Meta: A Universal Meta-Language for Augmenting and Unifying Language Families, featuring Meta(Oopl) for Object-Oriented Programming Languages

Wade Holst
Department of Computer Science
University of Western Ontario
London ON Canada
wade@csd.uwo.ca

ABSTRACT

Meta: a collection of meta-languages that augment/unify language-families, an advanced programming environment, a language interoperability framework, an infrastructure for reflection over source documents, and a generalization of XML. XML provides an extensible mechanism for defining syntax, while Meta provides an extensible mechanism for defining syntax and semantics.

Categories and Subject Descriptors

D.3.3 [**Programming Languages**]: Language Constructs and Features

General Terms

Languages, Documentation

Keywords

language design, meta-languages, meta-programming, language interoperability, object-oriented

1. AN INTRODUCTION TO META

This extended abstract presents an overview of $\mathcal{M}eta$ and the most important $\mathcal{M}eta$-language defined by $\mathcal{M}eta$, denoted $\mathcal{M}eta$(Oopl). $\mathcal{M}eta$ is an extensible meta-language whose overall purpose is the augmentation and unification of *arbitrary families of languages*. $\mathcal{M}eta$(Oopl) is a $\mathcal{M}eta$-language whose overall purpose is the augmentation and unification of object-oriented programming languages.

There are two orthogonal ways in which $\mathcal{M}eta$ can be characterized. First, $\mathcal{M}eta$ is an environment for augmenting and unifying arbitrary existing languages. Second, $\mathcal{M}eta$ extends the concepts present in XML, leading to the potential to collapse arbitrary languages into a common extensible syntax (in a manner analogous to how XML has collapsed markup languages into a single common extensible syntax).

Under the first characterization, $\mathcal{M}eta$ is an environment for augmenting and unifying pre-existing languages. More specifically, $\mathcal{M}eta$ is a set of *$\mathcal{M}eta$-languages*, and each $\mathcal{M}eta$-language is a family of augmented and unified versions of one or more *base languages*. A base language is simply an existing language that $\mathcal{M}eta$ provides support for, and can be a general purpose programming language like Java, a type-setting language like TeX, or any other structured set of syntactic elements with associated semantics. $\mathcal{M}eta$ organizes related base languages into *language families*, then introduces a $\mathcal{M}eta$-language for each family of languages. For example, $\mathcal{M}eta$(Oopl) is a $\mathcal{M}eta$-language for object-oriented programming languages like C++, Java and Perl, while $\mathcal{M}eta$(Doc) is a $\mathcal{M}eta$-language for typesetting languages like LaTex, HTML, etc.

Under the second characterization, $\mathcal{M}eta$ is a generalization of XML. $\mathcal{M}eta$ aims to do for arbitrary languages what XML has done for markup languages. XML is an eXtensible Markup Language; a universal syntax that can be used to define an infinite number of markup languages (each new language defined by an XML *schema*). $\mathcal{M}eta$ is also an extensible language, and introduces a universal syntax that can be used to define an infinite number of $\mathcal{M}eta$-languages (a $\mathcal{M}eta$-language is a language that uses $\mathcal{M}eta$ syntax). However, there are crucially important differences between $\mathcal{M}eta$ and XML. First, XML deals only with syntax (there is no intrinsic semantics associated with elements in an XML document), while $\mathcal{M}eta$ allows the language implementor to define syntax, semantics for that syntax, and an implementation of that semantics. Second, $\mathcal{M}eta$ uses a significantly more concise (but equally powerful) syntax compared to XML. As well, special syntax configuration facilities provided by $\mathcal{M}eta$ allow the content-to-noise ratio of $\mathcal{M}eta$ programs to be increased even further. Each $\mathcal{M}eta$-language (for example, $\mathcal{M}eta$(Oopl)), is defined in a manner similar to an XML schema. Each document written using the syntax of a particular $\mathcal{M}eta$-language is parsed relative to the schema for that $\mathcal{M}eta$-language. In the same way that XML allows new schemas to be defined (which use XML syntax), $\mathcal{M}eta$ allows new $\mathcal{M}eta$-languages to be defined (using $\mathcal{M}eta$ syntax). Unlike XML schemas, $\mathcal{M}eta$-languages can participate in inheritance-like relationships. For example, a user can define a new $\mathcal{M}eta$-language that "inherits" from $\mathcal{M}eta$(Oopl) but provides additional syntax and semantics.

