# Using The Meta-Environment for Maintenance and Renovation

M.G.J. van den Brand
Department of Mathematics and Computer Science, Technical University of Eindhoven
Eindhoven, The Netherlands

M. Bruntink, G.R. Economopoulos, H.A. de Jong, P. Klint, T. Kooiker, T. van der Storm and J.J. Vinju
Centrum voor Wiskunde en Informatica (CWI)
Amsterdam, The Netherlands
www.meta-environment.org

## Abstract

*The Meta-Environment is a flexible framework for language development, source code analysis and source code transformation. We highlight new features and demonstrate how the system supports key functionalities for software evolution: fact extraction, software analysis, visualization, and software transformation.*

## 1. Introduction

The Meta-Environment is an open framework for language development, source code analysis and source code transformation. It consists of syntax analysis tools, semantic analysis and transformation tools, and an interactive development environment (see Figures 1 and 2). It is supported by a growing open source community, and can easily be modified or extended with third party components.

The Meta-Environment is a generalization of the ASF+SDF Meta-Environment that has been successfully used in many analysis, transformation and renovation projects. In the software evolution domain, The Meta-Environment has been used for applications such as:

- Parsing (new and old) programming languages, for further processing the parse trees.
- Analysis of source code (fact extraction, type analysis, and documentation generation).
- Transformation, refactoring, and code generation.

## 2. Features of The Meta-Environment

The Meta-Environment has already pioneered several innovations in generic language technology:

- Modular grammar definitions—a consequence of our generalised parsers.

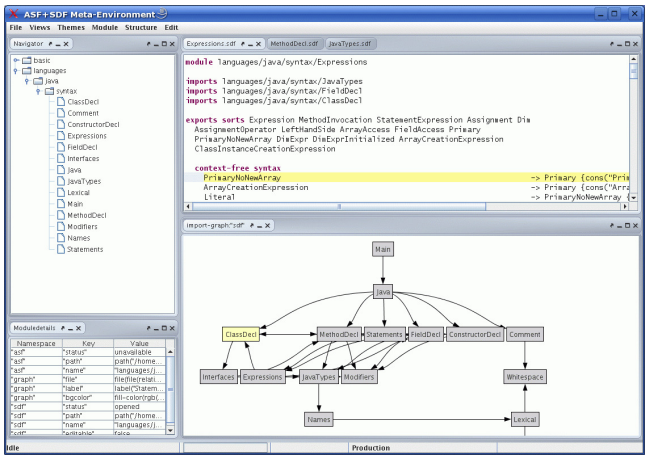


**Figure 1. User interface**

- Declarative disambiguation filters used to resolve many common ambiguities in programming languages.
- Conditional term rewriting used to perform software transformations.
- Seamless integration of user-defined syntax in rewrite rules, enabling the definition of transformation rules in concrete syntax, as opposed to using abstract syntax and getting much more obscure rules. This also guarantees fully syntax-safe source code generation.
- A highly modular and extensible architecture based on the coordination of language processing tools.
- ATerms as a language-independent intermediate data exchange format between tools.
- An Integrated Development Environment (IDE) that provides interactive support and on demand tool execution.

Version 2.0 of The Meta-Environment includes various new features that are directly relevant for software evolution:

