Automatic Implementation of Distributed Systems Formal Specifications

Luiz Henrique Castelo Branco[1], Antonio Francisco do Prado[1], Wanderley Lopes de Souza[1], and Marcelo Sant'Anna[2]

(1) Departamento de Computação, Universidade Federal de São Carlos, Av. Washington Luiz - 235, 04499-610 São Paulo, Brazil
{branco, prado, desouza}@dc.ufscar.br
(2) Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, R. Marquês de S. Vicente - 225, 22453-900 Rio de Janeiro, Brazil
santanna@inf.puc-rio.br

Abstract. The increasing demand for Distributed Systems(DS's) raised the need of a quality-assured development process, which could not only address the issue of requirement compliance, but also could help the construction of tools able to derive implementations automatically. In order to attend such a need, some Formal Description Techniques (FDT's) have been proposed. This paper defends the transformational approach as a good strategy to carry out the automatic implementation of DS's expressed in FDTs, focusing Mondel as FDT, and the DRACO-PUC environment as transformational system.

Introduction

Distributed Systems (DSs) are becoming increasingly popular. They have been used to meet the need of natural distribution of people and information, to allow for improved and more cost effective performance, to facilitate maintenance and to make computer systems highly reliable. To achieve all these objectives, DSs have become very complex and there is no consensus regarding an exact definition of DSs. According to Bochmann [1] a DS can be classified according to the four types of distribution. These types of distribution can be combined, increasing the system complexity, and must be carefully taken into through the different phases of a DS development.

In the specification phase the system is designed based on the user's requirements. Frequently many steps are necessary until a final specification of the system can be reached. In the implementation phase an instance of the system specification is produced in a high level programming language (e.g., Pascal, C, C++, Java) using software engineering techniques.

This paper concentrates on the implementation of DS's formal specifications. Section 2 deals with the subject of Formal Description Techniques and presents the specification language employed in this work. Section 3 introduces the approach and the tool used to derive automatic implementations from DS's formal specifications and the implementation issues, like a communication infra-structure. Finally, some concluding remarks are made in section 4.

Formal Description Techniques

Distributed Systems may be structured as a set of hierarchical layers [4] where each layer uses the services provided by the layers below it in order to provide its service to the next layer up. A service layer can be modeled as a black box. The service specification describes the external behavior for this black box when it interacts with the environment (users) through its interfaces (interaction points).

The main goal of the formal specifications is to produce unambiguous system descriptions. A formal specification can also be useful in other phases of a software production process. Some desired properties can be verified, (semi-)automatic implementations can be generated and the formal specification can be used as a reference for system implementation conformance testing.

A Formal Description Technique (FDT) is needed to produce formal specifications for distributed systems. FDTs are self-contained specification languages, which means that the system specification given in an FDT need not to refer to any informal knowledge about the system. A FDT must also have a mathematical basis that can be used to demonstrate a specification correctness.

Mondel Language

Mondel is the result of a joint research project involving the Centre de Recherche Informatique de Montréal (CRIM), the Université de Montréal (UdeM) and Bell Northern Recherche (BNR). This project focuses on modeling of operational and management aspects of communication networks, in particular the detection and recovery of failures. Despite its initial focus, however, its basic principles appear to be sufficiently generic to apply it to other types of distributed systems [3,2] such as real time control systems and open distributed processing. Because it is an object-oriented specification language, it uses terminology that is very similar to terminology used in the object-oriented paradigm.

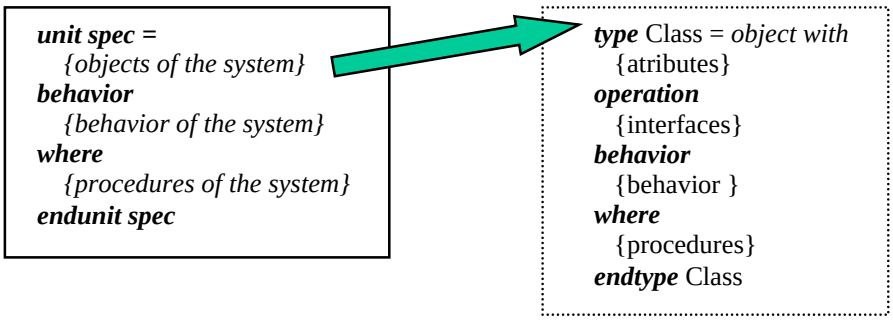

Fig. 1. The Mondel structure. An unit spec, as shown in figure, represents the specification properly said. In Mondel, the entities that communicate are represented as objects whose interaction points are represented by object interfaces.