Each $\mathcal{M}eta$-language increase the expressive power of all its base languages, by guaranteeing support for various *mandates* (features) regardless of whether the base languages provide direct support. Although individual $\mathcal{M}eta$-languages introduce $\mathcal{M}eta$-language-specific mandates, every $\mathcal{M}eta$-language is guaranteed to provide a core set of mandates:

1) *Language Interoperability*: Each $\mathcal{M}eta$-language provides various kinds interoperability among the base lan-

OOPSLA'05, October 16–20, 2005, San Diego, California, USA.
ACM 1-59593-193-7/05/0010.

guages it augments, including the incremental migration of a document written in one base language into another base language, the ability to write a document containing multiple base language implementations at the same time, and the ability to write a document using a base-language-neutral syntax that can be converted into any base language.

2) *Reflection*: Each $\mathcal{M}eta$-language provides support to allow programs to ask questions about (via *introspection*) and modify (via *intercession*) the syntax, semantics, and implementation of documents and the languages those documents are written in, at various phases at and between compile-time and run-time. Every $\mathcal{M}eta$-language defines a *meta-object-protocol* (MOP) to provide this reflection.

3) *Readability and Writability*: Each $\mathcal{M}eta$-language provides users with significant control over the syntax of $\mathcal{M}eta$ documents (via implicit syntax to capture common cases). However, since implicit syntax can also decrease readability, $\mathcal{M}eta$ provides canonicalization facilities to automatically make implicit syntax explicit. $\mathcal{M}eta$(Oopl) source code is 30-50% smaller than corresponding base-language source code. $\mathcal{M}eta$ allows one to say more with less.

2. META(OOPL)

Although $\mathcal{M}eta$ provides a framework for defining arbitrary new $\mathcal{M}eta$-languages, the augmentation and unification of object-oriented programming languages, refereed to as $\mathcal{M}eta$(Oopl), is both a primary reason for the existence of $\mathcal{M}eta$, and the means by which the generalized $\mathcal{M}eta$ compiler and all other technologies related to $\mathcal{M}eta$ are implemented. $\mathcal{M}eta$(Oopl) defines a family of related programming languages that augment and unify existing object-oriented programming *base languages*. For example, $\mathcal{M}eta$(Oopl)<C++> and $\mathcal{M}eta$(Oopl)<Java> (commonly abbreviated as $\mathcal{M}eta$<C++> and $\mathcal{M}eta$<Java>) extend the base languages C++ and Java, respectively.

The benefits provided by $\mathcal{M}eta$ and the various $\mathcal{M}eta$-languages are showcased by summarizing the *mandates* that $\mathcal{M}eta$(Oopl) is guaranteed to provide to its base languages, regardless of whether the base languages have direct support. The list of mandates includes: standardized syntax for classes/methods/fields, numerous forms of language interoperability (some provided by the universal syntax and language lattice common to all $\mathcal{M}eta$ languages, some specific to $\mathcal{M}eta$(Oopl)), advanced reflection via a $\mathcal{M}eta$-Object Protocol, an extended $\mathcal{M}eta$ Type System, the $\mathcal{M}eta$ Library (a collection of classes defined in all base languages and having exactly the same interface and semantics in all such languages), support for aspects, components, garbage collection, object serialization, augmented class-based enumerated types, multi-method dispatch, various inheritance extensions, instance/static/class-level methods and fields, various syntactic extensions, operator overloading, default arguments, automated unit testing harness, pre/post conditions, automated document generation, 2D and 3D visualization of programs, source code configuration and canonicalization, dependency management, version control, reverse translation, and bug-reporting facilities.

