A Flexible Integral Computing System based on a Structurally-Reflective Abstract Machine

F. Ortín Soler, D. Álvarez Gutiérrez, and J. M. Cueva Lovelle

University of Oviedo
Department of Computer Science
Calvo Sotelo s/n 33007, Oviedo, Spain
e-mail: {ortin, darioa, cueva}@pinon.ccu.uniovi.es

Abstract

Currently, integrating and interconnecting different computing systems on different platforms is a problem without a definite solution. An integral solution allowing the manipulation of the system as a whole in a flexible way has not yet emerged.

An abstract machine endowed with structural reflection implementing a reflective object model can be the basis to develop a multiplatform integral system, highly flexible and offering the abstraction of a single distributed computing system.

By developing an interpreter on the machine, computational reflection is introduced in the system, making the dynamic change of the language interpreted possible. A generic and methodical tool to develop these interpreters complements this. Besides, any programming language is able to access the object model defined by the abstract machine.

Keywords: Abstract Machine, Structural and Computational Reflection, Language Independence.

1. Introduction

1.1 Systems integration

There are some initiatives to integrate different computing systems, applications and programming languages. Object architectures such as CORBA [8] or COM [13] specify a model in which objects interoperate because of the standard definition of language-independent interfaces.

These systems lack the flexibility to modify the interconnection mechanism. Additional projects such as OpenCorba [14] try to remedy this using reflection. Besides, a single computing system is not envisioned, but one in which an application divided in subsystems is interconnected by a middleware. Each subsystem should be compiled and may use its local platform resources (not in an integral way).

Other alternatives propose to develop applications using assorted platforms and languages. Computation is divided into multiple layers, departing from the client-server concept. Interconnection is not done by invocating methods that appear in an interface. Instead, self-described data is sent between the different systems. An example is an n-tier system, in which XML [5] files are exchanged using HTTP or HTTPS protocols, interconnecting powerful hosts and thin clients. This is more flexible, as no fixed architectures as COM or CORBA are used, but no single computing system is adopted. An application must be divided into different physically-distributed platforms.

1.2 Our reflective solution

We favour a single distributed computing system developed on the basis of a very simple object-oriented abstract machine, endowed with structural reflection. Upon this foundation, the set of services of the computing system is built by dynamically accessing objects by means of structural reflection. Then, a tool for the dynamic specification of languages is developed, introducing computational reflection and language independence in the system.

Next section overviews the description of the abstract machine and the design of the computing environment by means of structural reflection. Section 3 shows how to get computational reflection on the existing machine. Section 4 presents a particular design to get language independence that builds on the reflection introduced in the system. Conclusions are drawn in the final section.

2. A Computing System based on Structural Reflection

Developing and using a single distributed computing system is more powerful than the before-mentioned systems. As an example, garbage collection, thread scheduling and object mobility would be carried on the set of computers assembling the system in an integral way. Using an abstract machine helps the implementation of the system, as it offers a single object model easing object interoperability.

The system will add functionalities such as persistence, distribution and security, being at the same time flexible enough to modify and extend any of the services (the persistence system of a platform could be modified [6], for example). The machine should also be small enough as to be easily portable to any platform.

2.1 Flexibility by means of structural reflection

Our previous research for integral-computing systems used an abstract machine with a basic object model [3]. Then, different features such as distribution, capability-based security and persistence were added to the machine, mostly in "user" space, but required some changes to the basic machine. An object-oriented database engine [6] was built on the persistence feature. This kind of extensions to the machine had two limitations:

1. Flexibility in these functionalities gets reduced, as some modifications do need a recompilation of the abstract machine.
2. The size of the abstract machine, although not excessive, gets increased. This limits portability to size-constrained platforms. Maintenance of versions gets worse, also.

We then decided to redesign the abstract machine that forms the basis of the system, to reincarnate as a basic system of object computing primitives, with structural reflection as the built-in flexibility and extensibility mechanism. The machine (the kernel of the system) will then be as lean as possible, but powerful enough as to be extended with functionality adapted to the target environment of the system.

2.2 The abstract machine of the computing system

The abstract machine provides a pure object model based on prototypes [2]. This object model is simpler than object models based on classes, retaining the same expressive power [2]. Self [10], and to some extent Smalltalk [7] are some platforms that use this model. Our machine, however, has a simpler structure and computing primitives.

