Software Reuse in the GARMENT Approach[1]

Zongyan Qiu, Naixiao Zhang and Minghui Wang
Department of Informatics, School of Mathematical Sciences
Peking University, Beijing 100871, China
Email: zyqiu@pku.edu.cn, naixiao@pku.edu.cn, znx@sxx0.math.pku.edu.cn

ABSTRACT

GARMENT is a mechanism for abstraction and encapsulation of languages. It aims specially to make best support on the design and implementation of *Domain Specific Language* (DSL). This approach opens some new prospects for reuse of software artifacts. In this paper, we include a brief introduction to GARMENT approach, and a discussion of various possibilities of software reuse supported by the GARMENT approach. Some of them are new.

Keywords
Garment, Software Reuse, Domain Specific Languages, Component-Based Approach

INTRODUCTION

Software reuse is thought widely as a powerful means to improve the productivity of software development and quality of the software systems produced. A lot of works have been done in this field. Early works here can be found in [1]. In a survey paper of Charles W. Krueger [3], a number of technologies or approaches are summarized, which more or less involve reuse of software artifacts in different forms, and on various levels.

A useful concept proposed by Krueger is *cognitive distance*. The concept serves as an intuitive measurement in comparing different technologies. Briefly speaking, *cognitive distance* is the amount of intellectual effort that the software developers must spend in their work, for the purpose of using the existing software artifacts into their systems. The approaches listed in Krueger's survey paper include *High-Level Languages*, *Code Segments or Libraries*, *Application Generators*, *Transformation Systems*, etc.

GARMENT is an on-going project aims to explore an approaches for supporting domain-specific programming languages (DSL) development [7, 6]. The main work of GARMENT project is to define a meta-level specification language and to build a general supporting environment for easily specification and implementation of DSL. We propose, in recent work, a component-based framework for the specification of DSL. In the work, we find some novel directions for reuse of existing software artifacts, especially, reuse of languages design components, reuse of existing language design, etc. We report our view in this paper.

[1]The work is supported by National Science Foundation of China.

THE GARMENT APPROACH

The GARMENT project aims to establish a approach for rapid development of DSLs. A general supporting environment is built, which offers a meta specification language (MSL) for the specification of DSLs. A MSL processor is developed to produce DSL processors from the language specifications. A general-purpose source to source program-transformation sub-system is used to support the implementation of DSLs. Additionally, a language library is included as the storage of language facilities, to support the construction of new languages.

A specification of DSL in MSL takes the form:

garment *language_name*
  - - *definition of the language*
end *language_name*

When the MSL processor is applied to a language specification (a definition of a DSL), an implementation corresponding to the specification is produced. This implementation includes a DSL processor and a set of transformation rules (TRs). The DSL processor serves to analyse the syntax of the DSL programs and translated them into internal Abstract Syntax Trees (ASTs). With the TRs, the transformation sub-system transforms the ASTs to their corresponding ASTs in the target language. With the output facilities, target programs can be displayed on the screen or stored as source files, or compiled into executables. In our work now, ADA95 is chosen as the target language.

Using the GARMENT environment, the development of application systems is divided into two separate stages. In the first stage, the application domain is analyzed, and the corresponding DSL is designed and specified using the MSL. The DSL processor is built automatically from the DSL specification by the system. In the second stage, the application systems are designed and programmed with the defined DSL. This language can be more succinct than the general-purpose programming

languages (which are all very complicated, very hard to understand and handle) in the specific use, and more suitable for the special purpose of the target domain.

This approach is efficient for the application domains in which systems have a common model, and where many systems need to be built with low cost and/or short time-limit. Obviously, once a suitable DSL is implemented, only the second stage is necessary in the development of an application. This two-stage approach may also be very effective in shortening the *cognitive distance* in reusing the existences. If a DSL is very well designed, with respect to a special domain, reusing the facilities and structures embedded in the language is relatively easy and convenient for the domain specialists.

Moreover, because the DSLs have a common specification form in the MSL, and a common implementation model in the GARMENT system, the cost to implement DSLs may become lower. This will promote people to think about suitable DSLs for their specific domain, and may promote the DSLs to come into bloom.