3. THE META LANGUAGE LATTICE

For each $\mathcal{M}eta$-language defined by $\mathcal{M}eta$, there is an associated *language lattice*, consisting of a hierarchically related collection of languages that provide progressively more expressive power and progressively less reliance on base-language syntax. At the bottom of this language lattice are

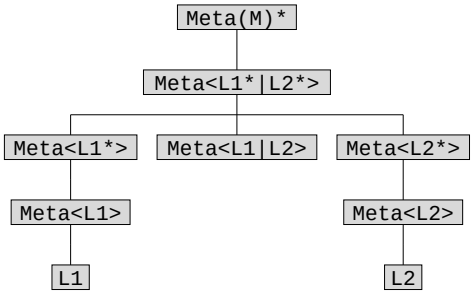

Figure 1: The $\mathcal{M}eta$(M) Language Lattice

the base languages (in $\mathcal{M}eta$(Oopl), languages like C++, Java and Perl). Above these languages are the languages $\mathcal{M}eta$<C++>, $\mathcal{M}eta$<Java>, etc. The syntax of these languages is a combination of special $\mathcal{M}eta$(Oopl) syntax for "high-level" syntactic constructs (i.e. classes, methods, fields, aspects, components, etc.), and base-language syntax for "low-level" constructs. What is considered high-level and low-level syntax is established by each $\mathcal{M}eta$-language; for $\mathcal{M}eta$(Oopl), statement-level syntax is considered low-level, and everything else is high-level.

Above $\mathcal{M}eta$<C++>, $\mathcal{M}eta$<Java>, etc., there are two directions by which the language lattice can be generalized. Along one branch, languages are introduced that reduce the reliance on base-language syntax, by providing $\mathcal{M}eta$ syntax even for "low-level" syntax. Along the other branch, languages are introduced that directly support language interoperability by allowing implementations in multiple base-languages within the same source document. For $\mathcal{M}eta$(Oopl), the languages $\mathcal{M}eta$<C++*>, $\mathcal{M}eta$<Java*>, etc. are examples of the former generalization, while languages denoted $\mathcal{M}eta$<C++|Java>, $\mathcal{M}eta$<C++|Java|Perl>, etc. are examples of the latter generalization. Above these languages are languages that allow both mixed syntax and mixed languages, like $\mathcal{M}eta$<C++*|Java*|Perl*>. At the root of the language lattice is the special *closure language*, $\mathcal{M}eta$(Oopl)*, which has no base-language syntax. Documents written in $\mathcal{M}eta$(Oopl)* can be compiled into any base-language supported by $\mathcal{M}eta$(Oopl).

4. CONCLUSIONS

To summarize, $\mathcal{M}eta$ is an extensible language for defining a potentially infinite number of $\mathcal{M}eta$-languages. $\mathcal{M}eta$ defines and implements a core set of $\mathcal{M}eta$-languages needed to implement itself, including $\mathcal{M}eta$(Oopl), a family of augmented and unified object-oriented programming languages. Each $\mathcal{M}eta$-language (both those defined by $\mathcal{M}eta$, and new ones defined by users) is responsible for augmenting and unifying a family of one or more base languages, providing facilities for increasing the readability and writability of documents written in those languages, providing support for interoperability/interaction between documents written in different base languages, providing an infrastructure for reflection on the syntax and semantics of documents and languages, and possibly doing much more at the $\mathcal{M}eta$-language level, as is the case of $\mathcal{M}eta$(Oopl), which guarantees the existence of many powerful and highly advantageous features regardless of whether its base languages provide support for those features.

5. REFERENCE

[1] Wade Holst. [URL] Univ. of Western Ontario, 2005. http://meta.csd.uwo.ca/Meta.