A Concept Representation Language (CRL)

Paul C. Brown
Guilderland, NY, USA
pcbarch1@gmail.com

***Abstract*— Formal languages, each with its own syntax and semantics, enable precise communications within specific domains. However, when describing most real-world situations, no single language provides a comprehensive description: multiple formal languages are required, and these must be precisely correlated with one another. This begs the questions as to what formal language can be used to express this precise correlation – the relationships between the elements of two or more formal languages? There is almost a total lack of such languages and, in their absence, most mappings are expressed procedurally with code that is specific to the referenced languages. This paper introduces a Concept Representation Language, capable of uniformly representing concepts and relationships at all levels of abstraction as well as the mappings between them. The paper covers the motivation, design principles, and infrastructure provided by the language.**

Keywords-concept; relationship; mapping; relationship mapping; representation; representation mapping; abstraction; meta-level

I. Introduction

Natural languages, while flexible, tend to lack precision. Where precision is required, formal languages are created: algebras in mathematics; programming languages and data languages in computer science; conceptual models for semantics. There are even formal languages to describe the syntax and semantics of natural languages. These formal languages enable precise communications in specific problem domains. They embody a wide range of concepts, from hardware-level programming (e.g. the MOV assembly language instruction that moves the contents from a register to a memory location) to conceptual semantics (e.g. the is-a concept, as in an automobile is-a vehicle) to mathematics (e.g. a morphism is a structure-preserving map from one mathematical structure to another).

While each language facilitates communications within its intended domain, each represents but a fragment of some world – real or abstract. To represent and communicate a more complete understanding requires employing multiple languages. Data fusion is one example. It seeks to combine information from two or more domains to create a more comprehensive representation of a situation [1] [2]. OMG's Semantic Information Modeling for Federation (SIMF) effort seeks to map data between different levels of abstraction to facilitate the transformation of data from one physical data structure to another (Fig. 1) [3]. Using this approach, an XML file containing information from a banking domain might be transformed into a JSON representation of information in a risk domain.

These efforts expose an underlying problem: the languages often have different representation schemes based on different

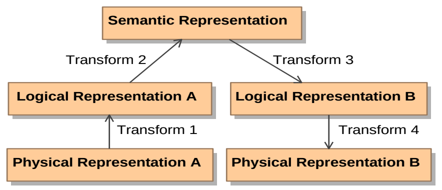

Figure 1. The SIMF Concept

sets of concepts. To work with these multiple domains, the concepts need to be related to one another. These relationships themselves require a representation language, one that subsumes the representations and concepts from both source languages. As the number of languages and domains increases, so does the complexity of the language expressing the relationships. This complexity presents a significant practical barrier.

Two factors drive this complexity: the variety of the concepts involved and the variety of language representations for those concepts. While the variety of concepts is arguably the inherent complexity of the problem, the variety of representation schemes is simply an accident of history – one that can be avoided. Any representation language that is capable of representing the relevant concepts and relationships can be used. So how about a representation language that can represent all concepts and relationships?

The Concept Representation Language (CRL) is designed to provide a universal solution to the representation problem. CRL is a language for representing concepts and the relationships between them in a uniform and consistent manner. It is intended to be able to represent any discrete concept and any discrete relationship between concepts. It is conjectured that there is an isomorphic mapping from any representation in any other language to a CRL representation. With these capabilities CRL provides a single language that can represent all discrete concepts and all discrete relationships.

II. A Concrete Example

The problem under consideration is common in the IT world. An order management system might generate an XML representation of a customer, the customer's orders, and the related shipments as shown on the left of Fig. 2. A corresponding UML canonical domain model is on the right. There are two differences in these representations that we will utilize to motivate the discussion. The first is that the XML representations on the left are hierarchical: a customer contains orders which, in turn, contain shipments [4] [5]. Containment is a one-to-many relationship. In the UML representation on the right these are peer-to-peer relationships. Specifically, there is a

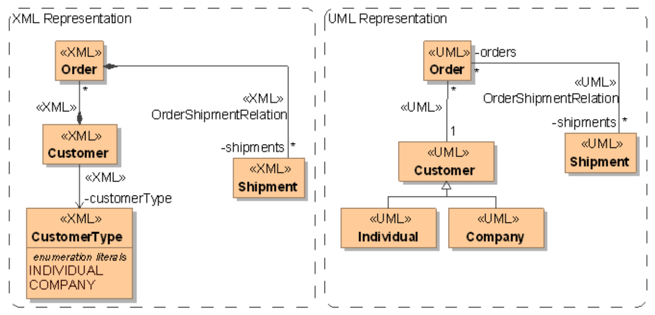

Figure 2. Differing Representations of Customers, Orders, and Shipments

many-to-many relationship between order and shipment in the UML model [6]. This reflects the ability for the enterprise to do consolidated shipping: one shipment may contain goods from multiple orders.

The second motivating difference is the manner in which the different types of customers, individuals vs. companies, are represented. In XML, the distinction is made via the Customer attribute customerType, with allowed values of INDIVIDUAL and COMPANY. In the UML representation, Individual and Company are represented by different classes with a common Customer parent class.

