A Cross-Language Type System for Information Semantics

Andruid Kerne, Yin Qu

Interface Ecology Lab, Department of Computer Science and Engineering
Texas A&M University
{andruid,yin}@ecologylab.net

1 Introduction

We are designing and prototyping a cross-language type system for information semantics. Programming languages' fundamental concepts of types and inheritance serve as the basis for cross-language representation of information, metadata semantics, and operations. The type system encompasses schemas, practical extraction rules from information published without formally exposed semantics, abstracted integration with applications, and presentation to users. From cross-language semantic entity definitions, source code is directly generated in object-oriented programming languages. Serialization of object graphs, using XML or JSON, enables exchange of structured information semantics across processes and platforms. The runtime uses the locations of documents as keys to perform the automatic instantiation of appropriately typed objects and extraction of semantics from published information sources. The representation is metadata oriented, rather than centering on reasoning. We develop alternatives to the clumsy triples of RDF, and the need for developers to build their own mappings from RDF to complex abstract data types.

2 Cross-Language Type System Foundation

Like a programming language type system, a cross-language type system deals with the representation of objects in memory. To exchange structured information or represent data structures with code generation across programming languages, a cross-language type system has to preserve the context of type translation, and deal with objects' serialized forms and deserialization process.

As the basic building blocks of a cross-language type system, scalar types are analogous to primitives. In a programming language, a primitive is essentially represented in a single unit of memory (typically from byte to double word). In a cross-language type system, a scalar type is easily represented, when serialized, as a single string, with minimal need for structural syntax delimiting and organizing fields. Thus, scalar types include all programming language primitives, plus strings, URLs, and other types, like dates and colors. The bindings between scalar types and primitives differ in different programming languages, though the serialization form needs to remain the same. For example, scalar type `Color`

is implemented by different Java classes for AWT or Android; objects with the same color value in either language should be represented identitcally for conveying correct information semantics.

In programming languages, the ability of type systems to represent *abstract data types*, including data structures and afforded operations, is valuable. This abstraction is equally important for representing and exchanging semantic objects in a cross language type system for information semantics, considering the complexity of real world semantics. In (strongly-typed) programming languages, the data structure of an abstract data type is usually defined by a list of fields, each of which describes a named memory location to hold a value of designated type, either a primitive or one defined by another abstract data type. Since fields can be designated to hold values of arbitrary types, this recursive mechanism allows creation of abstract data types with arbitrarily nested complexity.

A cross-language type system must support such abstract data type definition, and in addition, handle the serialization of abstract data type instances, enabling structured cross-language information representation and exchange. This involves delimiting data fields to preserve the structure, maintaining bindings of names to types or fields to convey the exact semantics, and so on. A cross-language type system also needs to resolve references to the same object, in order to serialize information semantics with cyclic graphs. Otherwise, the serialized form of objects could contain redundancy or even infinite recursion.

In practice, a special type family called *collection* is often used to meet the need to represent a set of values. The concept of collections is widely adopted by modern programming languages. The most common collection types are *array*, *list*, and *map*. The collection structure is widely seen in information semantics, such as a series of books or a set of employee records indexed by names. Thus it is preferable that a cross-language type system support representation and exchange of collections.

Programming languages usually define operations afforded by an abstract data type in *methods*. Although operations on information semantics are often application specific, cross-language operations, such as iterating through a collection, are needed in some cases. Ideally, a cross-language type system should support specification of cross-language operations to reduce redundancy in code that operates on information semantics.

3 Cross-Language Type System for Metadata Semantics

We argue that with S.IM.PL [3] serialization support, XML or JSON, rather than RDF, works as a primary representation for information semantics. Built upon S.IM.PL, we are designing and prototyping the *meta-metadata type system* to support cross-language representation and exchange of information semantics, and automatic source code generation for mapping information semantics to programming language objects [2] [4]. A *meta-metadata type* refers to a named schema (metadata structure and vocabulary) for information semantics. Fields can be defined inside a type, with specified names and types, to specify informa-

tion semantics data structure. Meta-information such as extraction/presentation rules can be attached, to regulate the life-cycle of information semantics.

Starting with a set of built-in types such as `document`, `image`, and `clipping`, new types can be created by extending existing types using the concept of *inheritance* in object-oriented programming. Inheritance represents the "*is-a*" (or generalization-specilization) relationship of interrelated concepts in reality; thus it is powerful for representing information semantics and reusing abstract data types. Like type systems for programming languages, meta-metadata type system allows for defining new fields or operations in subtypes. In addition, meta-information on fields can be overridden in subtypes, e.g. to attach or alter extraction/presentation rules, balancing convenience and flexbility. Notably, the type of a field can be overridden by a more specific type, as a form of generics.