REUSABILITY IN THE GARMENT APPROACH

Reuse in the Common Sense

With the two-stage process of the software development, following the GARMENT approach, the ordinary means for software reuse also have their places. For example:

- The implementation language, which serves as the target of the transformation, is a general-purpose high-level programming language (e.g., ADA95). The primitive structures of the language can be reused in the DSLs developed. There are two possibilities to use these structures. The first, a DSL can embed directly some of these language structures. Thus, users can use these structures in their DSL programs directly. Secondly, these structures may be used as the foundations to implement the higher-level structures of DSLs. This kind of usage is not visible, but very important too.
- Sub-programs in the target language, such as user-defined procedures, functions, data-types, etc., can be reused as usual. In the specification of a DSL, one can ask for the use of any target-language level entities existed, as building blocks for the implementation structures of the DSL.
- Existed DSL programs, or parts of the programs can be reused as those in programming of ordinary languages.

Beside of these, we have recognized some additional means to use software artifacts in the context of GARMENT approach. Before the detailed discussion, we first have a look on the software development process in this approach. Various activities related to software development here can be divided into three categories:

1. The design of the Meta Specification language and development of the GARMENT environment;
2. The analysis of application domains, the design of DSLs according to specific domains and the specification of the DSLs;
3. The implementation of domain-specific applications as programs in some DSL selected.

Obviously, the first kind of the work is done by the people who work directly in the GARMENT project. All knowledge and artifacts contained in this part of work will be reused in all applications developed in this environment. This reuse is implicit and obvious. Reusability in the third kind of activities is common, as described before. In GARMENT project, we have paid much attentions on the second kind of activities, and try to support definitions of DSL in various aspects, as much as possible. Some new prospects of reusing existing artifacts appear here. We have discussions on these in the following two sub-sections.

Language Components: A New Direction of Reuse in the Definition of DSL

Even if we have a detailed study on an special application domain, and have a skeleton of a DSL in our mind, formally specifying the language in the MSL is still a heavy work, with many principal points and trifles to deal with. For facilitating the specification of DSLs, and making the procedure with more flexibility, we proposed the concept of *component-based language definition*.

A collection of definitions, such as the collection of all description of lexical entities, the collections of descriptions about the forms of various expressions, of all statements, of all data types, etc., are all language components in the point of view in GARMENT. Instead of talking about components for construction of the application software systems, we think about components for specifying new languages, DSL especially. Following our view, definition of a DSL is thought as a construction from a number of assembly parts, the language components.

The specification of a DSL is divided into a number of sections. Each of the sections define one aspect (one component) of the language. The specifications has further the form:

garment *lan_name*
- - description of lexical entities
token_component
token_rule1
token_rule2
...
end of token_component

```
    - - description of expressions
    expr_component
        expr_rule1
        expr_rule2
        ...
    end of expr_component
    - - description of other components
    ...
end lan_name
```

From this description, we can see that a component is made up of a number of definition rules. Each rule defines a syntax structure of the DSL and the corresponding transformation. Rules in most components have the form:

rule_name aux_def con_syn $\Longrightarrow$ *transform*

The *con_syn* is in the left part of the rule. It defines the concrete syntax of a program structure of the DSL. The right part, the *transform* defines the *semantics* of the structure, by means of the structures of the target language (e.g., Ada95). The *aux_def* part is optional. It introduces in some local definition, for facilitating the description of remains. In fact, the new DSL definition also forms a program transformation system. In this aspect, each rule in the description plays role of a transformation rule, in which the *con_syn* part describes the match pattern and match conditions and *transform* part describes the substitution [3]. The general source to source transformation driver of the GARMENT system will use these rules to transfer the DSL programs into target form, the implementation.

The **type_component** is different. It takes a different form. The type component defines abstract types of the DSL. It contains a number of type definitions. A abstract type definitions consists of several parts. They are parts for literal descriptions, operations descriptions and so on. A type definition in **type_component** is indeed a module, which has the form:

```
- - description of an abstract type
type type_name (parameters)
  literals
    definition of literal1
    definition of literal2
    ...
  operators
    definition of operator1
    definition of operator2
    ...
  - - other definitions
  ...
end type_name
```

Moreover, the type definition can have type parameters to support the concepts of generalization and polymorphism.