The practical challenge is to express the relationships between these two representations in a precise and unambiguous manner. Specifically, how do we represent the mapping between containment relationships and peer-to-peer relationships and the mapping between enumerated values (instances) and classes?

III. Languages Make Ontological Commitments

To understand the CRL design, let's take a look at some language characteristics that keep many other languages from being able to play this universal role.

Most formal languages force the language user to make ontological commitments when expressing concepts and relationships [7] [8]. These are assumptions about the categories (kinds or types) of things that the elements of the language represent. English, for example, distinguishes between nouns, verbs, adjectives, and adverbs. UML distinguishes between classes, associations, activities, and packages. RDF has notions of triples, graphs, and terms. Most mathematical languages distinguish between operators and operands.

These ontological commitments – these built-in type distinctions – enable efficient communication within he intended domain of discourse. But when it comes to relating one domain to another, they actually present obstacles: the ontological commitments (the types) in one language do not necessarily align exactly with those of another. When we want to relate the elements of an XML data structure to the elements of a UML model, for example, we need a way to address these differences in ontologies. More generally, if we want to express relationships between languages, we need to be able to also identify and relate their ontological categories.

Let us consider the problems that arise when we try to use UML as a tool for modeling multiple languages as well as the relationships between them. This exercise is not meant to denigrate UML in any way, merely to point out its limitations in playing this unintended role. Our purpose is to identify the challenges so that we can craft a representational language without these limitations. Similar representational issues can be found in nearly all languages, including RDF.

A. Challenge 1: Entities vs. Relationships

Consider the problem of mapping of the XML representations of Orders and Shipments to their corresponding UML representations (Fig. 3). In UML, an association is the representation of a relationship between entities known as classes. Graphically a class is represented as a rectangle and an association is represented as a solid line between classes. Using this approach, the XML Order and Shipment and the UML Order and Shipment are represented as classes (rectangles) while the both the XML and UML OrderShipmentRelations are represented as solid lines. Similarly, the Order Mapping and Shipment Mapping can be expressed as associations between classes.

But we run into a problem when we want to represent the relationship between the XML and UML OrderShipmentRelations: a UML association cannot be used to represent an association between associations. Representing this in UML requires a change in the way in which the OrderShipmentRelations are modeled. Each of these associations would have to be converted into what is known in UML as an association class (a class rectangle with a dotted-line connection to an association solid line) and then the two Association Classes could be related by an association (Fig. 4).

Why is this a problem? Assume the UML models for both the XML and UML representations already exist as shown in Fig. 3. Given these existing models you now want to represent the mapping between the OrderShipmentRelations. But you

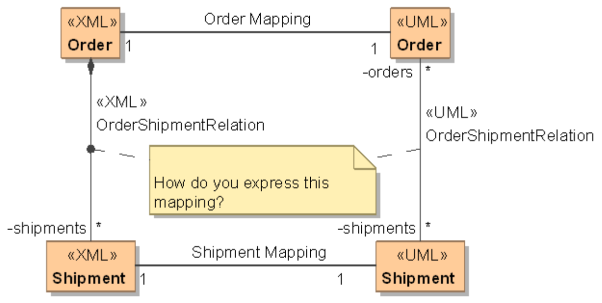

Figure 3. The Relationship Mapping Problem

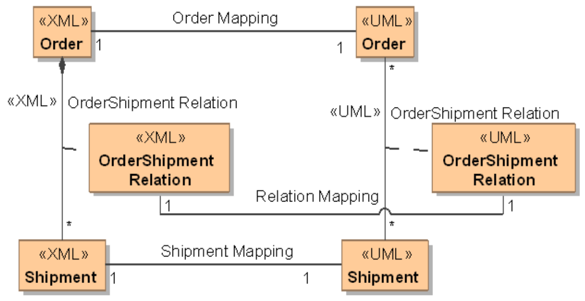
«XML» Order
Order Mapping
1
1
«UML» Order
«XML» OrderShipment Relation
«UML» OrderShipment Relation
«XML» OrderShipment Relation
«UML» OrderShipment Relation
Relation Mapping
1
1
«XML» Shipment
Shipment Mapping
1
1
«UML» Shipment

Figure 4. UML Representation of Relationship Mapping

can't - not without modifying the existing models of both languages.

What this example illustrates is the consequence of not treating relationships (i.e. associations) as first-class citizens (i.e. classes) in the representational language. This makes it impossible to represent a relationship between relationships. While UML does, indeed, have a common conceptual abstraction between classes and associations that could play this role (it is called the classifier), this concept is not instantiable (representable). To represent a concept in UML you must make an ontological commitment: it is either a class or an association – or an association class.

This leads to the first representational principle: *Relationships must be first-class elements in the representational language: It must be possible for a relationship to relate any element to any element, including another relationship.*

B. *Challenge 2: Types vs Instances*

Another common ontological commitment requires specifying whether a given element is a type or an instance of a type. Fig. 5 illustrates the challenge of mapping the XML customer representation into the corresponding UML representation. While the XML Customer (a class) can be associated with the UML Customer (also a class), UML provides no mechanism for associating instances (the enumeration values of the CustomerType) with classes: instances are at a different meta-level than classes. Other than the generic UML dependency (used in the diagram), the only supported UML relationship between classes and instances is the instance relation, which is not appropriate here: The INDIVIDUAL instance of CustomerType is not an instance of the UML Individual class.

