A Concept Representation Language (CRL)

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

***Abstract*— Formal languages, each with its own syntax and semantics, enable precise communications within specific domains. However, in model-based engineering and data fusion, no single language provides a comprehensive description: multiple formal languages are required, and these must be precisely related to one another. This begs the questions as to what formal language can be used to express 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 relationships 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. 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; modeling languages in model-based engineering (MBE), and conceptual models in semantics. These languages collectively represent 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). Each formal language enables precise communications within a specific domain of discourse. But comprehensive modeling of entire application domains typically requires the use of multiple formal languages and presents complex challenges in relating the languages. This paper is a start at answering a single question: can this complexity be reduced by crafting a single representation language capable of representing all possible concepts and relationships?

In MBE, representing an entire application domain and its corresponding solutions – from problem statement through requirements, design, and implementation - inevitably involves the use of multiple models and multiple formal languages. Languages such as RDF, OWL, and SBVR seek to capture natural language and business semantics [1, 2, 3]. Others such as UML and SysML focus on the behavior and structure of the design [4, 5]. Various implementation languages (e.g. Java, Go, Javascript, SQL, XML, JSON) complete the design.

Similarly, the Internet of Things (IoT) gives rise to a wide variety of information about the world. Its related data fusion problem requires correlating this information to establish a comprehensive model of the world. This requires relating models from multiple domains to formulate a complete situational representation [6, 7].

These multiple models – and their related languages - do not live in isolation: to understand a domain in its entirety, these various representations must be related to one another. "Core to MBE is the integration of descriptive/design models with the computational models" [8]. The ability to relate the concepts of one model to those of other models is essential.

Comprehensive modeling presents challenges in representing relationships. To begin with, most languages simply do not provide the capability of directly representing all possible relationships between combinations of the concepts they represent. The relationship problem becomes further complicated when the concepts to be related are expressed in different languages.

Since most formal languages are only intended to represent one aspect of a domain (e.g. a semantic model, a process model, a database schema, a data representation), a comprehensive domain model requires not only these separate representations but also a means for representing the relationships between them. If these relationships cannot be expressed within one of the existing languages, then new languages must be introduced to represent the relationships.

The abundance of transformation languages such as ATL, QVT, and XSLT is indicative of the extent of the problem [9, 10, 11]. But even these languages are insufficient: they generate one model from another rather than expressing a relationship between representations. The challenge is to relate elements of two or more existing languages and maintain these relationships as the models evolve. Even more challenging is the representation of relationships between relationships – relations between relations.

Consider the Object Modeling Group's (OMG's) Semantic Information Modeling for Federation (SIMF) effort. SIMF seeks to facilitate the transformation of data from one physical data structure to another by leveraging mappings to and from semantic models (Fig. 1) [12]. 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. The physical XML data representation is transformed into a logical XML representation (Map 1) and then into a semantic representation (Map 2). The semantic representation is then transformed into a logical JSON representation (Map 3) and

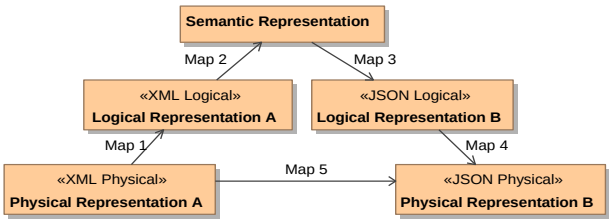

Figure 1. The SIMF Concept

then into a physical JSON representation. In practice, however, such multiple transformations are likely to be inefficient. The ultimate objective may be to collectively take the series of data structures and mappings and generate a direct transformation from the physical XML representation to the physical JSON representation (Map 5).

The SIMF effort exposes an underlying problem in common with MBE and data fusion: the formal languages used often have different representation schemes based on different sets of concepts. To work with these multiple domains, the represented concepts need to be related to one another - across languages. These relationships (Map 1 through Map 5 in Fig. 1) require a language to represent them, one that subsumes the representations of concepts from the languages being related. Sometimes the relationships themselves may need to be related to one another: in the example Map 5 is derived from Map 1 through Map 4, requiring mappings from the Map 1 through Map 4 relationships to the Map 5 relationship.

Expressing relationships generally requires adding languages, and this addition of languages adds significant complexity to constructing and maintaining comprehensive domain models. Two factors drive this complexity: the variety of 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.