The object is the only abstraction. An object can dynamically inherit from other object (this is also called delegation), and can contain an unlimited number of object references (association relationships). There are only two primitive objects: the root object (*nil*) and string objects.

An object's structure is accessible at runtime by means of structural reflection. An object can send a message to other object using an object. Computing is performed by evaluating an object as a computation (like the *eval* function of LISP, evaluates a list as an instruction). Therefore, objects are the only abstraction. The methods of an object are then just aggregate objects than can be evaluated.

The basic primitives for constructing and destroying objects, message passing and the inheritance mechanism are implemented by the machine, and belong to the *nil* object that is the root of every object.

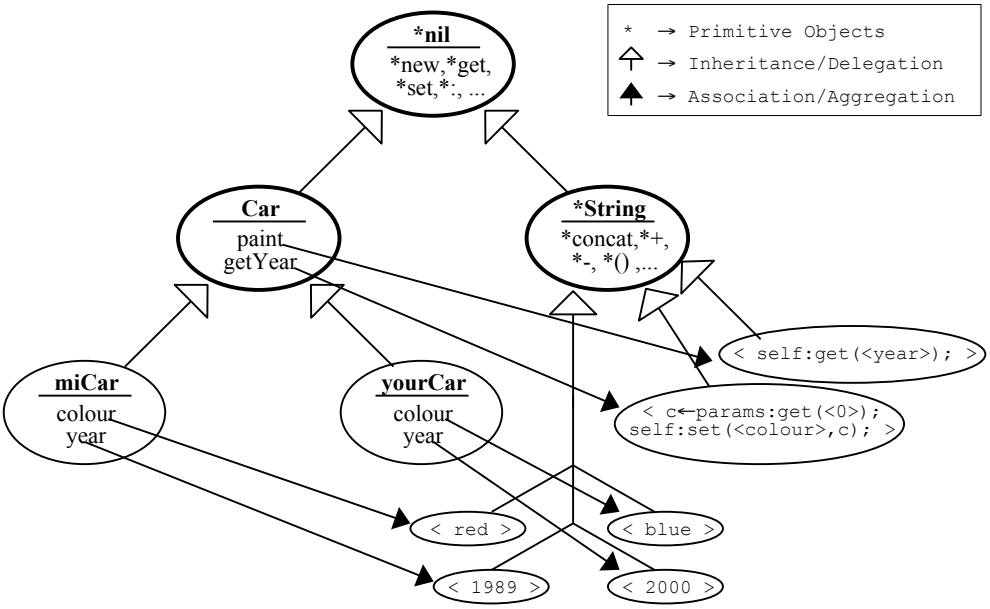

Figure ¡Error! Argumento de modificador desconocido.**: Prototype based object representation.**

Any additional functionality such as garbage collection, scheduling, persistence, distribution, etc. will be implemented if needed as objects external to the machine. Structural reflection allows to modify the objects without the need to modify the machine itself, thus flexibility to adapt the functionality is achieved. Garbage collection, for example, will extend the way objects are created, recording additional information to implement this collection.

3. Computational Reflection by means of Structural Reflection

A flexible computing system is achieved using structural reflection through an abstract machine that defines an object model. A dependence on the language of the virtual machine exists, though, as applications still have to be developed using that language.

3.1.Endowing the System with Computational Reflection

Computational reflection is the behaviour shown by a computing system that can access and act upon itself with the support of a causal connection mechanism [9]. So, computational reflection reifies computation in a system, and therefore its behaviour. A computational-reflective system may change its own semantics. As an example message passing mechanism could be increased to develop a distributed system.

To achieve computational reflection in the system we will develop an interpreter on the computing system, and then language independence is also achieved.

3.2.Design Strategy for the Interpreter

An interpreter for a high-level language is built. By means of a reifying instruction in the language, it is possible to evaluate a set of instructions of the abstract machine. The interpreter will take these instructions and will pass them to the machine for evaluation.

These instructions can modify (using structural reflection) the objects of the interpreter itself (and therefore the objects describing the computation). In this way, a program written in a given programming language is able to modify the semantics or even the language itself in which it is implemented.