The problem here is that UML represents instances differently than types (classes) and does not have a general mechanism for modeling relationships between instances and types.

A similar issue arises with elements at higher metalevels (Fig. 6). In UML, classes (e.g. the XML Customer the UML Customer) are instances of the metaclass Class that is part of the UML specification. This relationship is not explicitly representable, which makes it impossible to map the class-metaclass relationship in one representation to a different kind of relationship in another representation. Furthermore, UML

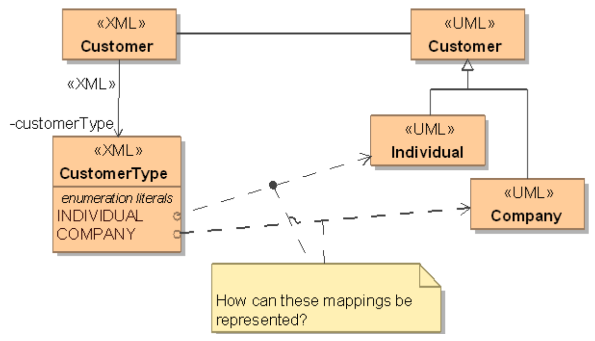

Figure 6. Mapping Instances (Enumeration Values) to Classes

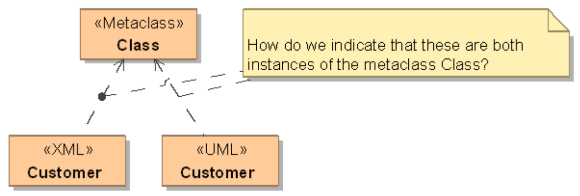
«Metaclass»
Class
How do we indicate that these are both instances of the metaclass Class?
«XML»
Customer
«UML»
Customer

Figure 5. Elements at Different Meta-Levels

itself provides no mechanism for denoting relationships between metalevels other than the generic dependency.

Note also that XML Customer and UML Customer play the role of the class (the type or category) in Fig. 5 while the same elements play the role of instances in Fig. 6: the notions of type and instance are relative, not absolute. In other words, they are both relationships.

This leads to the second representational principle: *A representational element should be able to represent a concept at any level of abstraction without making an ontological commitment to the level of abstraction. Thus, the same element can represent both a class and an instance of some other class. More broadly, the notion of a meta-level is just a concept that can be represented and the membership of a given element in a given metalevel can also be explicitly represented as a relationship within the model.*

IV. Representational Desiderata

Before getting into the design of CRL, one thing needs to be made perfectly clear: ontological commitments are useful. What is needed is a representational language in which ontological commitments are not forced by the representational language but are explicitly representable within the language.

So, what does our representational language need to do?

1) *Represent a discrete concept*
2) *Represent a reference to an existing representation*
3) *Represent the composition of representations to form more complex representations*
4) *Not force any ontological commitment to any level of abstraction*
5) *Represent that one representation is a refinement of another representation*
6) *Represent relationships in such a way that relationships can, themselves, be related*

The notion of refinement subsumes the notions of both generalization and instantiation since the representations being related are uncommitted with respect to their level of abstraction. Both refinement and composition are relationships between representational elements.

While these capabilities provide the ability to represent arbitrarily complex concepts and relationships, the resulting representations will contain much fine-grained detail that generally comes into play only when detailing mappings. Since such detail can be distracting to people, we add one more requirement:

7) Enable simple (abbreviated) representations of full representations when full explicit detail is not required. These representations simply elide the display of the detail.

V. CRL Core Ideas

With these representational capabilities in mind, we now turn to the design of the Concept Representation Language (CRL).

The first requirement for CRL is to be able to represent a concept – any concept, at any level of abstraction. This is the role of an Element (Fig. 7). Each Element has an identifier that provides a unique identity (e.g. a GUID) for the concept being represented. Of course, without further information it will be unclear to a human reader which concept is being represented: this is addressed later in the discussion of grounding.

The second requirement is to reference an existing concept. The Reference serves this purpose. It has one attribute, the referencedElement, which is a pointer to the Element being referenced. A Reference is a refinement of an Element, i.e. every Reference is an Element. In Fig. 7 we are using the UML Generalization notation to represent refinement. Refinement, as a representational concept, will be discussed shortly.

The third representational requirement is composition. Some concepts are complex, comprising a number of sub-concepts. This composition is expressed through the Ownership relation: ownedElements are a part of the owningElement. Let's look at an example.

A. *Example: Representing an Association in CRL*

Fig. 8 shows the CRL representation of the UML OrderShipmentRelation from our example. The relation is represented by an Element, indicated using the UML Stereotype notation. The names in the boxes are just labels to remind you of the concept being represented and are not part of the formal model. Each association end is represented by an ElementReference, and each of these is owned by (is a part of) the relation. Each owned reference represents the role of the indicated end-point concept with respect to the association. Associations with arbitrary cardinality can be represented in this manner.

Fig. 9 shows an instance of the UML: OrderShipmentRelation. Each object in the instance is a refinement of the corresponding object of the relation's definition.

