SAFARI: A Meta-Tooling Framework for Generating Language-Specific IDE’s

Philippe Charles, Julian Dolby, Robert M. Fuhrer, Stanley M. Sutton Jr., Mandana Vaziri
IBM T. J. Watson Research Center
19 Skyline Drive
Hawthorne, NY USA 10532
{pcharles,dolby,rfuhrer,suttons,mvaziri}@us.ibm.com

Abstract
Getting a new programming language into the hands of users is still a huge undertaking. SAFARI is an Eclipse-based meta-tooling framework for generating language-specific IDEs that greatly accelerates that process. It exploits common themes and structures that recur in many languages and language tools. It supports the generation of language-dependent IDE services, while allowing developers to focus on the language-specific aspects of their environments rather than the surrounding IDE framework. SAFARI has been used to generate IDEs for several languages. These IDEs include such features as parser generation; editors with keyword highlighting, text folding, text completion, hyperlinking and so on; outline views, project building; and more.

Categories and Subject Descriptors D.2.6 [Software Engineering]: Programming Environments—integrated environments, programmer workbench. D.2.3 [Software Engineering]: Coding Tools and Techniques—program editors.

General Terms Languages

Keywords Environment framework; SAFARI, Eclipse

1. Introduction
Programming language design remains a vital area of research. In particular, domain-specific and scripting languages have received much attention lately, as have languages for high-performance and parallel computing. A critical barrier to the practical evaluation of a new language, as well as to its widespread adoption, is the availability of a full-featured integrated development environment (IDE): users are loath to forego the accustomed navigation, visualization, and manipulation tools provided by modern IDE’s.

Unfortunately, despite advances in IDE extensibility, it is still quite tedious to build robust, high-functionality IDEs for new programming languages. Although much of the work in developing a language-specific IDE is inevitable, depending upon language-specific details of structure and semantics, a significant portion of this work embodies themes and code structures that are common across IDEs for a wide range of languages. These themes represent a significant opportunity for simplifying IDE development through meta-tooling and code reuse.

SAFARI is an ongoing project at IBM Research to develop such a meta-tooling framework for Eclipse. Our demonstration will show two SAFARI-based IDEs, and walk through the development of an IDE for a simple language using SAFARI.

2. Desired IDE Capabilities
A good baseline of functionality representative of the state of the art for language-specific IDEs is the Eclipse Java Development Toolkit (JDT). Ultimately, our goal is for SAFARI to support all of the Eclipse JDT’s rich set of capabilities:

- source code editors with syntax highlighting, annotations, hover help, compiler error indicators, text folding, and so on
- structural views including project navigators, source outline views, and type hierarchies
- content assist and completion; quick fixes to common errors
- navigation (e.g. hyperlinking, “find declaration”, “find type”)
- incremental project building and dependency tracking
- refactoring
- debugging

Longer term goals include capabilities such as dynamic analysis and visualization.

3. SAFARI Development Process
Development of a SAFARI IDE is an incremental process, divided roughly into the following major components:

3.1 Language Grammar and Parser
The development of a language-specific IDE using SAFARI begins with the selection or definition of the language syntax. SAFARI accommodates languages that are defined as incremental extensions to some pre-existing base language, as well as languages that have entirely new definitions. In either case it is necessary to provide a set of abstract syntax tree (AST) implementation classes. Although much of the work in developing a language-specific IDE is inevitable, depending upon language-specific details of structure and semantics, a significant portion of this work embodies themes and code structures that are common across IDEs for a wide range of languages. These themes represent a significant opportunity for simplifying IDE development through meta-tooling and code reuse.

SAFARI is an ongoing project at IBM Research to develop such a meta-tooling framework for Eclipse. Our demonstration will show two SAFARI-based IDEs, and walk through the development of an IDE for a simple language using SAFARI.

