# CAMILA: Prototyping and Refinement of Constructive Specifications

J. J. Almeida, L. S. Barbosa, F. L. Neves and J. N. Oliveira

Computer Science Department
University of Minho
Largo do Paço — 4710 Braga
Portugal

{jj,lsb,fln,jno}@di.uminho.pt

**Abstract.** This paper accompanies the demonstration of CAMILA, an experimental platform for formal software development, rooted in the tradition of constructive specification methods. The CAMILA approach is an attempt to make available at software development level the basic problem solving strategy one got used to from school physics — *create, experiment and reason on a mathematical model.* Based on a notion of *formal software component*, it encompasses a set-theoretic language and an inequational *calculus* for classification and refinement. Its kernel is a functional prototyping environment, fully connectable to external applications, equipped with a classified component repository and distribution facilities.

*Keywords* Constructive specification, prototyping, program calculation.

## 1 An Overview

In the structure of an information system, Software Engineering distinguishes between *entities*, which represent information sources, and *transformations* upon them. The former originates the data structures, the later the algorithms. The fundamental observation behind *constructive* (or *model-oriented*) specification methods such as VDM [Jon86], Z [Spi89] or RAISE [Geo91], is that a similar duality appears in the definition of an algebra (sets and functions) or relational structure (sets and relations), making them suitable as a mathematical semantics for such systems.

CAMILA [1] development affiliates itself to the research in *constructive specification methods* as well as in exploring Functional Programming as a *rapid prototyping* environment for software models, a program whose origin can be traced back to Peter Hendersen's me too [Hen84]. The project main contributions may be summarized as follows:

[1] CAMILA is named after a Portuguese 19th-century novelist — Camilo Castelo-Branco (1825 - 1890) — whose immense and heterogeneous writings, deeply rooted in his own time controversies, mirrors a passionate yet difficult life.

1. As a (executable) specification language, CAMILA takes full advantage of the (Cartesian closed) structure of the category Set of sets and set-theoretical functions. In particular data sorts are regarded as endofunctors [2] over Set, transforming uniformly either sets and functions. This mirrors the intuition that (parametric) sorts are (some kinds of) functors and morphisms arise as natural transformations between them.
2. Traditionally, in the constructive style for software development design is factored in as many "mind-sized" steps as required. Every intermediate design is first proposed and then proved to follow from its antecedent. Such an "invent-and-verify" style is often impractical due to the complexity of the mathematical reasoning involved in real-life software problems. At the core of the CAMILA approach is a *calculus* — named SETS, after *Specification in* Set [Oli90,Oli92,Oli97] — which introduces explicit transformational rules in program data structuring. Here an intermediate design is drawn from a previous one according to some *law* available in the *calculus*, which is structural in the sense that model components can be refined in isolation (and, consequently, previous refinement results re-used). Proof discharge is achieved by replacing proofs from first principles by calculation. This is the point of a calculus, as witnessed elsewhere in the history of mathematics, and corresponds to a maturation stage emerging naturally after two decades of intensive research on the foundations of formal methods for software design.
3. CAMILA is particularly oriented to the development of systems involving some degree of distribution. Facilities are provided to prototype software components as communicating agents.

Last but not least, a major goal in CAMILA design has been to scale up formal development methods to the industrial practice. This has shaped the project as a collection of portable working tools, with simple interfaces but easily connectable to external applications (*e.g.*, databases, interface generators, document processors, etc.). Some real *case studies* include the development of reuse mechanisms for a CASE tool [Oli95b], the project of temporal databases and the design of a *building description language* [Oli95a]. The following section details the four main components of the CAMILA platform.

## 2 A "Guided Tour"

### 2.1 The Functional Prototyper

The *Functional Prototyper* animates Set as a mathematical space for building specifications as collections of parameterized data sorts and functions upon them.

The basic set constructors capture essential operations upon information:

[2] More rigorously, as combinations of polynomial functors plus the direct powerset functor $\mathbf{2}^{-}$.

- *Cartesian product* ($A \times B$) for aggregation in the spatial axis,
- *coproduct* ($A + B$), for choice (i.e., aggregation in the temporal axis) and
- *exponentiation*, or function space, ($A^B$) for functional dependence.

When processing data definitions, the prototyper generates automatically the constructors and selectors of each product type as well as the canonical embeddings associated to coproducts. Derived constructors, for finite $A$ and $B$, are $\mathbf{2}^A$ (*finite subsets*), $A \hookrightarrow C$ (*finite mappings*), defined as $\sum_{K \subseteq A} C^K$, and $A^*$ (*finite sequences*), defined as $\sum_{n \subseteq \mathbb{N}} A^n$, as well as recursive definitions in the form $X = \mathcal{F}(X)$, where $\mathcal{F}$ is a functor involving the above constructs.

By functoriality, those constructs also act upon functions (either primitive or user-defined) lifting its effect to the generated structure. For example, the expressions `(f-set)-seq` and `(f-seq)-set` correspond, respectively, to the action upon the function $f : A \longrightarrow B$ of the functors $(\mathbf{2}^{\texttt{-}})^*$ and $\mathbf{2}^{(\texttt{-}^*)}$. These enables a particularly fruitful *modular calculus* in which *enrichment* and *specialization* amount to composition with suitable functors, respectively, on the right and on the left.