The fifth representational requirement is the ability to indicate that one representation is a refinement of another more

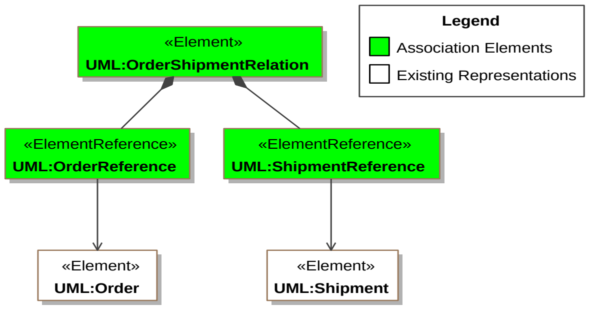

Figure 7. Representation of an Association

abstract representation. This is the CRL notion of Refinement, which subsumes the ideas of both generalization and instantiation. In Fig. 9 refinement (here represented by the UML Generalization notation) is being used to indicate traditional instantiation (e.g. the relation between UML: OrderShipmentRelation and <order 123 to shipment 57>). If it is desired to explicitly differentiate generalization and instantiation, refinements of the Refinement concept can be defined to represent the Generalization and Instantiation concepts. Instances (refinements) of these can, in turn, be used to represent the generalizations and instantiations in the model.

VI. Grounding the Model

Reducing CRL to practice introduces some challenges, beginning with the representation of refinement. While refinement could be represented in the same manner as the association shown in Fig. 8, this approach uses refinement, which makes the representation infinitely recursive. To break this recursion and ground the model, Refinement is reified as a distinct representational concept and modeled as a refinement of Element (Fig. 10). The reified Refinement has two pointers, one indicating the abstractElement and the other indicating the refinedElement.

Another issue that arises in grounding the model is the need to represent values: pointers and literals. The CRL Literal represents the literal value as a string. The CRL Pointer