A Mondel object is defined by the reserved word type, followed by its attributes. The signature and behavior of the methods offered by the object are indicated by the interface and behavior clauses, respectively.

In Mondel, states are represented by procedures. The current state is represented by the name of the procedure, and the next state is represented by the name of another procedure, or the same in case it continues in the same state.

The following section justifies the transformational approach as a language independent means to generate DS's implementations from their formal specifications. At the same time, the transformational environment DRACO-PUC is introduced, showing how it was successfully used in the implementation of DS's specified in the Mondel FDT.

Implementation Approaches

Along the last decade, several FDT's have been proposed, yelding a plethora of language dependent tools. At the same time, research has been done in the software engineering area to overcome the burden of having to develop a new compiler as a new FDT or implementation language arises. In this sense, the transformational approach comes as a natural choice, as it represents the state-of-the-art in compiler technology.

The DRACO-PUC environment, presented in the following sub-section, is a meta-compiler comprised of a domain network containing the domains Mondel, Estelle, C++ and Java, among others. In this environment, the language descriptions are kept apart from the semantic and transformation rules. This modularity saves effort when defining a new edge in the domain network.

This work produced Java implementations of DS's written in Mondel. Reusing the Mondel and Java domains, it is now possible to describe transformations between these high-level domains with a great economy in effort and time.

The DRACO-PUC Environment

The DRACO-PUC environment uses the ideas of the DRACO paradigm [6], which states that it is possible to develop software based on the reuse of high level abstractions. It is a domain-oriented transformational system where system or software descriptions, in high level specification or programming languages, may be automatically transformed into executable code. The following parts define a domain for DRACO:

- a **language**: a syntax must be well defined in order to make it possible to write programs and to build a parser.;
- a **prettyprinter**: this is an unparser responsible for mapping the internal Draco representation into concrete syntax representations of the domain language, that is, exhibits the application DAST in the domain language; and
- the **transformation libraries**: The handling of the DAST is carried out by formally defined steps called transformations. Each transformation rule has a Left

hand side (Lhs) describing the recognition pattern, and a Right hand side (Rhs) describing the rewriting pattern. The normal execution order of the transformation rules can be changed by tasks executed in association with the check-points available in the Draco machine. The main transformation control points are: Pre-Match (executed every time a Lhs rule is tested on the input description), Post-Match (executed just after the Lhs rule matches some piece of the input description), Pre-Apply (executed immediately before the Rhs rule is applied, replacing the selected piece of the input description) and Post-Apply (executed just after the input space selected on the Lhs is replaced by the pattern declared in the Rhs).

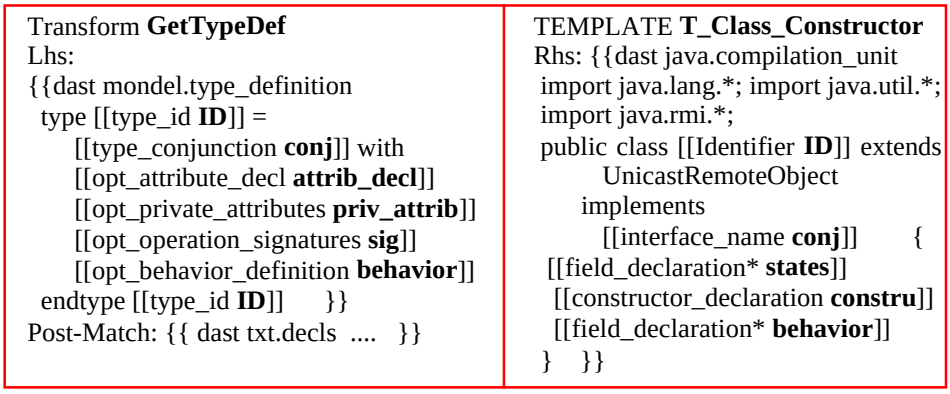

Fig. 2. Internal aspects of the DRACO Transforms. Figure shows one of the transformations that is used to transform an Mondel type declaration into its corresponding Java class. The left side is related to the Mondel domain, while the right one is related to the Java domain.

