Toward a Language-Parametric Code Completion Editor Service

Daniel A. A. Pelsmaeker
Delft University of Technology
Delft, The Netherlands
d.a.a.pelsmaeker@tudelft.nl

Eelco Visser
Delft University of Technology
Delft, The Netherlands
e.visser@tudelft.nl

Abstract

Code completion is an editor service that suggests keywords and identifiers that are relevant at the caret location in the editor, from which the user can choose to either insert one or continue typing. This reduces coding errors and aids in discovering the possibilities in a language or API. Providing a code completion editor service for a new programming language requires development effort in addition to the effort required for defining the language. The goal of this work is to automatically produce an intelligent editor-agnostic code completion editor service that is parameterized only by the declarative specification of the language. We implement our approach in the Spoofax language workbench, which enables language developers to provide a declarative specification of their new programming language. We will use the declarative specification to produce a platform-agnostic code completion editor service for Spoofax languages automatically.

1 Introduction

Developing a new programming language comes with its own sets of challenges, from defining a proper syntax to implementing the semantics of the language. There exist various libraries, tools, and frameworks that help with one or more of these concerns, but none of these tools help the language developer get good editor support for their growing language.

One such editor service is code completion, which is an example of a code recommender system [7]. When invoked, code completion displays a list of proposals that may be relevant at the caret position, allowing the user to pick one to insert into the document. These proposals include keywords and code snippets, but also commonly used expressions, and the identifiers of variables, functions, and classes.

An intelligent context-aware code completion service proposes only keywords that are legal at the caret position, and identifiers that are visible and reachable. Additionally, it can filter the list of proposals to only show those whose type is compatible with the expected type at the caret location.

Code completion is valuable for both new and experienced users of a language. New users use code completion to discover the possibilities of a language or API, whereas experienced users use code completion to reduce typing and mitigate the risk of making code errors.

Traditionally, a code completion service has to be implemented for each specific language, usually as part of an editor plugin. The most popular editors can be extended to support additional programming languages through plugins, but each editor has its own architecture for this. Some IDEs, such as Visual Studio, provide only the bare interface for code completion and leave a lot of flexibility to the implementation. Eclipse and IntelliJ on the other hand provide functionality that removes boilerplate for implementations in languages that support it, but at the cost of flexibility [6].

Implementing just a single editor plugin for a language can be a daunting task, let alone when trying to support multiple popular editors [3]. But a code completion service could rely heavily on information already encoded in the language specification.

The Spoofax language workbench [2, 9] provides a number of meta-languages that allow the language developer to define these aspects of their language in a declarative way. Each meta-language handles a particular aspect of the language, such as the syntax definition, name binding, type information, and static and dynamic semantics. While previously a language developer had to write extra rules and constructs to support code completion [2], more recently Spoofax was extended to generate a context-aware syntactic code completion service from the syntax definition [1], but which is tightly integrated into the Spoofax core framework.

In this work, we present a new architecture through which we can provide an intelligent context-aware code completion editor service for both syntactic and semantic code completion. It uses both the syntactic and semantic aspects derived from the declarative specification of the language to only propose identifiers, snippets and keywords that are legal at the caret location.

2 Architecture

In its most basic form, a code completion editor service has to return a list of code completion proposals to be presented to the user. This service needs an editor-agnostic interface to allow any editor to call it.

As Figure 1 shows, code completion is invoked by the user in a particular context. The context is formed by the document and a caret location within it, together with the corresponding locations in the abstract representations of the program, such as the lexical scope and the term in the