This paper seeks to answer a single question: can this complexity be reduced by crafting a single language capable of representing all possible concepts and relationships? Section II presents an example illustrating some major challenges in this endeavor. Section III analyzes the reasons that most existing languages cannot express certain relationships. Section IV summarizes the representation language objectives. Section V introduces the core concepts of the Concept Representation Language (CRL), a trial balloon representation language. Section VI discusses the challenges in grounding CRL and making it practical. Section VII presents some examples, Section VIII discusses related work, and Section IX concludes with a discussion of current status and planned work.

The Concept Representation Language (CRL) presented here is intended to provide a comprehensive solution to the representation problem. CRL uniformly and consistently represents both concepts and the relationships between them. It is intended 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

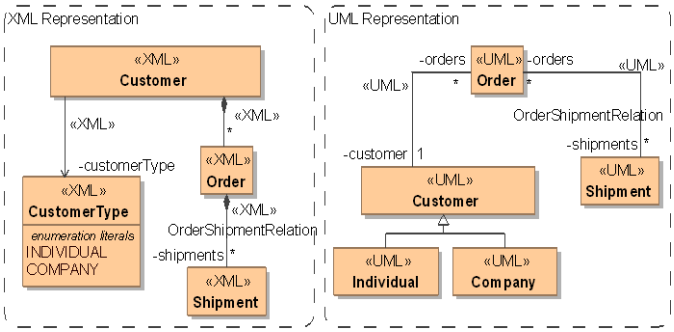

Figure 2. Differing Representations of Customers, Orders, and Shipments

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 relationship problem is common in the IT world. An order management system might generate an XML representation of a customer, orders, and related shipments (Fig. 2 left). (For simplicity, an abstracted logical model is shown.) A corresponding UML domain (semantic) model is shown on the right. There are two important differences in these representations that motivate the ensuing discussion. The first is that the XML representation on the left is hierarchical: a customer contains orders which, in turn, contain shipments [13, 14]. XML containment is a one-to-many relationship. In the UML representation on the right, the relationships are peer-to-peer. Specifically, there is a many-to-many relationship between orders and shipments in the UML model [4]. This reflects the enterprise's ability to consolidate shipping: one shipment may contain goods from multiple orders.

The second motivating difference is the representation of different kinds of customers, individuals vs. companies. 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 specific challenge is to express the relationships between the XML containment relationships and the UML peer-to-peer relationships and between the XML enumerated values (instances) and the UML classes.

III. Languages Make Ontological Commitments

There are certain language characteristics that keep many languages from being able to play this universal role. Most formal languages force the language user to make ontological commitments when representing concepts and relationships [15, 16]. These commitments codify 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 the 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. In order to relate the elements of an XML data structure to the elements of a UML model these differences in ontologies must be addressed. More generally, to express relationships between languages, their ontological categories must be related.

The following considers the problems that arise when UML is used as a tool for modeling both multiple languages and 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. The attempt identifies the nature of the challenges so that they can be avoided when crafting a less restrictive representational language. Similar representational limitations can be found in nearly all languages.

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 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 a problem arises in representing 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 a UML 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, how can the mapping between the OrderShipmentRelations be represented? The answer is, it can't - not without modifying the existing models of both languages.

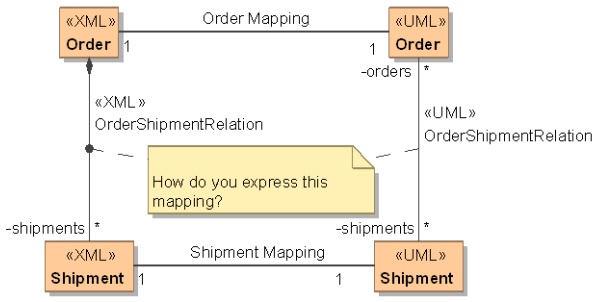

Figure 3. The Relationship Mapping Problem

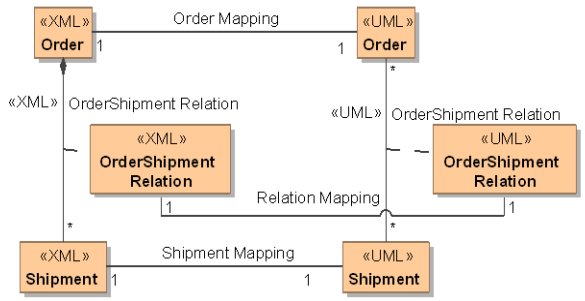

Figure 4. UML Representation of Relationship Mapping

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 (the dashed arrow 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