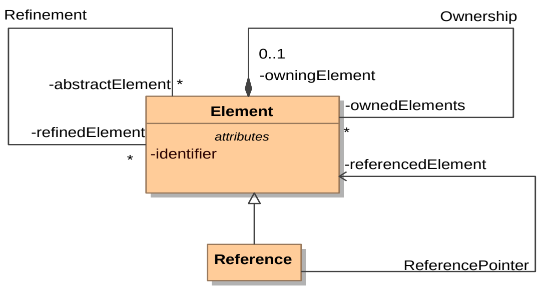

Figure 8. CRL Core Ideas

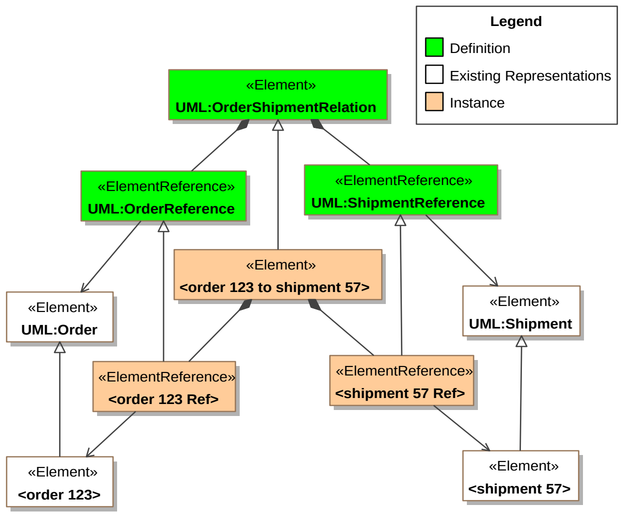

Figure 9. An Association Instance

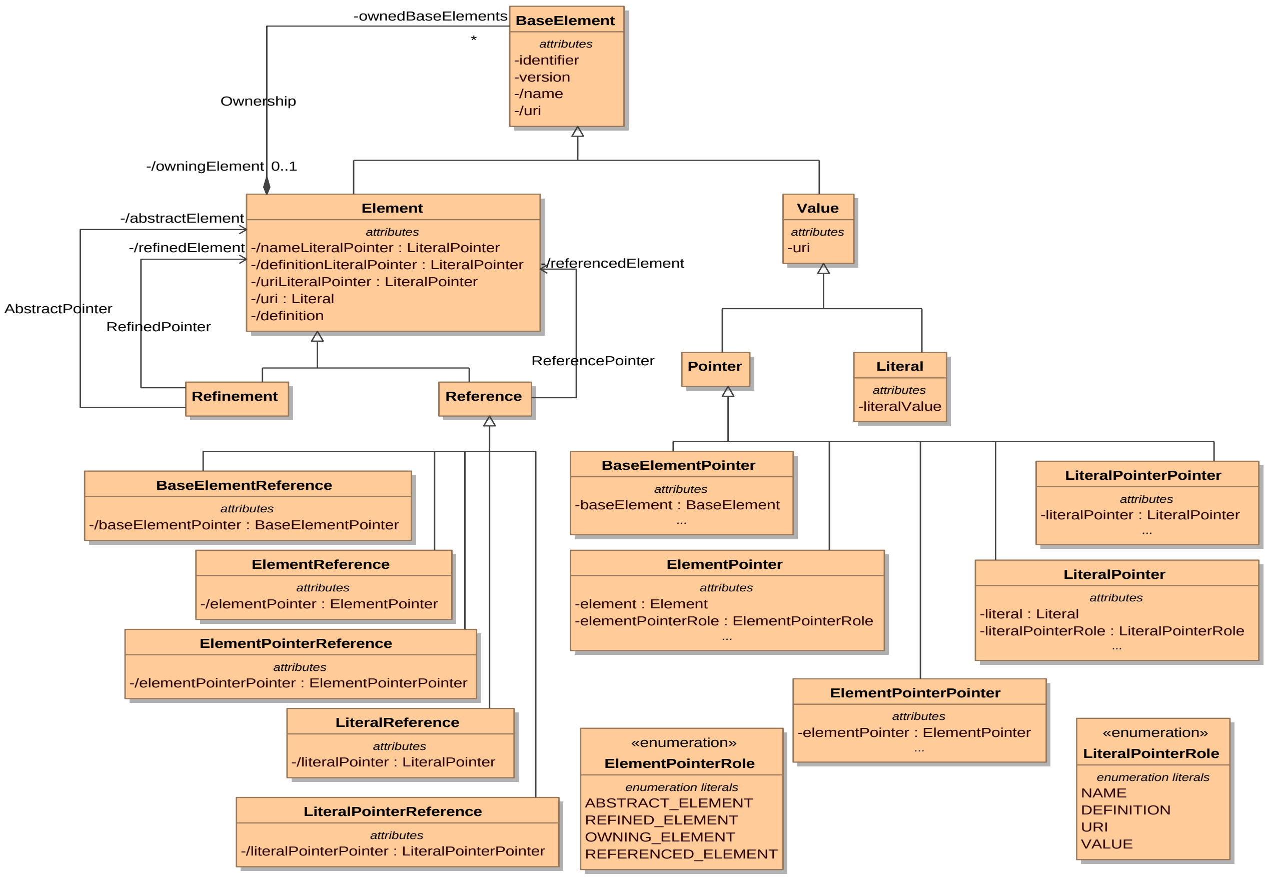

Figure 10. Grounded Model

represents the value of the pointer. The BaseElement provides a common abstraction between Element and Value. Note that Elements now own BaseElements, which means they may own values as well as concepts.

With the addition of BaseElement, Value, Pointer, and Literal arises the need to be able to reference these concepts. For this purpose, Pointer and Reference are further refined to indicate the type of the pointer and reference. Each of the refined pointers has an attribute indicating the object (this is the actual value), and each of the references has a derived attribute indicating the relevant pointer. Note that, for the purpose of mapping pointers, pointers to pointers are introduced along with corresponding references.

Giving values explicit representations and providing references to them allows the creation of associations in which both values and elements can participate. Thus CRL can represent a relationship of anything to anything. This satisfies representational requirement six.

A. Attributes Derived from Ownership Relations

A number of derived attributes are shown in Fig 10: following the UML convention, these are attributes with a slash ("/") in front of the name. For Elements and their refinements these values are derived from the presence of specific ownedElements. For example, the /literalPointer attribute of a LiteralReference is an indirect way of indicating a LiteralPointer that is an ownedBaseElement of the LiteralReference (Fig 11). There is a constraint that, for these derived attributes, only one instance of the indicated ownedElement is allowed.

The abstractElement, refinedElement, referencedElement, and owningElement attributes are similarly derived: each is indicated by the presence of an ElementPointer as an ownedBaseElement with the Pointer's elementPointerRole indicating which attribute the pointer is intended to represent.

B. Aiding Human Understanding

As mentioned previously, concepts identified only by the Element's identifier are not going to be recognizable by human readers. For this reason, the optional derived attribute /name is added to the model. The actual value of the name is derived from the structural pattern shown in Fig. 12. Providing a name for element J involves adding a LiteralPointer to J's ownedBaseElements with the literalPointerRole = "NAME".

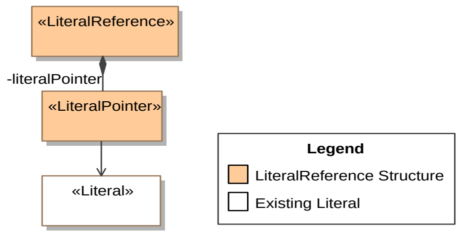

Figure 11. LiteralReference Structure

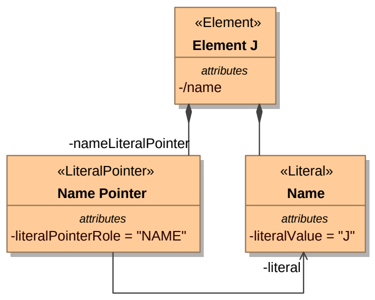

Figure 12. Structure for Providing Element Names

The literal indicated by this pointer contains the actual name. By convention, this literal would also an ownedBaseElement of J, but this is not a requirement of the model. The reason for providing this structure is to facilitate the representation of mappings involving the name and the pointer to the name (i.e. the value of the pointer).

It is important to note that this name is not intended to model the potentially complex semantic relationship between names and concepts: its provided simply as an aid to human understanding in identifying the concept being represented. In contrast with Elements, whose names are assignable, the names of Values are constants built into the model.