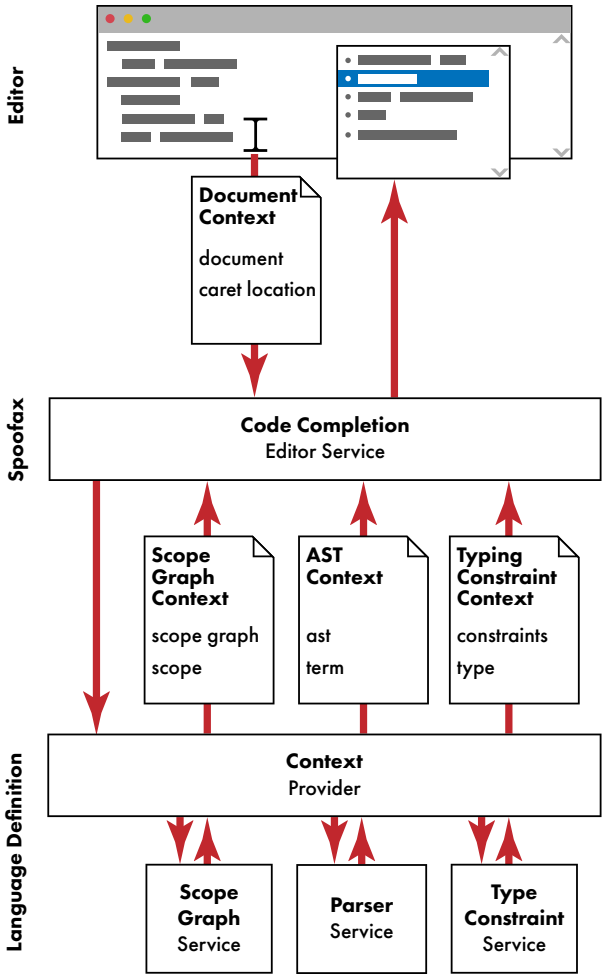

Figure 1. Overview of the code completion architecture. The code completion service uses the given document context to retrieve and query the corresponding language-specific contexts, through which the service determines the code completion proposals to return.

abstract syntax tree. The construction of these abstract representations is very specific to the programming language used in the document. Therefore, we need a generic and language-agnostic architecture that allows these abstract representations to be computed, retrieved, and queried, such that the code completion service can be both intelligent and language-agnostic.

Language developers can use the Spoofax language workbench to provide a specification for their language in a declarative way. From this declarative specification, we derive a number of services such as a parser service that produces an abstract syntax tree, and a semantic analysis service that produces a language-independent scope graph [5, 8] constructed as part of the name and type resolution algorithms used in Spoofax.

The code completion service needs to determine the context in which it was invoked. The document context, which is the document itself and the caret location, is provided by the editor. The other contexts, such as the corresponding term or scope, are returned by the language-specific context provider, which calls upon the generated services.

The service then queries the contexts to find, for example, the keywords applicable and identifiers visible at the caret location. It can also use this information to filter the proposals to only contain those whose type would be compatible with the type expected at the caret location. From this information, an ordered list of code completion proposals is constructed.

The code completion service is not specific to any one editor. We have described an editor-agnostic interface for code completion as part of the Adaptable Editor Services Interface (AESI) architecture [6]. This allows the editor service to be used with supported editors, without extra effort on the side of the editor or plugin.

3 Conclusion

In this work we present an architecture for intelligent context-sensitive code completion that leverages only the declarative language specification of the language itself.

We will evaluate this architecture in the Spoofax language workbench by using it to implement intelligent context-sensitive semantic code completion, and integrating this architecture into the upcoming Spoofax incremental and interactive build system [4].

References

[1] Luis Eduardo de Souza Amorim, Sebastian Erdweg, Guido Wachsmuth, and Eelco Visser. 2016. Principled syntactic code completion using placeholders. In *SLE*. 163–175. https://doi.org/citation.cfm?id=2997374

[2] Lennart C. L. Kats and Eelco Visser. 2010. The Spoofax language workbench: rules for declarative specification of languages and IDEs. In *OOPSLA*. 444–463. https://doi.org/10.1145/1869459.1869497

[3] Sven Keidel, Wulf Pfeiffer, and Sebastian Erdweg. 2016. The IDE portability problem and its solution in Monto. In *SLE*. 152–162. https://doi.org/citation.cfm?id=2997368

[4] Gabriël Konat, Michael J. Steindorfer, Sebastian Erdweg, and Eelco Visser. 2018. PIE: A Domain-Specific Language for Interactive Software Development Pipelines. *programming* 2, 3 (2018), 9. https://doi.org/10.22152/programming-journal.org/2018/2/9

[5] Pierre Néron, Andrew P. Tolmach, Eelco Visser, and Guido Wachsmuth. 2015. A Theory of Name Resolution. In *ESOP*. 205–231. https://doi.org/10.1007/978-3-662-46669-8_9

[6] Daniël Pelsmaeker. 2018. *Portable Editor Services.* Master's thesis.

[7] Francesco Ricci, Lior Rokach, and Bracha Shapira. 2015. *Recommender Systems: Introduction and Challenges.* Springer US, Boston, MA, 1–34. https://doi.org/10.1007/978-1-4899-7637-6_1

[8] Hendrik van Antwerpen, Pierre Néron, Andrew P. Tolmach, Eelco Visser, and Guido Wachsmuth. 2016. A constraint language for static semantic analysis based on scope graphs. In *PEPM*. 49–60. https://doi.org/10.1145/2847538.2847543

[9] Guido Wachsmuth, Gabriël Konat, and Eelco Visser. 2014. Language Design with the Spoofax Language Workbench. *IEEE Software* 31, 5 (2014), 35–43. https://doi.org/10.1109/MS.2014.100