Dividing language into components is not only a method to separating different parts of the language definition, to make it more clear and tractable, but also a new level of construction and abstraction. In fact, this new level introduces other possibilities for reuse of existing software artifacts.

In the work, we try to make the language-components independent from each other up to some degree. Thus, an existing component can be reused when people want to define new languages. Any section of a new DSL specification can include an existing language component, as the foundation for its definition, and extending the existing structures by a number of new definitions.

For example, an expression component can be defined by extending from another expression component named E1 by the form:

```
- - description of expressions
expr_component extend E1
    expr_rule1
    expr_rule2
    ...
end of expr_component
```

Thus, the expression component of the new language includes both the rules in existing component E1 and that defined newly. This is the way to reuse the existing component.

For supporting reuse of the components, we introduce in a facility called component library maintained by the GARMENT environment. Language components can be defined by above form and stored in this library. These components are directly available in definition of new languages. New components can be added and reused too. With this component library, efforts which have to be paid in specification of a new language will decrease greatly.

The components library seems similar to what people talk about commonly, but it is in a totally different level. The components stored in this library are building blocks for definition of new languages, DSLs, but not applications as usual. We are currently in study the relationship of these two concepts.

Language Reuse: Inheriting an Existing DSL

Languages defined in GARMENT are assumed to be procedural languages with structures like data-types, program variables, assignments, control structures, procedural abstraction mechanisms, etc. All of these can be

designed as suitably as demanded according to the specific application domain.

When a new language is required, we need to have a specification for all of its necessary features. As we said above, in the GARMENT environment, components of the new languages can be defined based on existing components by extending. Furthermore, a new language can also be described by extending from an existing DSL. In our notation, new language L1 can be defined by inheriting all features from an existing language L0 using following form:

garment L1 **extend** L0
    - - *definition of* L1 *by extending*
end L1

With this form, all structures and mechanisms of language L0 are inherited by the new language L1, and can be used in all L1 programs. The MSL processor will combine all rules defined in the components of L0 with those described specially for L1, and produce a DSL processor according to these. This should be recognized as another new way to reuse software artifacts.

CONCLUSION

With GARMENT approach, another level of abstraction, the language level is established. We have a study on reuse of the existing software artifacts on this level, and recognize some more possibilities. We think that language-level reuse is a very interesting topic for further investigation. In the literature, people think that design and effective implementation of DSL will become an important field in the near future. We think that GARMENT approach and the *Component-based DSL Specification* will play their roles in this field. Many further works can be started now.

Many works in the field of Domain Specific Languages are reported, for example, [5]. Some relative works in the literature take similar view of software development as ours, for example, [2, 4]. The GARMENT project puts its aims more on the aspects to support general DSL specification and automatic implementation. Interesting readers can refer papers [6] for some more details. Some simple languages have been built following the GARMENT approach. A number of more complicated languages are under working, such as, a language for supporting network-based decision-making. On the same time, we still work in improving the meta-language and rebuilding the supporting environment.

REFERENCES

1. T.J. Biggerstaff, A.J. Perlis, eds, Software Reusability, Volume I and II, ACM Press, 1989.
2. Richard E. Faith, Lars S. Nyland, and Jan F. Rrins, KHEPERA: a system for rapid implementation of domain specific languages. in [5], 1997.
3. Charles W. Krueger, Software reuse, ACM Computing Surveys, Vol. 24, No. 2, June 1992.
4. Yannis Smaragdakis and Don Batory, DiSTiL: a transformation library for data structures. in [5], 1997.
5. UNENIX Association, Proceeding of Conference on Domain-Specific Languages, Santa Barbara, California, Octor 1997.
6. Naixiao Zhang, Hongjun Zheng and Zongyan Qiu, Garment—A Mechanism for Abstraction and Encapsulation of Languages, ACM SIGPLAN Notices, Vol.32, No.6, p53-60, 1997.
7. Hongjun Zheng and Naixiao Zhang, An abstract model for programming languages, International CASE Symposium'95, Changsha, China, October, 1995.