In similar fashion, definitions and URIs can be optionally added to the model. Definitions provide a facility for deeper human understanding of the concept being represented, while URIs provide a mechanism for identifying individual concepts without having to know the system-generated identifier.

VII. Examples

Let's look at a few simple examples.

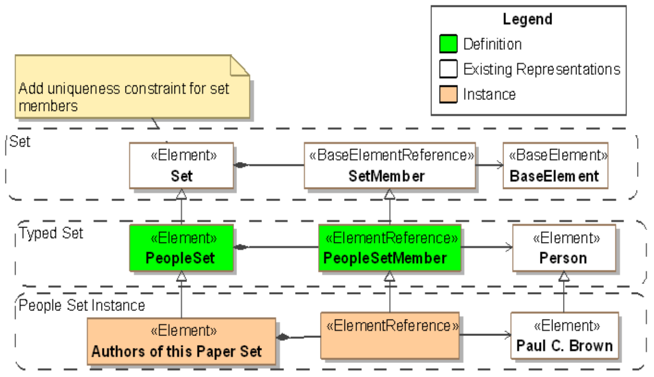

Figure 13. Representations of Sets

A. Modeling Sets

The concept of a Set (Fig. 13) is represented by an Element. Each member of the set is represented as a BaseElementReference, which can point to any BaseElement. Sets can be refined to indicate the type of element that is allowed in the set. In the figure, a PeopleSet uses PeopleSetMember references to indicate that the member is a Person. PeopleSet is a refinement of Set, PeopleSetMember is a refinement of SetMember, and Person is an Element which, by definition, is a refinement of BaseElement.

An instance of a PeopleSet might be the AuthorsOfThisPaperSet, which is a refinement of PeopleSet. The member references are refinements of PeopleSetMember, and the members must all be refinements of Person.

B. Modeling RDF Abstract Syntax

One of the most widely used semantic representation languages is RDF. Fig. 14 shows a partial representation of the