Copyright is held by the author/owner(s).
OOPSLA ’06 October 22–26, 2006, Portland, Oregon, USA.
ACM 1-59593-491-X/06/0010.
generate implementation skeletons for services such as syntax highlighting, source-text folding, hyperlinking, and outline viewing (among others). These skeletons must be completed by the IDE developer according to the specific syntax and semantics of the language and the desired characteristics of the IDE. Many of these implementations will make use of the token stream, the AST, and the results of various analyses (e.g. name resolution or type-checking). For example, syntax highlighting is often specified based on token kinds; the outline view is typically populated by an AST visitor or declarative specifications; source folding is driven from the AST structure, while hyperlinking and content assist typically require semantic processing such as name resolution.

3.3 Project Building and Program Compilation

Another important aspect of IDE behavior is project building and program compilation. SAFARI provides a wizard to create a builder skeleton from the SAFARI class library, which helps automate tracking of inter-compilation unit dependencies. The builder can then be implemented as a wrapper around an existing (external) compiler or as a new compiler implementation starting from the resolved AST. SAFARI does not presently provide any support for implementing the compiler itself. However, if the target language is Java-derived, Polyglot [1] may be used, and it is then necessary to implement the standard Polyglot analyses for type checking, reachability, and so on.

3.4 Refactoring and Analysis

SAFARI supports refactoring through the combination of a generic AST rewriting framework and an extensible static analysis engine. In the future SAFARI will also provide a library of common refactorings. The AST rewriting framework provides a language for specifying AST patterns and rewrite rules. Enabling this framework for a new language requires writing a language-specific adapter. For a new language, new refactorings will naturally have to be defined for it. If on the other hand the language is defined as an extension of an existing language, then the refactorings of the base language can be extended incrementally to address both any new language constructs and the differences in semantics. For the static analysis that underlies the refactorings, SAFARI builds on an existing scalable analysis framework, DOMO, developed within IBM Research. DOMO provides many building blocks for static program analysis, including efficient core data structures, fixed-point system representations and solution algorithms, along with a suite of extensible analyses such as pointer analysis, call-graph construction, effects analysis, and type inference. To enable DOMO to analyze a new language, a translator is written from the language-specific AST into a somewhat lower-level representation, which is extended as needed by defining new node types. Next, new instruction types in the DOMO intermediate representation (IR) are created as needed (e.g. there are ~10 instructions for the X10 language [3]). A translator from the new node types to the augmented set of instruction types is then written. Finally, constraint handlers are implemented for existing analyses to handle the new instructions.

4. Accelerating IDE Development

SAFARI helps to accelerate IDE development in several ways. It provides a framework that encapsulates common language-processing idioms in reusable classes and templates. It supports the automatic generation of high-quality ASTs, which are central to many IDE services. It makes extensive use of meta-tooling, including wizards that guide the user in the specification of services and focus attention on relevant APIs. It accommodates the definition of new languages and implementation of new services by extension from existing languages and services. Finally, SAFARI incorporates a robust, scalable and extensible framework for multi-language source analysis.

5. SAFARI Today and Tomorrow

SAFARI presently includes wizards, code generation, and a class library to support language services including parser generation, syntax highlighting, source-text folding, hyperlink detection, content outlining, content assist, hover annotations, hover help, parsing, and project building. Parsing support includes incremental scanning and parsing, and grammar-directed error recovery. SAFARI also provides support for debugging breakpoints (via JSR 045) and a framework of components for multi-language static analysis.

We currently have working IDEs for X10 (an extension of Java for highly concurrent programming) and the JikesPG parser specification language, as well as a prototype IDE for JavaScript.

In the future, we will continue developing the refactoring framework, along with adding more language services, such as indexing, searching and quick fixes, and will reduce our reliance on the Eclipse JDT for features such as the Package Explorer and project configuration. As our stable of languages grows, we will improve the support for language inheritance in the various wizards and core APIs. We plan to enrich the set of grammar annotations (e.g., for scoping constructs, name resolution), borrowing from the extant literature on attribute grammars. We are also working on several declarative languages to help simplify the specification of presentation characteristics and language semantics. Related projects by the authors and others at IBM Research are actively working on extending the program analysis and refactoring support to languages such as PHP and JavaScript. An open-source release of SAFARI is planned for early 2007.

Acknowledgements

This work has been supported in part by the Defense Advanced Projects Agency (DARPA) under contract No. NBCH30390004. SAFARI is based in part on ideas and an early prototype by Chris Laffra of IBM.

References