The end result is a system which exhibits computational reflection by means of structural reflection using a two-level interpreter tower [1].

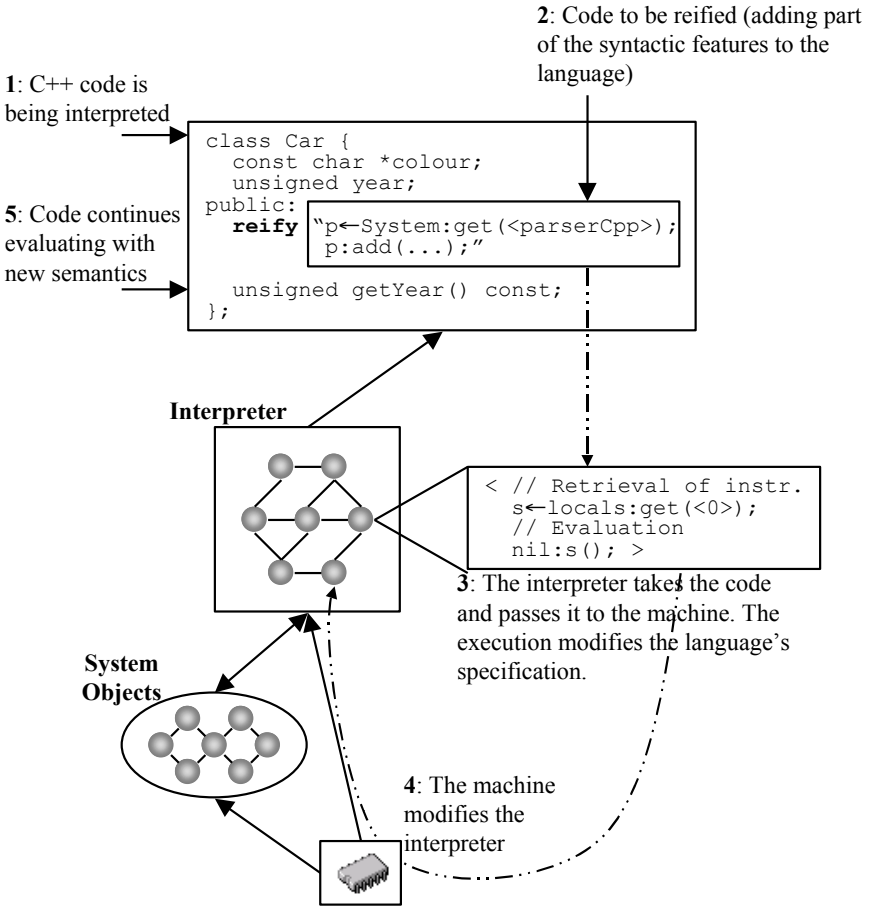

Figure ¡Error! Argumento de modificador desconocido.**: Achieving Computational Reflection.**

4. Language independence

The interpreter achieves language independence as it can be modified at run time. However, the dynamic change of an interpreter is a complex task, which needs a well-defined structure in order to be done in a simple way.

4.1 Language specification

There are many tools for developing language processors. Lex and Yacc are classic examples. Javacc [11] and antlr [12] are newer. These tools use a specification of a lan-

guage written in a given format, which is then pre-processed and compiled. This pre-processing step limits the dynamic modification of the specification of the language.

We propose to use a lexical, syntactic and semantic specification tool based on an object structure. The interpretation of a language is defined by an object graph built with objects of the object model of the machine. The evaluator of this graph making use of the *Visitor* pattern [4] will produce the interpretation of the language specified.

Dynamic modification of the objects that specify the language can be done by means of the structural reflection of the system. We are then able to change at runtime the language being interpreted. Different specifications of programming languages can be stored (by using the persistence subsystem implemented on the machine [6]) and then a selection of the appropriate language can be invoked.

