MetaL

XML based Meta-Programming Language written in PHP

PHP Conference 2001

Frankfurt - Germany

November 5-7, 2001

Manuel Lemos © 2001 – mlemos@acm.org

October 28, 2001

1. History

This part tells the history of the creation and development of MetaL.

1.1 The Web ERP project

- Late 1998: The Web ERP project was started as a proposal of some investors.
- The goal was to develop a Web based business management suite of programs: accounting, billing, stocks, etc..
- The project was taking too long with only one person learning the business processes, planning and developing.
- Finding available qualified Web developers was hard and when you found them they were expensive to hire due to the Internet hype. /

1.2 The birth of MetaL

- Middle 1999: MetaL was born with special goals meant to defeat these difficulties:
  - Be simple enough to not require expensive qualified developers to write applications with it.
  - Allow the design of business domain specific languages to help writing applications faster.
- Late 1999: The original investors back out due to lack of faith on the conclusion in useful time of any part of the project that had market value. /

1.3 The Integral project

- Early 2000: A previous business partner proposes to invest on restarting the Web ERP project.
- The project was rebaptized as *Integral* - a generic integrated system for modular Web applications.
- A few modules were developed:
  - Base installation and setup of modules
  - User access management
  - Companies records management
- Late 2000: The Integral project was suspended due to an irrefusable job proposal. /

1.4 Today

- Middle 2001: MetaL was opened to closed circle of trusted developers for evaluation in preparation for release as Open Source.
- MetaL was presented publicly for the first time in the O'Reilly Open Source Conference (`http://conferences.oreillynet.com/cs/os2001/view/e_sess/1874`) in San Diego.
- MetaL and Integral are expected to be released as Open Source projects until the end of 2001.
- The development of MetaL and the Integral project are being carried out as my free time permits. //

2. Motivation

This part details the reasons that motivated the development of a Meta-Programming Language.

2.1. The ideal language

- Question: What is the best programming language?
- Answer: It depends on your purpose.
- Question: What if there was a language that is suitable for all purposes?
- Answer: There is no such ideal language.
- Question: What if there was a language that could be translated into other languages?
- Answer: That is the main goal of *MetaL*: a meta-programming language. /

2.2. Meta-Programming

- Developing a new language from scratch would be of great risk and would take much time.
- Giving up on existing languages would mean giving up on their maturity and fitness to solve real world problems.
- Developing a meta-programming compiler would represent a compromise between developing a new language without giving up on existing languages. /

2.3. Future opportunities

The adoption of MetaL as the language for software development may present opportunities that developers may benefit in the future.

- *Retargettable Programming*

  If I can generate a program in different programming languages, I may as well generate its code in a suitable manner to run optimally on different platforms, even those that do not exist today.

- *Future-proof Software*

  The languages that are most appropriate for developing applications today, may not be the best choice for developing future applications.
  Whatever are the languages of choice in the future, MetaL programs and components may not need to be

adapted to be generated in the languages of the future. /

2.4. Language of powerful commands

Programs written in a language with powerful commands need less code to write.

- Less development time
  -> Less time to market
- Less bugs
  -> Higher code quality
- Greater code maintainability
  -> Less time to develop new features
  -> Greater user satisfation /

2.5. Domain specific languages

- A programming tool that is appealing to less qualified developers, may also be appealing to experts on areas that are important for a software product.
- For instance:
  - An *economist* could define *business rules* to be implemented by a software product.
  - A *translator* could *localize* a software product.
- These experts could participate directly in the development of a software product, especially if they do not have to learn traditional programming languages.
- *Domain specific languages* could be used by these professionals.
- *Domain experts* that are not programmers, are easier to find and less expensive to hire. /

2.6. Meta-Meta-Programming

- If a meta-programming language can be used to generate programs in any target language, it may as well be used to generate programs in the meta-language itself.
- This means that there may be very high level commands in a meta-language that can be interpreted to generate source code with only lower level meta-language commands.
- Translation of higher level commands in lower level ones may be done in one or more compiler passes. In a final pass, the meta-programming compiler generates code in a given target language.
- The method of developing programs with meta-language commands that are translated into lower level commands is called *Meta-Meta-Programming* or just *Meta-N-Programming*.
- Very high level domain specific languages can be developed

with more flexibility with *Meta-Meta-Programming.* //

3. Introduction to MetaL

This part explains how MetaL works.