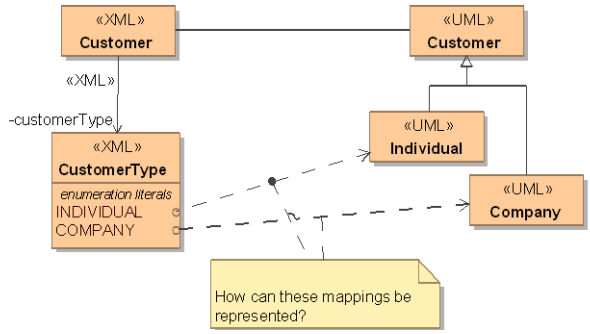

Figure 5. Mapping Instances (Enumeration Values) to Classes

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 itself provides no mechanism for denoting relationships between metalevels other than the generic dependency shown in Fig. 6.

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 to a meta-level, not absolute. In other words, they are both relationships to meta-levels.

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 instead are explicitly represented within the language.

So, what does a 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.

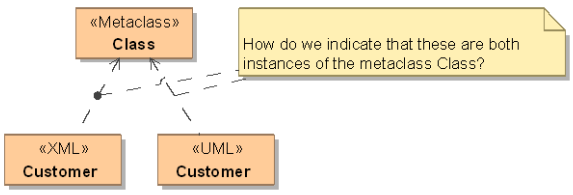

Figure 6. Elements at Different Meta-Levels

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 relationships. Since such detail can be distracting to people, one more requirement is added:

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

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 represents the idea of a reference. It has one attribute, the referencedElement, which represents the actual value of the pointer to the Element being referenced. The reference and the pointer value are distinct: the reference retains its identity even if the pointer value changes. Graphically, an arrow represents this pointer value. A Reference is a refinement of an Element: every Reference is an Element. Fig. 7 uses the UML Generalization notation to represent this 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. Here's an example.

A. Example: Representing an Association in CRL

Fig. 8 shows the CRL representation of the UML OrderShipmentRelation from Fig. 2. The relation is represented as a refinement of Element (indicated as <<Element>> using the UML Stereotype notation to conserve space). The names in the rectangles are just reminders of the concepts being represented and are not part of the formal model. Each association end is represented by a refinement of ElementReference, and each of these is owned by (is a part of) the relation. Each owned

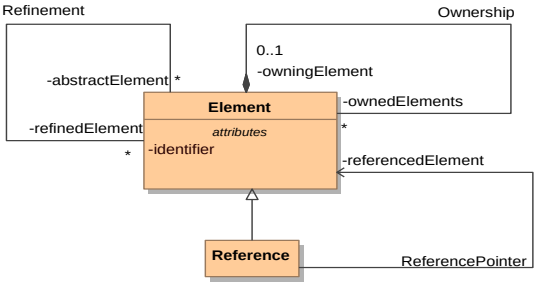

Figure 7. CRL Core Ideas

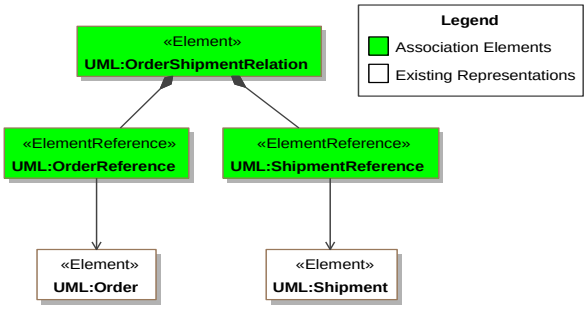

Figure 8. Representation of an Association

reference represents the role of the indicated concept with respect to the association and the pointer arrow indicates the concept being referenced. Associations with arbitrary cardinality can be represented in this manner.

Fig. 9 shows an instance of the UML: OrderShipmentRelation. The element labeled <order 123 to shipment 57> represents the instance as a refinement of the UML:OrderShipmentRelation. Its <order 123Ref> instance is a refinement of the UML:OrderReference. The referenced order, <order 123>, is a refinement (an instance) of the concept UML:Order.

Thus far, the concept of Refinement has been used without a formal definition. The fifth representational requirement is the ability to indicate that one representation is a refinement of another more abstract representation. The CRL notion of Refinement subsumes the ideas of both generalization and instantiation. In Fig. 9 refinement (represented using 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 explicit differentiation between generalization and instantiation is desired, the Refinement concept can be further refined to represent Generalization and Instantiation as concepts. Instances (refinements) of these can, in turn, be used to represent the generalizations and instantiations in the model.