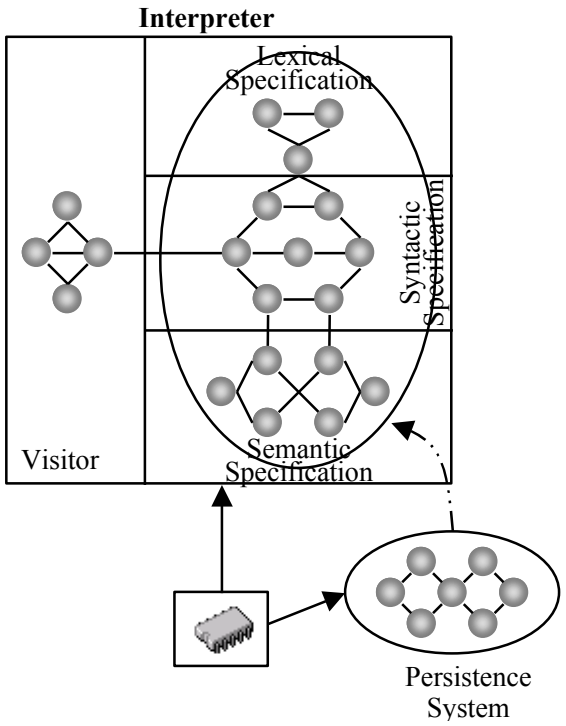

Figure ¡Error! Argumento de modificador desconocido.**: Structural Reflective Language Representation.**

The end result is a flexible distributed computing environment with a single object model that can be accessed using any programming language.

5. Conclusions

The integration and interconnection of heterogeneous systems currently has different approaches, but a flexible single integral computing system that is able to easily modify and extend system services has not been achieved yet.

Reflection is the key feature we deem essential to achieve that goal. Initially, structural reflection is used to develop a computing environment based on a simple abstract machine that defines a reflective object model based on prototypes, with only two primitive objects: string objects and the *nil* object that has the primitive operations for constructing and destroying objects, message passing and inheritance. The functionality of the machine is extended by dynamically accessing the structure of the objects.

There is a dependence on the language of the machine. Developing an interpreter on the machine is the way to get computational reflection and language independence. The interpreter is built with a graph of machine objects, which are evaluated using the *Visitor* pattern to produce the interpretation of the language. Using structural reflection these objects are modified to change the semantics of the interpreted language, thus achieving computational reflection.

A tool for the structural specification of programming languages is proposed to develop these interpreters in a generic and methodical way. Modification of these interpreters is done following a well-defined scheme.

Applications written in an arbitrary programming language can be executed, dynamically loading the specification for the language previously. Before executing a source program, the language being interpreted is specified and modified using computational reflection.

Reflection achieves a flexible integral computing system that defines an object model that can be accessed with independence from the programming language.

6. References

[1] B. C. Smith. Reflection and Semantics in a Procedural Language. MIT-LCS-TR-272. Massachusetts Institute of Technology. 1982.

[2] Borning, A.H. Classes Versus Prototypes in Object-Oriented Languages. *In Proceedings of the ACM/IEEE Fall Joint Computer Conference*. 1986.

[3] Darío Álvarez, et al. An object-oriented abstract machine as the substrate for an object-oriented operating system. *11th European Conference on Object-Oriented Programming (ECOOP'97)*. Jyväskylä (Finland). June 1997.

[4] Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides. Design Patterns. Addison-Wesley. 1994.

[5] Extensible Markup Language (XML) 1.0. World Wide Web Consortium. February 1998.

[6] Francisco Ortin, Dario Alvarez, Belen Prieto and Juan M. Cueva. An Implicit Persistence System on an OO Database Engine using Reflection. *International Conference on Information Systems Analysis and Synthesis (ISAS'99)*. Orlando (USA). July 1999.

[7] Goldberg A. y Robson D. Smalltalk-80: The language and its Implementation. Addison-Wesley. 1983.

[8] Object Management Group (OMG). The Complete Formal CORBA/IIOP Specification. October 1999.

[9] Pattie Maes. Issues in Computational Reflection. Meta-Level Achitectures and Reflection. North-Holland. Belgium. August 1987.

[10] Randall B. Smith, David Ungar. Programming as an Experience: The Inspiration for Self. Sun Microsystems Laboratories. 1995.

[11] Sun Microsystems. Java Compiler Compiler 0.8pre1. 1999.

[12] Terence Parr. Antlr Reference Manual. Magelang Institute. January 2000.

[13] The Component Object Model Specification. Microsoft Corporation. October, 1995.

[14] Thomas Ledoux. OpenCorba: a Reflective Open Broker. *Reflection'99*. Saint-Malo.

France. July 1999.