programming language, application and enterprise boundaries.

Outsourced types help to reduce the reliance on particular representations by allowing the use of alternative definitions (standards, specifications and ontologies) thus helping to avoid problems such as vendor lock-in. Another advantage of alternative definitions is that there will be a greater probability of interaction partners understanding at least one of the offered definitions. Alternative definitions also provide redundancy in a possibly unreliable web environment.

The outsourced type definition also provides the basis for downstream service composition by allowing specific statements of what capabilities the outsourced type expects. Outsourced type definitions could be registered as tModels in a UDDI registry. The information provided by outsourced types provides an explicit declaration

of the semantics intended by the service provider, thus reducing the cognitive workload on mediators such as those proposed for WSMF [32].

In the more general area of modeling of web applications, Atzeni et. al. [7, 6] draw data from HTML web pages, and selectively rearrange and amalgamate it for presentation as new HTML pages. In contrast, outsourced types selectively draw information from a wide variety of sources in order to be able to describe aspects of a web application to users.

In the context of semantic web services, middle agents [63], data mediators, or planning agents will be used by clients to evaluate, select and compose services with disparate data models. A comprehensive introduction to the issues associated with semantic data integration is given in [104]. Semantic data integration will be facilitated by the extended and alternative descriptions of data elements provided by outsourced types.

The definition and creation of outsourced types will place a greater work load on web service providers but the benefits of these reusable definitions are likely to outweigh increased costs.

Looking ahead: outsourced types form a foundation for the chapters that follow.

Chapter 4 discusses how the context and operations or capabilities of services can be described for discovery and use across the web. Using outsourced type definitions to describe aspects of capabilities gives all the benefits of outsourced types, such as grounding terms in external sources, to the web service description layer. Note that capability descriptions were also found in outsourced type descriptions where they are used to attach functionality to types in a similar manner to the association of methods with objects.

Outsourced types are also essential to the efficient working of the interaction mechanism described in chapter 5. Outsourced types provide the contextual grounding of terms used by both parties to an interaction and allow them to come to agreements on substitutable or alternative terms by reference to the information sources attached to the outsourced type descriptions.