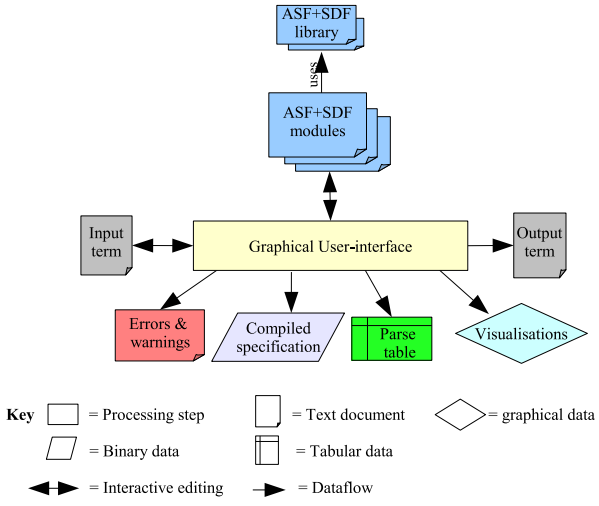


**Figure 2. End-user view**

- A grammar library containing grammars for C, Java, Cobol and other programming languages.
- Rewriting with layout. This enables fine-grained analysis and transformations such as high-fidelity source-to-source transformations that preserve comments and whitespace.
- Automatically generated syntax highlighting based on syntax definitions for arbitrary languages.
- Automatically generated prettyprinters that can be refined by the user.
- Rscript—a relational calculus engine that enables easy computing with facts extracted from source code.
- Advanced information visualization tools for the interactive display of relations, parse trees and dependencies.
- A fully customizable, plugin-based user-interface with dockable panes, and user-defined menus and buttons. Plugins can run user-defined scripts to interact with other tools.

A major architectural improvement in version 2.0 is the division of the system into several separate layers that enable the creation of a family of related applications that share common facilities such as user-interface, parsing infrastructure, and error reporting (the kernel layer). The facilities for syntax analysis (SDF layer) and transformation (ASF layer) are implemented on top of this kernel. See Figure 3 for an overview of this layered architecture.

## 3. Relevance for software evolution

The Meta-Environment is an ideal framework to implement tools that perform essential tasks when solving software evolution problems:

- The analysis that forms the basis to assess which maintenance actions are needed to remedy the effects of software evolution.
- The factory-like transformations that are selected on the basis of this analysis.

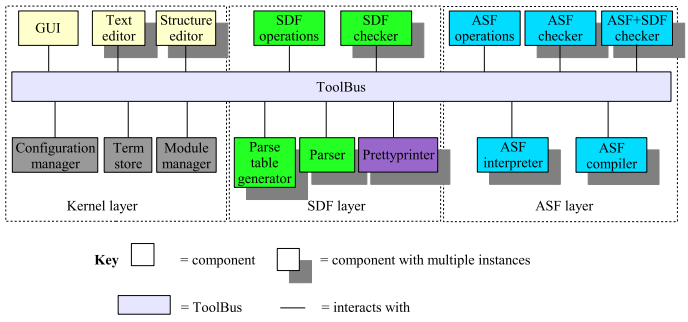


**Figure 3. Run-time architecture**

In the area of software evolution, The Meta-Environment has been successfully applied to the analysis of embedded SQL and the transformation of database schemas [2], Cobol prettyprinting [1] and restructuring [3], PL/I parsing, analysis and restructuring of C++, smell detection [5] and dead-code detection in Java, and also aspect mining in C.

Due to the many extension points (rules for defining syntax, prettyprinting, analysis and transformation; extensible user-interface; connection of third-party components; extensible architecture) the system can be easily adapted to the requirements of a specific software evolution or renovation problem.

To download The Meta-Environment and to access its documentation, publications and applications see [4].

## References

[1] Mark van den Brand, Taeke Kooiker, Jurgen Vinju, and Niels Veerman. A Language Independent Framework for Context-sensitive Formatting. In *CSMR '06: Proceedings of the Conference on Software Maintenance and Reengineering*, pages 103–112, Washington, DC, USA, 2006. IEEE Computer Society Press.

[2] Anthony Cleve and Jean-Luc Hainaut. Co-transformations in Database Applications Evolution. In *Generative and Transformational Techniques in Software Engineering*, volume 4143 of *Lecture Notes in Computer Science*, pages 409–421. Springer, 2006.

[3] Steven Klusener, Ralf Lämmel, and Chris Verhoef. Architectural Modifications to Deployed Software. *Science of Computer Programming*, 54:143–211, 2005.

[4] `http://www.meta-environment.org`.

[5] Eva van Emden and Leon Moonen. Java Quality Assurance by Detecting Code Smells. In *Proceedings of the 9th Working Conference on Reverse Engineering*. IEEE Computer Society Press, October 2002.