3.1. XML as programming source code

- Developing a compiler is not a trivial programming matter.
- It requires knowledge on:
  - Parsing source code with a given syntax.
  - Generating program output in a target executable format.
- Using XML as source code greatly simplifies the problem.
  - XML requires a small set of strict syntax rules.
    -> XML parsers are much simpler to build.
  - Generic XML parsers are widely available. /

3.2. Programming as simple as cooking

- People learn faster abstract concepts when these are presented using analogies with real world concepts.
- Not everybody is able to cook or only eats cooked food, but...
- Cooking is a real world concept that everybody understands how it works.
  1. First, you need to find and prepare the *ingredients*.
  2. Then, you follow the *recipe steps*. /

3.3. MetaL ingredients

- The ingredients are the commands of the language that you want to use in your programs.
- Each command is associated with a XML tag.
- MetaL commands are defined in *input* files. /

3.4. MetaL recipe steps

- The recipe steps are just a sequence of XML tags that define your program commands.
- MetaL commands are defined in *output* files. /

3.5. Compiling MetaL programs

1. To build a MetaL program, the compiler takes an *input* file and an *output* file.
2. The compiler examines the input file to figure the language commands that you want to use.
3. Input files also define which compiler modules implement each command.
4. The compiler traverses the output file and invokes the commands associated to each XML tag by calling the respective compiler module. /

3.6. Generating target source code

- Each compiler module may execute any type of action to implement a MetaL command.
- Actions may be immediate, like creating a file, or may be generating some output like the lines of code of a program in a given target language.
- A compiler module may be able to generate code for the same MetaL command in multiple target languages.
- The compiler *bindings* define which module implements the translation of commands to a given target language.
- The definition of the compiler bindings is done in a separate XML file with a specific structure. //

4. State of development MetaL

This part describes the current state of MetaL.

4.1. Current implementation

- The current implementation of MetaL compiler engine and the available modules were developed with about 20.000 lines of *PHP* (`http://www.php.net/`) code.
- PHP is a scripting language that was originally meant for Web programming.
- It may seem odd to the use PHP for a heavy task such as compiling source files.
- Today PHP is not such a slow language, thanks to the *Zend* (`http://www.zend.com/`) compiler and optimizer.
- The MetaL compiler engine also makes extensive use of caching of previously parsed XML files. /

4.2. Available modules - Project building

- All compiler parts and modules are implemented as OOP classes.
- The compiler modules were not meant just to generate program's source code.
- Some modules were developed to assist all steps of project building:
  - `compiler` - Compiler core engine and base services
  - `build` - *Makefile*-like build process control
  - `documentation` - Generation of documentation from general a device independent document definition format
  - `distribution` - Creation of distribution packages from programs built with MetaL /

4.3. Available modules - Runtime control

- The most important meta-programming modules are meant for controlling the execution flow and data manipulation.
  - `expressions` - Data type manipulation functions and operators
  - `flow` - Execution logic flow control constructs
  - `script` - Generation of output source code as scripts.
  - `program` - Access to context values passed to the programs
  - `class` - Creation, access and documentation of OOP classes /

4.4. Available modules - Web programming

- Some modules were developed to assist Web programming in specific:
  - `cgi` - Access to data values interchanged with Web scripts via CGI
  - `template` - Generation of output from text based template files
  - `menu`, `image`, `link`, `form`, `email` - Interface with existing native PHP classes
  - `options` - Access to application wide global option values
  - `locale` - Localization of application texts and data /

4.5 Available modules - System access

- The remaining modules are meant for accessing to system resources:
  - `file` - File and directory access functions
  - `database` - DBMS independent SQL database access
  - `time` - Date and time access manipulation functions
  - `xml` - XML parsing functions using *Expat* (`http://www.jclark.com/xml/expat.html`)
  - `crypt` - Cryptographic functions
  - `system` - External program invocation functions /

4.6. Examples of existing applications

Integral (http://www.integral.UpperDesign.com/)Modular integrated systemof Web based applications.