Meta-metadata types are stored in XML or JSON as *wrappers* and manipulated using S.IM.PL. A wrapper is a S.IM.PL object that specifies information semantics name, schema, extraction/presentation rules, operations in the form of *semantic actions*, and selectors. Semantic actions provide flow controls like `<if>` and `<for_each>`, and bridge functions that connect information semantics to applications. Selectors prescribe the MIME type or URL pattern for sources of targeted information semantics. The system automatically generates source code of classes in target programming languages, e.g. Java, C#, and Objective-C, mapping typed information semantics to serializable *metadata objects* that can be manipuated directly in programming languages.

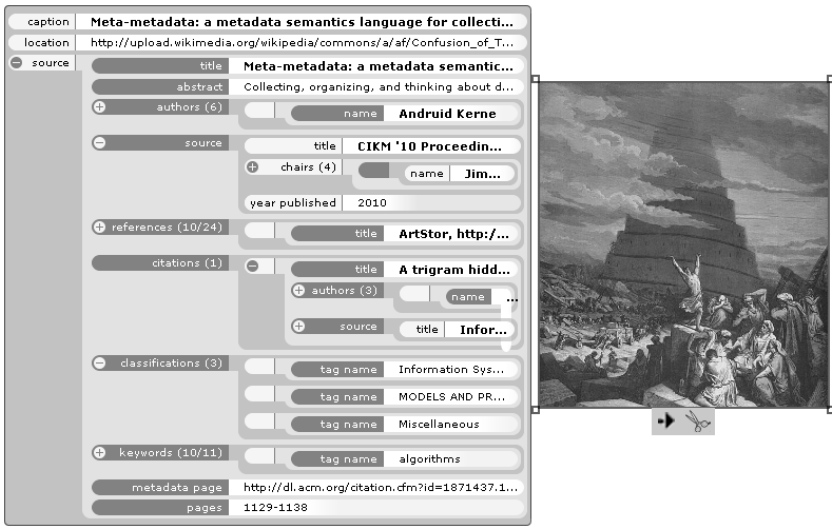

Fig. 1. Sample information semantics through meta-metadata, in *InfoComposer*.

The system supports specification and control over each phase of an information semantics life-cycle, connected by wrappers: it selects the appropriate

wrapper for an input URL, extracts information semantics from sources (e.g. web documents) into metadata objects, and performs semantic actions on them. Metadata objects can be serialized and exchanged across languages and systems through S.IM.PL without losing or changing semantics.

4 Repository of Wrappers

As modern programming languages are usually supported by a large number of "standard" classes, the meta-metadata type system comes with a reusable repository of wrappers for a set of widely used sources of information semantics [2], including Google / Bing / Wikipedia / Flickr searches, RSS feeds, digital libraries or repositories of creative works (including Google Patent / Book Search, ACM Portal, Informa World, CiteSeerX, etc.), products (Amazon), artworks (MoMA) and others. Developers can easily use information semantics from these sources directly in their applications, or derive new wrappers from them while maintaining interoperability [5]. The repository allows reuse of schemas, extraction rules, and other aspects regarding information semantics across contexts.

5 Relationship to Prior Semantic Web Technologies

We know that many researchers are deeply committed to existing semantic web standards like RDF and OWL [1]. While we do not intend to offend anyone, in accord with this venue, we outrageously assert that completely different structural approaches to semantic web representation may prove valuable. As homogeneous approaches to culture have been rejected by cultural studies, in the name of multi-vocality, we outrageously assert that the idea of "The Semantic Web" is a bad one. Instead, we posit co-existence of many semantic web representation substrates. While this will require developing interoperability, it will also enable rich intellectual diversity to flourish. Just as triples are not the preferred representation of abstract data types in programs, so it could prove beneficial to use an ADT representation for structured information across languages.

References

1. I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. From shiq and rdf to owl: the making of a web ontology language. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(1):7 – 26, 2003.
2. Interface Ecology Lab. Meta-Metadata Guide. `http://ecologylab.net/research/simplGuide/metaMetadata/index.html`.
3. Interface Ecology Lab. S.IM.PL Guide. `http://ecologylab.net/research/simplGuide/index.html`.
4. A. Kerne et al. Meta-metadata: a metadata semantics language for collection representation applications. In *Proc. of CIKM*, 2010.
5. Y. Qu et al. Interoperable metadata semantics with meta-metadata: A use case integrating search engines. In *Proc. of DocEng*, 2011.