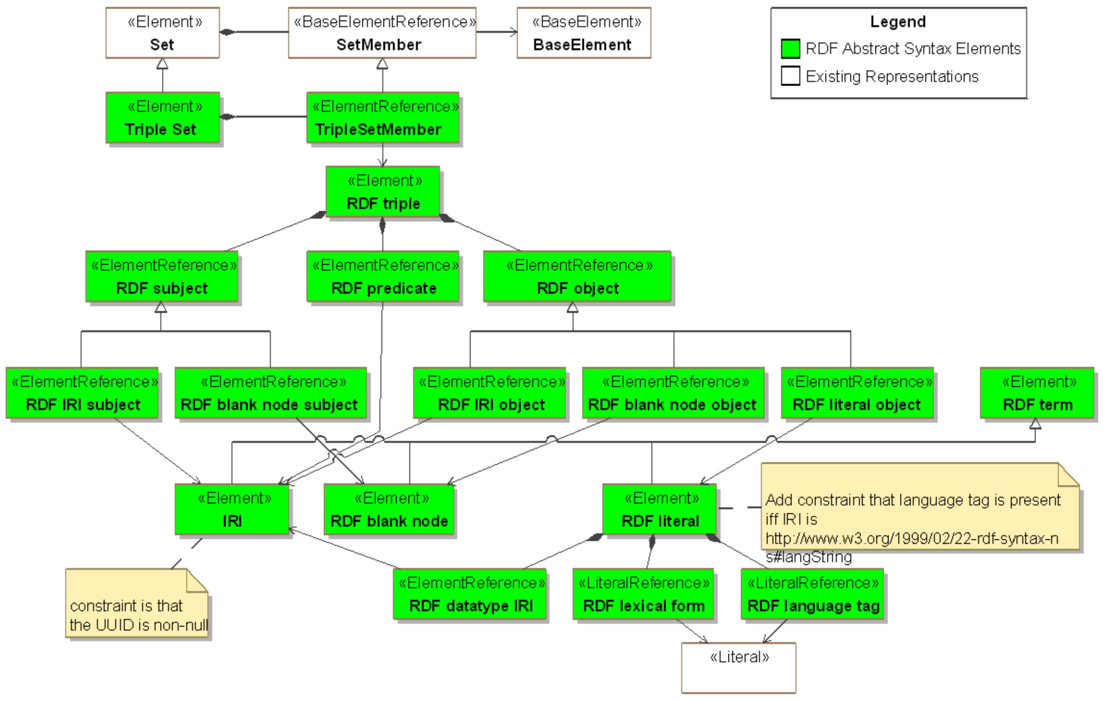

Figure 14. Partial Representation of RDF Abstract Syntax

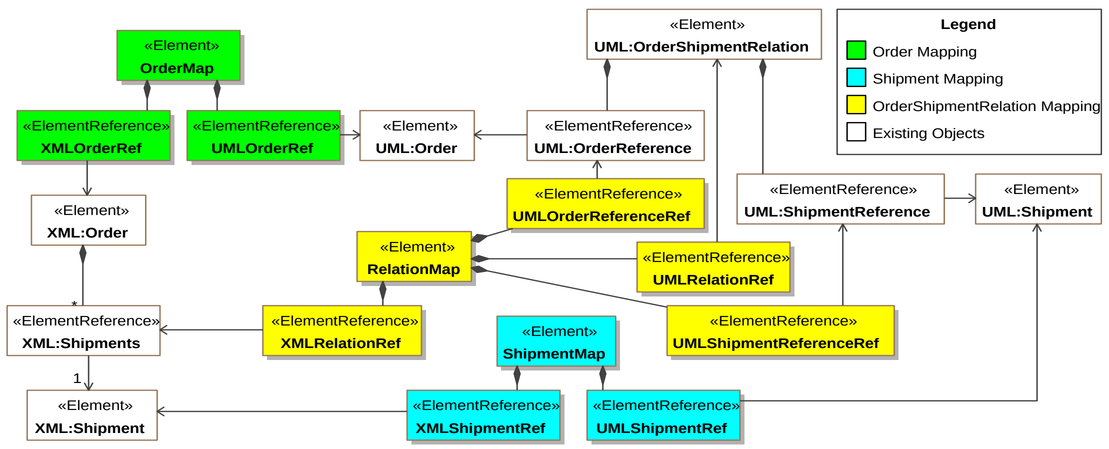
«Element» OrderMap
«ElementReference» XMLOrderRef
«ElementReference» UMLOrderRef
«Element» UML:Order
«ElementReference» UML:OrderReference
«Element» UML:OrderShipmentRelation
Legend
🟩 Order Mapping
🟦 Shipment Mapping
🟨 OrderShipmentRelation Mapping
⬜ Existing Objects
«Element» XML:Order
«ElementReference» UMLOrderReferenceRef
«ElementReference» UML:ShipmentReference
«Element» UML:Shipment
«Element» RelationMap
«ElementReference» UMLRelationRef
*
«ElementReference» XML:Shipments
«ElementReference» XMLRelationRef
«Element» ShipmentMap
«ElementReference» UMLShipmentReferenceRef
1
«Element» XML:Shipment
«ElementReference» XMLShipmentRef
«ElementReference» UMLShipmentRef

Figure 15. OrderShipmentRelation Mapping

RDF abstract syntax. Note that the RDF triple, the core construct for expressing relationships in RDF, cannot itself be the subject, object, or predicate of a triple: RDF cannot express relations between relations.

C. *Modeling a Mapping*

An everyday challenge in IT requires mapping between different data structures, often with different representations of relationships. A common requirement is to map the hierarchical structure of XML to a peer-to-peer structure. Our example of Fig. 2 shows an XML structure and a corresponding UML structure. Lets take a look at the mapping between them.

The mappings between Orders and between Shipments in the two representations is straightforward: it is a simple 1:1 corresponding UML representation (Fig. 15). The mapping of the OrderShipmentRelation is more complicated. Because of the hierarchical structure of XML, the XML representation of the relationship can be simply modeled as a collection of ElementReferences belonging to the XML:Order, each referencing an associated XML Shipment. The UML representation is more complicated. Because it is a peer-to-peer association, the UML representation has a representation of the relation (the UML:OrderShipmentRelation) which, in turn, owns two ElementReferences, one to the UML:Order and the other to the UML:Shipment. The RelationMap thus maps a single object on the XML side (the XML:Shipments) to a structure on the UML side (the UML:OrderShipmentRelation and its two owned element references). Each instance of the XML:Shipments thus corresponds to an instance of the UML:OrderShipmentRelation structure.

The mapping of Customer is a bit more involved (Fig. 16). Although the top-level correspondence is simple, the XML indication of customer type involves a reference to a CustomerType enumeration value while the UML representation utilizes different classes. Consequently, the CustomerMap utilizes two subordinate maps, the IndividualMap and the CompanyMap, that show the correspondence between the XML CustomerType values and the classes used by the UML representation.

Our last requirement is to enable the elision of details from the representation. Fig. 17 elides references and sub-structure: only the pointers beyond the sub-structure boundary are shown.

VIII. Related Work

Most work on representations is either confined to a specific domain or addresses mappings between domains. Two of the most widely used representations within domains are UML [6], which is focused on engineering designs, and RDF [9], which is focused on information representation on the web. XSLT [10] is a widely used language designed to model the transformation (mapping) of XML documents into other XML documents.