The **Lhs** of the **GetTypeDef** *Transform* contains the pattern to be recognized according to the **type_definition** Mondel grammar rule. The word **ID** is a meta variable of **type_id** type, **conj** is a pattern variable of type **type_conjunction**. This *Transform* does not have an **Rhs**. Instead, it does some data manipulation in its *Post-Match* control point and calls the **T_Class_Constructor** *Template*.

The **Rhs** in **T_Class_Constructor** contains the pattern to be written in the Java domain. The word **ID** is a meta variable of **Identifier** type and **conj** is a meta variable of **interface_name** type.

It should be noted, by Figure 3, that the transformations are generic. The meta variables are placeholders that hold grammatical patterns independently of the specification. Therefore, the Mondel operational semantics implemented in the transformations are reused across implementations.

The principal steps taken to use the Draco Domains in the automatic implementation of distributed systems specifications (Mondel) into executable languages (Java), using the Draco machine are presented in the next section.

Setting up the Implementation Environment

Using the strategy proposed by Prado [9], it is possible to rebuild software by direct porting of the source code to languages of other domains. According to the Draco machine, a domain consists of three parts, i.e. a *parser*, a *prettyprinter*, and one or more *transformers.*

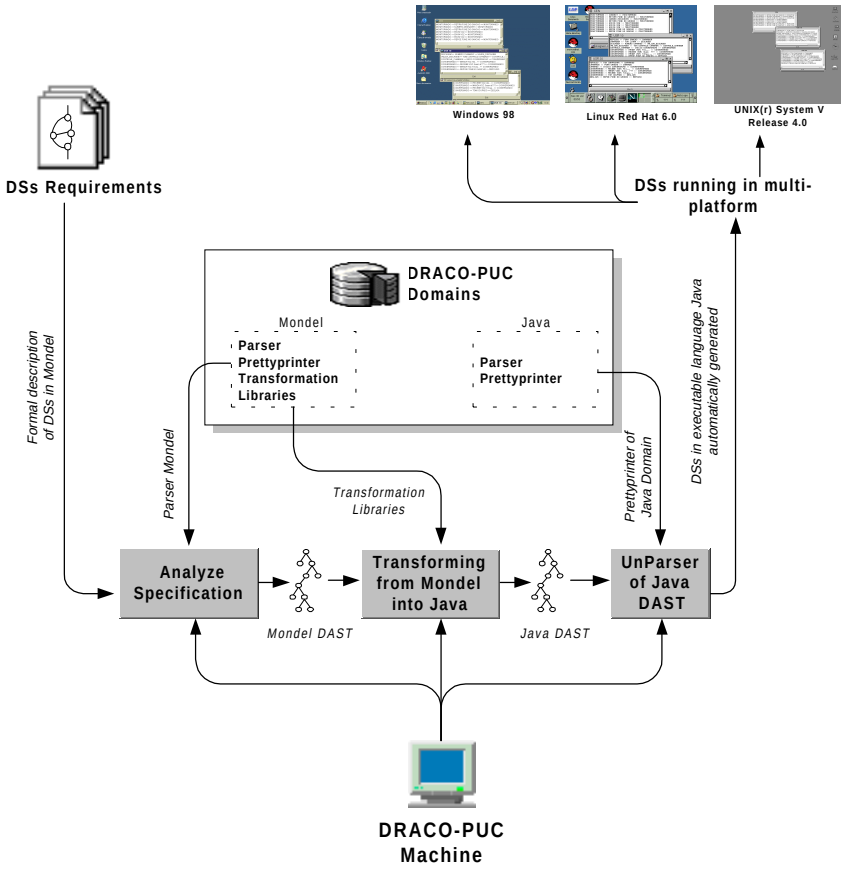

Fig. 3. Model of Automatic Transformation from Mondel to Java. The figure presents the principal steps taken to use the Mondel and Java Domains in the automatic implementation of the Mondel specifications into Java, using the Draco machine.

In **Analyze Specification**, the Draco Machine using the Mondel parser analyzes the Mondel specification. The output of this activity is the code in the internal representation; the DAST Mondel guided by the concrete Mondel syntax thus obtaining the syntactically correct Mondel specification.

