論 文 の 内 容 の 要 旨
Abstract

論文題目

A Study on Static and Dynamic Programming Assistance for Embedded Domain-Specific Languages
（埋込みドメイン特化言語向け静的および動的プログラミング支援に関する研究）

氏名　　奥田　勝己

This dissertation presents our research to improve the domain-specific programming assistance for embedded domain-specific languages (DSLs). Using embedded DLSs is a promising approach to efficient software development for specific application domains. The designer of an embedded DSL can implement them with less effort compared to external DSLs. For example, the embedded DSL designer does not need to develop parsers for the embedded DSL since it is implemented as a library or framework for the host language. Moreover, the embedded DSL designer does not need to develop an integrated development environment (IDE) since the DSL user can use IDEs for the host language. However, using the syntax and IDEs of the host programming language limits the domain-specific assistance for the embedded DLS since only programming assistance for the host language is available in the embedded DSL. To provide better programming assistance for the embedded DLS, the embedded DSL designers have to develop additional tools. This effort reduces the benefit of the embedded DSL. To address this problem, we categorized programming assistance into static one and dynamic one and developed efficient methods for implementing each

assistance. The dissertation includes three studies: (1) proposal of lake symbols for island parsing, (2) proposal of interactive grammar editing for island grammars, and (3) design approach to embedded DSL for dynamic programming assistance. Study (1) and Study (2) achieve reducing the effort to implement static programming assistance for embedded DSLs. On the other hand, Study (3) proposed the importance of dynamic domain-specific assistance for embedded DSLs and the language design that can exploit the IDE for domains-specific assistance.

We define static programming assistance as domain-specific abstraction provided in the syntax level of the language. The key to an efficient implementation of syntax extension is to decrease the effort for developing a parser. The number of rules in the grammar that the parser depends on reflects the effort to developing a parser. The island grammar is a promising technique to reduce the number of rules in the grammar by omitting the rule for the uninteresting part of the language. Moreover, its application to syntax extension has also been proposed. However, the description of practical island grammar is complex because it requires a complex definition of the rule to skip the uninteresting part of the language.

The lake symbol proposed in Study (1) eases the description of island grammar. The lake symbol is a novel grammatical symbol similar to nonterminal symbols. The embedded DSL designer can use lake symbols as a wildcard symbol at the place in the grammar where she wants the parser to skip the input until it finds an extended programming construct of interest. The lake symbol automatically calculates symbols called alternative symbols that prevent lake symbols from skipping the interesting part of language as a wildcard. Without lake symbols, the embedded DSL designer must find alternative symbols manually and specify them in the island grammar. Previous work has been tackled the same problem to ease the description of island grammars. However, it calculates the subset of alternative symbols. This limits the place in the grammar where the parser can skip the uninteresting part of the language. Our lake symbols relax this imitation.

While the lake symbols ease the description of island grammar, writing island grammar is not easy. The description of correct island grammar requires iterations of trial-and-error. Hence, an efficient way to editing island grammars is required. Based on this motivation, Study (2) propose the interactive editing method and tool called PEGSEED. With PEGSEED, the language designer can write a working island grammar in a step-by-step manner. In each

step, she adds a rule for a new island. After adding a new rule, she can test the grammar on an example text by highlighting the text area recognized by the latest rule. A rule for a new island can be added by concatenating already tested islands. By incrementally refining the island grammar tested in each step, DSL designers can efficiently get the expected island grammar. PEGSEED also provides a GUI operation to add a new rule by using an example text. By selecting a text area and applying one of the GUI operations. The user can add a new rule without writing it by hand. Our case study shows that the parsers for syntax extension can be available only with the GUI operations provided by PEGSEED.

In Study (3), we introduce the importance of domain-specific programming assistance for embedded DSLs. We define auto-completion and error checking provided by IDEs as dynamic programming assistance. Careful language design enables dynamic domain-specific programming assistance via an IDE for the host language. We demonstrate this with our practical processor description language called MELTRANS. Our case study shows that domain-specific assistance can be available by exploiting an IDE for the host programming language. Moreover, because our design approach does not need customizing the IDE or developing a specialized IDE, it does not sacrifice the benefit of embedded DSLs that the construction cost is low.