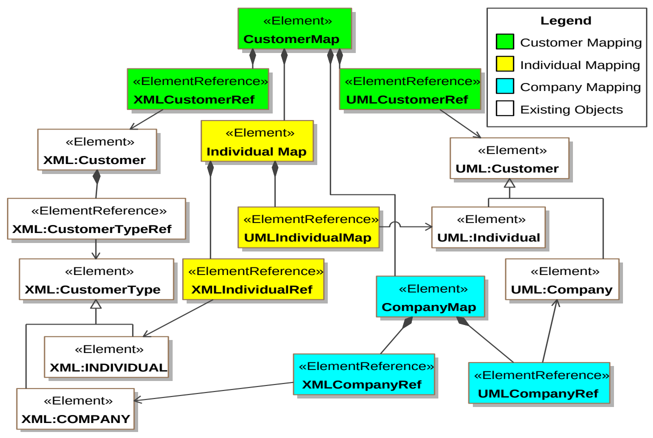

Figure 16. Customer Mapping

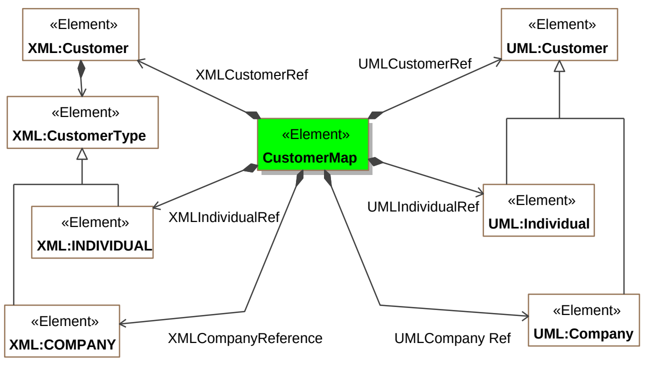

Figure 17. Eliding References and Sub-Structure

The most closely related work is that of Medhavan et al. [11] which explores mappings between domain models. The challenges they identify correlate with those discussed herein, "The first issue … is that we need to specify a mapping between models in different representation languages. The second issue is that not all concepts in one model exist directly in the other." However, their approach to heterogeneity differs. "By nature, mappings will involve multiple representation languages. Therefore, a mapping language needs to be able to represent mappings between models in different languages. An alternative approach would be to first translate all the models into a common representation language, and then specify mappings as formulas in this language. However, it is often desirable to avoid the problems associated with designing a single representation language for all models…" They choose to pursue the multi-representational method, while the CRL work explores the common representation language approach.

IX. Current Status and Planned Work

CRL was originally designed to represent data structures and the mappings between them. It is conjectured that CRL has an isomorphic representation of any data structure comprising discreet entities and discrete relationships. One current research objective is to prove or disprove this conjecture.

While developing CRL it became apparent that it can also be used to represent functions and programs. Thus, CRL can provide a uniform representation for both programs and the data upon which they operate. Furthermore, it was recognized that, by associating code fragments with individual function representations, these function representations can also be made executable.

An initial implementation of CRL was done in Java. However, the threading model and transactional semantics of Java turned out to be suboptimal for the desired execution model. A new implementation was created in the Go programming language. This implementation is named ActiveCRL to reflect its support for executable representations.

The present implementation provides the representational infrastructure with serialization/deserialization, and an optional undo-redo capability for all changes. It has the infrastructure to associate Go functions with CRL Elements representing functions and to automatically trigger the execution of those functions when changes are made to their associated representation structure. Altering the CRL structures automatically triggers reevaluation of the functions.

At present, all of the functions needed to manipulate the CRL representations have been implemented in Go and associated with corresponding CRL Elements. The stage is set for creating representations of programs that manipulate CRL representations and making them executable. The intent is to create an interactive graphical editor for CRL with all functionality except the device-specific graphic rendering being provided by executable CRL representations. ActiveCRL is very much a work in progress. The current version can be found at https://github.com/pbrown12303/activeCRL.

While CRL provides many capabilities, it is not a panacea for multi-domain computing. Its representations are fine-grained and require more computational resources than more specialized representations. On the other hand, CRL representations can be continuously extended to include new domains without modifying the core representational scheme. This means that new domains and mappings can be added at runtime rather than design time. The aim of current work is to establish the practicality of CRL for both passive representations and active executions.

References

[1] [1] F. Castanedo, "A Review of Data Fusion Techniques," The Scientific World Journal, Vols. 2013, Article ID 704504, 19 pages, 2013.

[2] [2] D. Lahat, T. Adali and C. Jutten, "Multimodal Data Fusion: An Overview of Methods, Challenges, and Prospects," Proceedings of the IEEE, vol. 103, no. 9, pp. 1449 - 1477, 2015.

[3] [3] Object Management Group, "Semantic Information Modeling for Federation (SIMF) Request for Proposal," OMG Document: ad/2011-12-10, Needham, MA, 2011.

[4] [4] W3C, "Extensible Markup Language (XML) 1.0 (Fifth Edition)," World Wide Web Consortium, 2008.

[5] [5] W3C, "W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures," World Wide Web Consortium, 2012.

[6] [6] OMG, "OMG Unified Modeling Language TM (OMG UML) Version 2.5," Object Management Group, 2015.

[7] [7] J. F. Sowa, Knowledge Representation: Logical, Philosophical, and Computational Foundations, Pacific Grove, Ca: Brooks/Cole, 2000.

[8] [8] G. Guizzardi, Ontological Foundations for Structural Conceptual Models, Enschede, The Netherlands: CTIT PhD Thesis Series, 2005.

[9] [9] W3C, "RDF 1.1 Concepts and Abstract Syntax," World Wide Web Consortium, 2014.

[10] [10] W3C, "XSL Transformations (XSLT) Version 1.0," World Wide Web Consortium, 1999.

[11] [11] J. Madhavan, P. A. Bernstein, P. Domingos and A. Y. Halevy, "Representing and reasoning about mappings between domain models," in Eighteenth national conference on Artificial intelligence, Edmonton, Alberta, Canada, 2002.