In **Transforming from Mondel into Java**, the syntactically correct Mondel specification is automatically transformed by the Draco machine, which uses the *mondelTojava.tfm* inter-domain transformer. This activity generate a Java DAST guided by the concrete Java syntax what is the entry to the **Unparser of Java DAST** which makes textual again - this activity is guided by the Java *PrettyPrinter*.

The Java programs generated automatically are based on Java/RMI structure. The RMI enables the programmer to create distributed Java-to-Java applications, in which the methods of remote Java objects can be invoked from other Java virtual machines, possibly on different hosts.

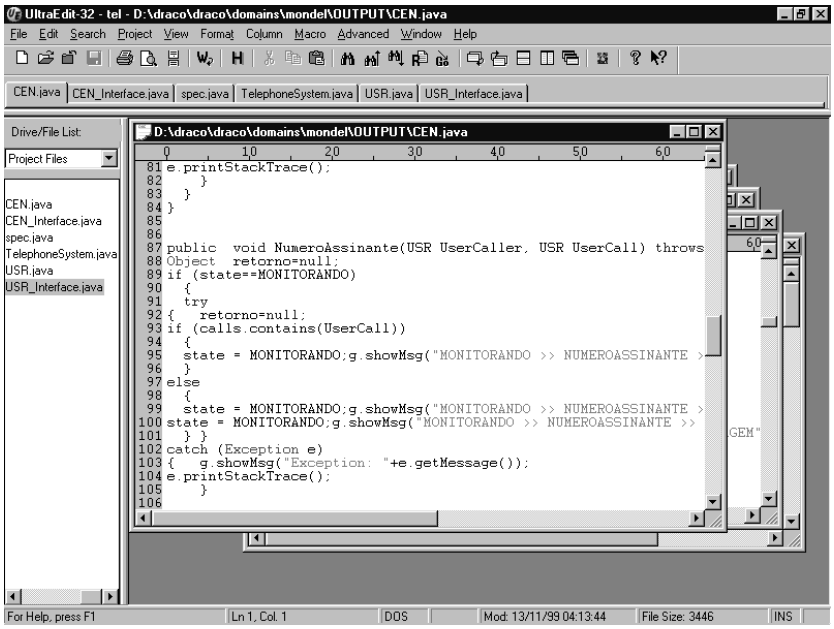

Fig. 4. Part of source-code in Java obtained automatically for the Telephone System. The source-code obtained include the library java.rmi.* to support the objects distributition from Distributed System and it's running in multiplatform.

The major solutions for distributed applications (e.g. CORBA [7] and PVM [10]) rely on a run-time mechanism to supply services such as message passing between remote objects, process creation and remote procedure calls (or remote method invocations). The solutions adopted in this work also consider some kind of run-time support through of the Remote Method Invocation (RMI) Java library .

The methods of an RMI object can send messages to other objects in a remote Java Virtual Machine (JVM), usually using the net, as if it was calling a local object. In that sense, RMI is very similar to CORBA. And like CORBA, RMI allows the clients interact with remote objects by public interfaces. The clients do not interact directly with the classes that implement the interfaces.

Conclusions

FDTs are becoming increasingly important in the development of distributed systems. They are not restricted to the specification phase since they are also used as a reference for the production process phases of these systems. The use of supporting tools is very important to reap the full benefit from an FDT when developing different kinds of distributed systems.

One of the results of this work was the definition of a strategy for the automatic implementation of Mondel specifications using the DRACO-PUC tool. The importance of this strategy lies in the use of a transformation system that allows for the implementation of Mondel to other different Java languages. It is possible, for instance, to define a library of transformations that transform Mondel specifications into Pascal or any language whose domain is defined in the DRACO-PUC tool.

The use of the DRACO-PUC tool in software development is important because it allows for automatic generation of the code based on high level specifications. Maintenance of the DRACO-PUC generated systems is easier, even in Distributed Systems with different protocols, architectures, operational systems and databases, because the designer can perform the task of maintenance at the level of Mondel language specifications.

Although some Mondel development environments already exist that permit semi-automatic generation of the code based on specifications, our contribution consists of the use of an up-to-date technology [8] that can be broadened and improved upon to make the entire automation process feasible for different hardware and software platforms.