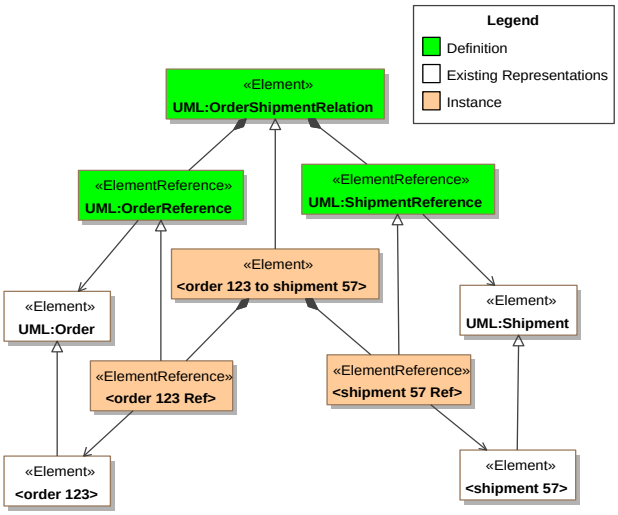

Figure 9. An Association Instance

VI. Grounding the Model

Reducing CRL to practice presents 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 example uses refinement: the association is a refinement of Element, and each reference is a refinement of ElementReference. This would make the Refinement representation infinitely recursive. To break this recursion and ground the model, Refinement is reified as a distinct concept and modeled as a refinement of Element (Fig. 10). The reified Refinement has two pointer values indicating the abstractElement and the refinedElement.

Another issue that arises in grounding the model is the need to represent – and reference - values: pointers and literals. In order for values to be referenced and participate in associations, they must have identities. The BaseElement provides identity and serves as a common abstraction between Element and Value. The CRL Literal represents the literal value as a string. The CRL Pointer represents the value of the pointer. Note that with this change, Elements now own BaseElements, which means they may own values as well as concepts.

The newly added BaseElement, Value, Pointer, and Literal concepts also need to be referenced so that they can participate in relationships. 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 pointer 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. The Fig. 5 relationship between the XML "INDIVIDUAL" value and the UML "Individual" class can now be represented. CRL can now represent a relationship of anything to anything, satisfying 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.