Multi-page forms class (http://phpclasses.UpperDesign.com/browse.html/package/108)

Calendar generation class (http://phpclasses.UpperDesign.com/browse.html/package/121)

Web based *ezmlm* mailing listcreation and configuration tool (http://phpclasses.UpperDesign.com/browse.html/package/177)

Table wrapper base class (http://phpclasses.UpperDesign.com/browse.html/package/120)

Query result table display class (http://phpclasses.UpperDesign.com/browse.html/package/130)

Date and time utility class (http://phpclasses.UpperDesign.com/browse.html/package/230)

XML Writer class (http://phpclasses.UpperDesign.com/browse.html/package/250)

File cache class (http://phpclasses.UpperDesign.com/browse.html/package/313)

Database access class (http://phpclasses.UpperDesign.com/browse.html/package/231)

SOAP server class (http://phpclasses.UpperDesign.com/browse.html/package/251)

*I-Know*Multi-word key expression indexingand searching technologyfor building knowledge bases //

5. Future of MetaL

This part addresses near future initiatives and opportunities of development with MetaL.

5.1. Planned work

- The development of MetaL depends on my current needs.
- The original motivation was to develop *Integral*: a modular Web based integrated system.
- Development of *Integral* is suspended due to the lack of time after my day job.
- Work on MetaL is often motivated by needs of my current job. /

5.2. Planned features and modules

1. Complete bindings for other Web programming languages like Perl and Java.
2. Module for building GUI applications.
3. Module for generating documentation in other formats besides HTML.
4. Bindings for compiled languages like C: *Heavy-MetaL project.* /

5.3. Planned Meta-Meta-Programming modules

1. Module to assist the development of new compiler modules in pure *MetaL*.
2. Database Procedure Language
3. Data Cliping Language /

5.4. Going Open Source

- A project like MetaL has too much potential to be maintained just by one developer.
- MetaL was not developed to take commercial advantage of its potential.
- It makes complete sense to open the source of MetaL to other developers interested to cooperate.
- The proof of concept of MetaL is to generate an application in two or more languages and demonstrate that all implementations work indifferently.
- At the time of this writing, that proof of concept is almost done.
- After that, MetaL will be turned into a full fledged *Open Source* (`http://www.opensource.org/`) project.

- The Open Source license will be *Apache* (`http://www.opensource.org/licenses/apachepl.html`)/*BSD* (`http://www.opensource.org/licenses/bsd-license.html`) like. /

5.5. Opportunities for other developers

- Carry on the development of Integral and its modules.
- Rewrite the compiler and its modules in lower level language like C/C++ to optimize compilation speed.
- Develop MetaL IDE - Integrated Development Environment.
- Develop bindings for other languages.
- Develop programs to reverse-engineer source code written in traditional languages to translate them to MetaL.
- Participate in the process of standardization and documentation of MetaL. //

6. Conclusion

This part presents final remarks and a pointer to obtain further information.

6.1. Final remarks

- MetaL exists now and has been used successfully in real world projects.
- It is not intended to replace the traditional programming languages.
- It is meant enhance the software development processes and improve the productivity of the developers.
- MetaL has indeed a lot of potential.
- It is a job for much more than one developer to take full advantage of MetaL potential. /

6.2. For more information...

More information on MetaL is available at: www.meta-language.net (`http://www.meta-language.net/`) //

The End

Thank you!

Manuel Lemos

mlemos@acm.org

http://www.ManuelLemos.net/