Some significant results have already been obtained using the strategy presented herein. The Draco environment has proved its effectiveness in enabling automatic transformation of object-oriented models of Mondel specifications into Java, as shown in the *Telephone System* case (300 source code lines in Mondel specification and 600 source code lines in Java generated automatically). Other case study, the specification of the *ODP Trader Function* [5] (1500 source code lines in Mondel specification and 3000 source code lines in Java generated automatically), was submitted to the library of transformations of the *mondelTojava* transformer. A large library of transformations has already been built that currently allows for the automatic transformation of a large part of the Mondel specifications into Java. Further research work is underway to enlarge this library with the aim of implementing all Mondel specifications.

Future research work includes the following: enlargement of the transformation library to allow for implementation of more complex systems, creation of other libraries for implementation in new languages and operational systems, and the use of other architectures, such as CORBA[7] and DCE [8] to support distributed computation. Another research work, very important, to validate all process of transformations using the DRACO-PUC environment is related to the studies that guaranteed the semantic maintained of the Formal Specification, in the language of the implementation, when applied the transformations.

Others programs transformations systems were compared with DRACO-PUC. The PetDingo tool that transform Estelle specifications to C++ language produce 90% of the process transformation automatization. In DRACO-PUC the rate of the automatization to C++ language and other languages was 100% without anyone manual interference.

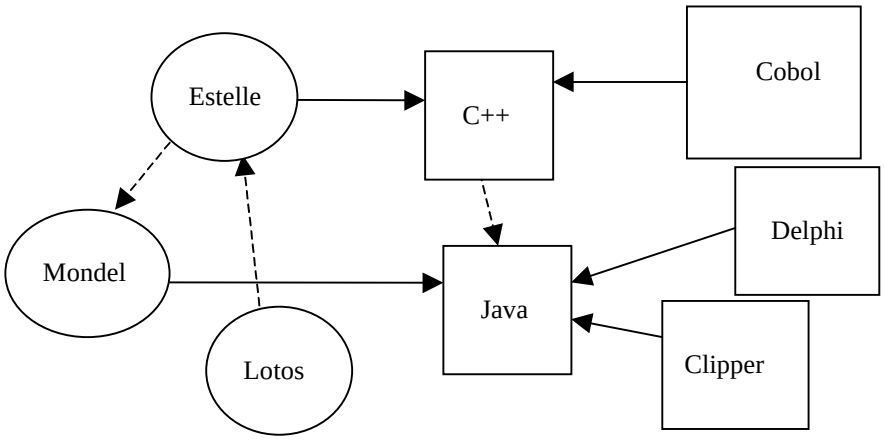

Fig. 5. Part of the Draco Domain Network. In figure some possible transformations are shown. The full lines represent transformation libraries under development or already built. The dashed lines represent transformation libraries to be built.

References

1. BOCHMANN, G. V.: Concepts for Distributed Systems Design. Springer-Verlag, 1983.
2. BOCHMANN, G. v., Poirier, S., Mondain-Monval, P.: Object-Oriented Design for Distributed Systems: The OSI Directory Example, Université de Montréal, 1991.
3. BOCHMANN, G. v., Barbeau, M., Erradi, M., Lecomte, L, Mondain-Monval, P., Willians: An Object-Oriented Specification Language, Université de Montréal, 1990.
4. ISO IS 7498: Information processing systems - Open Systems Interconnection - Basic Reference Model, 1984.
5. ITU-T Draft Rec X.904, ISO/IEC DIS2 13235: ODP Trading Function - Part 1 - Specification, Document approved for standardization, Jan 1997.
6. NEIGHBORS, J.: Software Construction Using Components, Tese de Doutorado, University of California at Irvine (EUA), Set/1980.
7. Object Management Group and X/Open: The Common Object Request Broker: Architecture and Specification, Document Number 91.12.1, 1992.
8. Open Software Foundation: DCE Application Development Guide, Cambridge, MA (EUA), 1992.
9. PRADO, A.F.: Estratégia de Re-Engenharia de Software Orientada a Domínios, Tese de Doutorado, Pontifícia Universidade Católica, Rio de Janeiro, Ago/1992.
10. SOUZA, P.S.Lopes de: O Impacto do Protocolo TCP/IP na Computação Paralela Distribuída no Ambiente Windows95, Proceedings of the 15th Brazilian Symposium on Computer Networks, São Carlos(SP), 19-22/05/1997, pp.118-134.