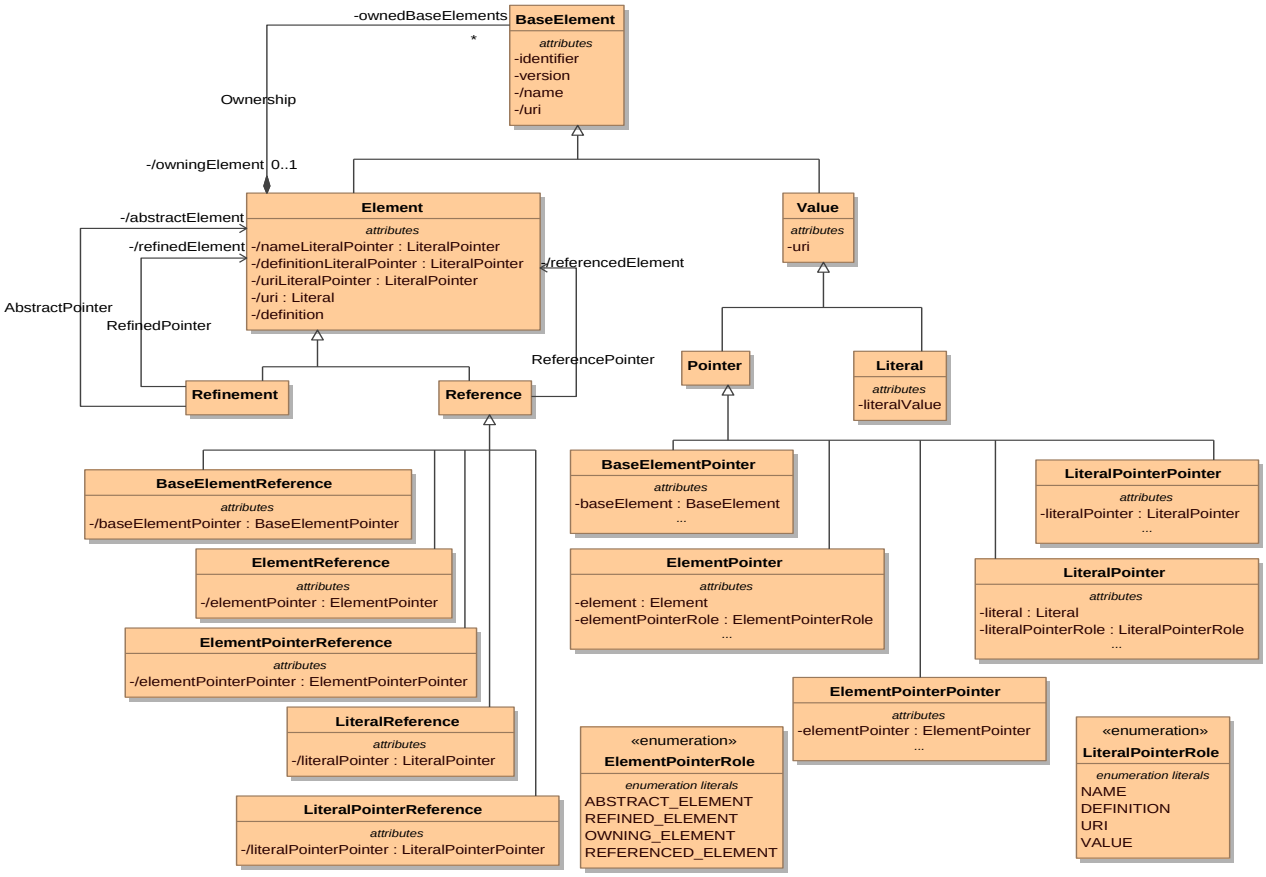

Figure 10. Grounded Model

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". 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 and Literals, whose names are assignable, the names of Pointers 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.

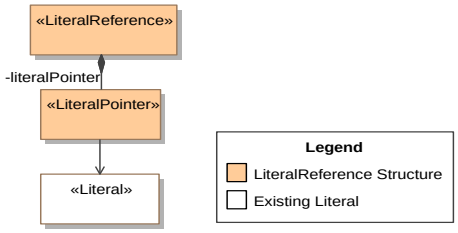

Figure 11. LiteralReference Structure

VII. Examples

A. *Modeling Sets*

The concept of a Set (Fig. 13) is represented by an Element. Each member of the set is represented as a

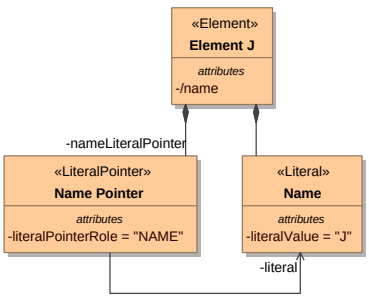

Figure 12. Structure for Providing Element Names

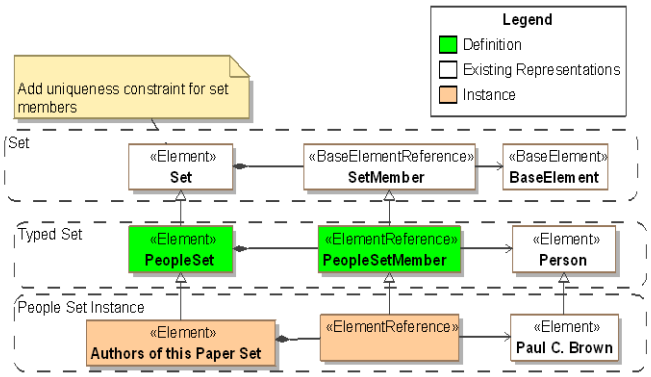

Figure 13. Representations of Sets

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 a Mapping

An everyday challenge in IT is the transformation of one data structure into another, often with different representations of relationships. A common requirement is to map the hierarchical structure of XML to a peer-to-peer structure.

