Reflective interfaces

A Position paper for the OOPSLA'93 Workshop
on Reflection and Metalevel Architectures

Donald Liib

Dept of Teleinformatics
The Royal Inst of Tech
Electrum 204, 16440 Kista
Sweden
e-mail: donald@it.kth.se

Integration supported by reflective techniques in heterogeneous distributed computations in the context of object-based programming is presented.

As the development of technology in one domain has its physical limits there is need for such computational models that can build or make explicit interfaces between different domains that results not only in better interaction but also in creating new properties supported by underlying system. Reflective techniques are under development to make such process feasible via softening the boundaries between different programming domains and allow possibly mutual extensions.

Advanced real-time programming system is characterized as flexible system. I claim that reusability can guarantee flexibility which is in turn portability dependent. If those conditions are fullfilled then system can always behave properly in interaction to the environment to find balanced state between goal-orientedness and situation-orientedness. It contains not only development of reflective techniques but the compilers and the operating systems that have flexible real-time systems features, are extensible and modifiable. There is already under development programming systems [IcMaYo92, HoTo92, Yok92].

Reflective programming system is considered as two-level system, where the base-level contains abstraction and the meta-level implementation. Basically the causal connection between two-levels can determine the nature of computations whereas those two levels can be used separately. As a conclusion the view to software engineering, that abstraction hides implementation is not generally possible. One can say that given abstraction is not necessarily adequate to the implementation. For example, XWsuch situation can be seen when working with application program. When user wants to add some new features to the application system to make it more adequate to its needs, he extends the application program usually. It is the only possibility, because there is no access to the underlying software systems like window system and operating system from within application. The interface between those systems are not visible to the user. As a result such extensions make the application program code more complex and the portability issues for the future can become obstacle. Open implementation as new view to the programming system development is needed [Ki92].

In other terms programming system has many levels of abstraction where those abstractions are programmed in different languages and have different functionality. There have been some attempts to open the implementation using reification mechanisms to make certain aspects of reflection explicit. So instead to make ad-hoc changes on upper-level of system abstraction one can work out systematic approach how to move to the proper level and make most efficient changes.

Communicational reflection [Li92a] is proposed to support the reification process. Communicative reflective computations are characterized as the system ability to chose such order of computations that quarantee reasonable efficiency for reflective computations. Especially in case of interactive separately compiled heterogenuous systems the requirement for portability becomes important.

How to make an object more flexible? What are the object-based programming techniques is needed to fullfill requirement? One possibility is to apply reification in top-down programming manner. It can give certain results, but experience as shown that the power of reification applied to system that is not self-adaptive is limited. Another approach, what I propose, is to build general as possible interfaces for the system that can function as flexible encapsulation [Li92b]. The following requirement are applied to the flexible encapsulation

- make explicit domain dependent aspects of implementation
- simplify system representation according to implementation to make it more portable

and extensible
- performance maintance for proper behavior selection
- reflective techniques to access certain parts of implementation in order to modify and make them explicit.

Flexible interface contains system shell, where known types, procedures, functions are described. One interesting feature of flexible encapsulation is to control the attribute private/public changing. It gives certain efficiency in opening the implementation and quick access to those parts of the system what are needed to modify.

Object-based programming system PL/LL have been under development [ThLi92]. It is two-level programming system, where LL stands for linking modules into systems and PL[ThLi91, LiTh92] stands for compiler to compile modules as system building blocks and generate interfaces that suit for linking. Composition, restriction, renaming and parameter supplying are main operations supporting linker activities. Composition is based on solving export-import relationship between modules applying restriction to the exports and renaming to both exports and imports when needed. Composition operation is legal when exported concepts of composable modules are disjoint, i.e.linking of an LL-expression is successful if all composition operations in the whole expression are legal, otherwise illegal.

Reflective techniques will be developed to make linking more advanced via adding adaptivity to the composition operation. As a result modules can make mutual changes so that the composition will be legal. The process of reification and reflection can be activated and at least reliable answer will be given, if it is unknown how to make changes or changes are not reasonable. As a result it simplifies the LL-expression specification and restriction and rename operations will have richer semantics.

References

[HoTo92]
Yasuaki Honda, Mario Tokoro: Soft-Real-Time Programming through Reflection, Reflection and Meta-Level Architectures, Intern Workshop on New Models for software Architectures'92, Nov. 4-7, 92, CSK Education Center, Tokyo, pp. 12-23.

[Ki92]
Gregor Kiczales: Towards New Models of Abstraction in Software Engineering, Reflection and Meta-level Architectures, (as previous), pp. 1-11.

[IcMaYo92]
Yuji Ichisugi, Satoshi Matsuoka, Akinori Yonezawa:RbCL:A Reflective Object-Oriented Concurrent Language without a Run-Time Kernel, (as previous), pp. 24-35.

[Li92a]
Donald Liib: A Note on Communicational Reflection. Position paper for the ECOOP'92 Workshop on object-oriented Reflection and Metalevel Architectures, Uthrecht, the Netherlands, 30 Juni 92.

[Li92b]
Donald Liib: Flexible encapsulation, Position paper for the OOPSLA'92 Workshop on object-oriented languages: the next generation, Vancouver, Canada, Oct. 18-22 92.

[LiTh92]
Donald Liib, Lars-Erik Thorelli: A Compiler for the Object-based Programming Language PL, TRITA-TCS-EDA-9204-TR,KTH, 1992, 19p.

[ThLi91]
Lars-Erik Thorelli, Donald Liib: Introduction to PL: the language and simple Target Machine, TRITA-TCS-9105, KTH 1991, 30p.

[ThLi92]
Lars-Erik Thorelli, Donald Liib:PL/LL - An object-based programming system with a linking language, Software Practice & Experience (submitted), 25p.

[Yok92]
Yasuhiko Yokote: The New Mechanism for Object-Oriented System Programming, Reflection and Meta-Level Architectures, International Workshop on New Models for Software Architectures'92, Nov. 4-7 92, CSK Education Center, Tokyo, pp. 88-93.

----------------end of message--------------------