The example of Fig. 2 shows an XML structure and a corresponding UML structure. The mappings between Orders and between Shipments in the two representations is straightforward: each is a simple 1:1 mapping (Fig. 14). 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

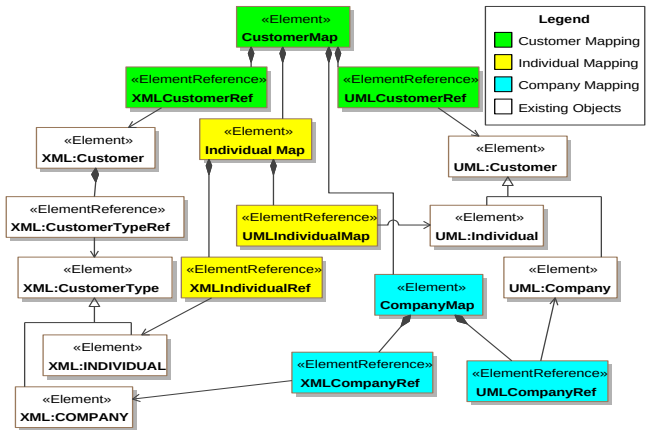

Figure 15. Customer Mapping

association, the UML representation comprises the UML:OrderShipmentRelation and two owned 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 element references). Each instance of the XML:Shipments corresponds to an instance of the UML:OrderShipmentRelation structure.

The mapping of Customer is also complex (Fig. 15). Although the top-level correspondence of Customer 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.

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 [4], which is focused on engineering designs, and RDF [1], which is focused on information representation on the web. XSLT [11] is a widely used language designed to model the transformation (mapping) of XML documents into other XML documents.

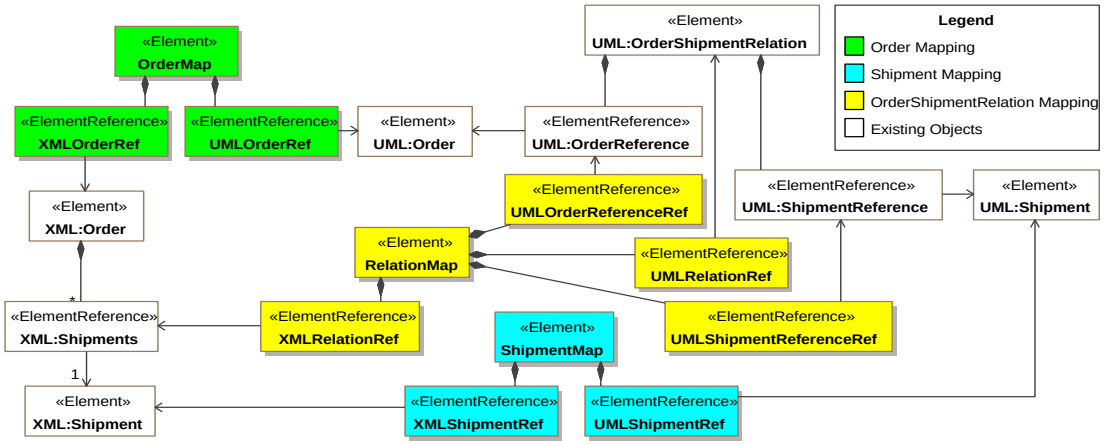

Figure 14. OrderShipmentRelation Mapping

The most closely related work is that of Medhavan et al. [17] which explores mappings between domain models. The challenges they identify relate 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 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 specialized representations. On the other hand, CRL representations can be continuously extended to include new domains without modifying the representational scheme. This means that new domains and mappings can be added at runtime. The aim of current work is to establish the practicality of CRL for both passive representations and active executions.

References

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

[2] OWL Working Group, "OWL Web Ontology Language," W3C, 11 December 2012. [Online]. Available: https://www.w3.org/OWL/. [Accessed 26 December 2017].

[3] Object Management Group, "Semantics of Business Vocabulary and BusinessRules Version 1.4," 1 May 2017. [Online]. Available: http://www.omg.org/spec/SBVR/1.4/PDF. [Accessed 26 December 2017].

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

[5] Object Management Group, "OMG Systems Modeling Language Version 1.5," 1 May 2017. [Online]. Available: http://www.omg.org/spec/SysML/1.5/PDF. [Accessed 26 December 2017].

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

[7] 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.

[8] NDIA Model Based Engineering Subcommittee, "Final Report of the Model Based Engineering (MBE) Subcommittee," National Defense Industrial Association, 2011.

[9] W. Piers, M. Barbero, F. Allialaire, T. Fortin, F. Jouault and H. Bruneliere, "ATL/Developer Guide," 31 August 2012. [Online]. Available: http://wiki.eclipse.org/ATL/Developer_Guide. [Accessed 26 December 2017].

[10] Object Management Group, "Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification Version 1.0," 1 April 2008. [Online]. Available: http://www.omg.org/spec/QVT/1.0/PDF. [Accessed 26 December 2017].

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

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

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

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

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

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

[17] 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.