The operator repertoire in CAMILA is very rich and highly structured. Each operator is an arrow in Set, either *set-theoretic* (*e.g.*, $\mathbb{N} \stackrel{\lambda x.x^2}{\longrightarrow} \mathbb{N}$), or *category-theoretic*. The last group includes arrows classified as *implicit* (*e.g.*, $A \times B \stackrel{\pi_1}{\longrightarrow} A$), *functorial*, *i.e.* the action of some functor on another arrow (*e.g.*, $A \hookrightarrow X \stackrel{A \hookrightarrow f}{\longrightarrow} A \hookrightarrow Y$), *universal*, *i.e.* arising as the unique arrow in an universal construction (*e.g.*, $A + B \stackrel{[f,g]}{\longrightarrow} C$ or $C \stackrel{<f,g>}{\longrightarrow} A \times B$) or *natural*, *i.e.* regarded as a component of a natural transformation (*e.g.*, $X^* \stackrel{\text{elems}}{\longrightarrow} \mathbf{2}^X$). The SETS constructs, as well as the basic algebras associated with them, are directly expressible in the prototyper, from which a high level description is automatically generated (in the form of a LaTeX file). So are the propositional connectives and quantifiers, anonymous function definition, in the form of $\lambda$-expressions, and high-order functions.

Specifying in CAMILA is done in a stepwise-elaboration style, each stage of the model being immediately prototyped and quick feedback about its behavior being gathered within the design team. The prototyper type-checking filters primary syntactic errors and unexpected semantic behavior is likely to be spot and corrected. The tool is able to handle *lazy evaluation* and *partially defined functions*, i.e. functions whose signature has been declared but whose computation rule has not yet been supplied.

### 2.2 The SETS Animator

The reification phase, in the CAMILA life-cycle, is a systematic process of stepwise transformation of the original specification into another which can be eventually recognized as the formal semantics of a particular command, or program fragment, in the intended target technology. The purpose of the SETS *Animator* is to animate the calculus so that concrete data-structures

modeling the specification sorts can be found by calculation, accompanied by the synthesis of abstraction functions and induced implementation-level invariants.

Different laws of SETS lead to different implementation structures and platforms. A typical example of a common target technology is that of relational databases, typically materialized by a particular SQL server. A database table is trivially formalized, in the SETS notation, by a relation in $\mathbf{2}^{A \times B}$ or a mapping in $A \hookrightarrow B$, for $A$,$B$ arbitrary products of "atomic" types. Therefore, all SETS laws which somehow "lead" to such structures are welcome by such a target environment [3]. A SETS law is an (in)equation of the form $A \trianglelefteq_f^{\phi} B$, stating that every instance of $A$ can be reified into the corresponding instance of $B$, by adopting abstraction function $f$ and provided that concrete invariant $\phi$ is enforced over $B$. On the whole, the following abstraction invariant, using the terminology of [Mor90], is synthesized: $\lambda ab \,.\, (a = f(b)) \wedge \phi(b)$. For instance, law

$$A \hookrightarrow D \times (B \hookrightarrow C) \trianglelefteq_{\bowtie_n}^{dpi} (A \hookrightarrow D) \times ((A \times B) \hookrightarrow C)$$

states that finite mapping nesting of can be flattened. Repeated application of this law makes it possible to boil arbitrarily nested, intricate mapping-based data structures, down to products of atomic relation tables. The relevant abstraction function ($\bowtie_n$) computes a kind of "nested join" and invariant *dpi* will guarantee that such a join operation is effectively computed.

In the balance *data vs algorithms*, CAMILA is strongly oriented to the data component as the refinement of data sorts induces by itself the *simulations* of the abstract operations in the reified context. Calculating a simulation for operation $\alpha$ amounts to prove (constructively) the commutativity of the refinement diagram for $\alpha$. CAMILA does not claim any originality in this process, relying instead in other calculi such us the FOLD-UNFOLD [Dar82] or the Oxford REFINEMENT CALCULUS [Mor90] [4].

The animator is implemented with genetic algorithms [Mic94] which support a flexible, and surprisingly efficient, inequational term-rewriting. The "genetic engine" represents each potential solution as a chromosome of integer genes, whereas the size of each chromosome fixes the number of refinement steps to be applied. What is interesting is that, by modifying the evaluation function over SETS expressions, the user is able to bias the animator towards the target implementation technology [NO95].

---

[3] Should the target programming language be, for instance, C, then laws leading to structures of the $1 + A$ pattern (the "pointer to $A$ abstraction") will become relevant, in particular recursive structures of the $X = 1 + \mathcal{F}(X)$ shape.

[4] In fact SETS and the REFINEMENT CALCULUS blend together nicely. The last is a weakest pre-condition (algorithmic) calculus in which change of representation is handed by choosing coupling invariants. SETS main purpose is that of *calculating*, rather than choosing, such coupling invariants.

### 2.3 The Repository

The CAMILA platform includes a *reusable components repository* which catalogues the available specifications as well as its implementations and the associated refinement calculations. In particular abstraction functions and induced invariants are recorded as CAMILA expressions allowing for the dynamic interconnection between different levels of abstraction.

The SETS *Animator* is used to classify and compare components (in a "classify-by-data" style). Furthermore architectural relationships such as *is-a*, *is-used-by*, *is-implementation-of*, *is-special-case-of* are formally decided and recorded, rather than fixed by intuition. In the *Repository* component aggregation can be expressed by "software-circuit" diagrams using a graphical notation suggestively resembling the conventional hardware design notation.

The *Repository* and the *Functional Prototyper* bears "full citizenship" at C/C++ programming level. A collection of libraries, enable both the processes of

- embedding CAMILA prototypes in (partial) implementations;
- enriching, either static or dynamically, the prototyping environment with modules supplied or already implemented in the target programming environments.

This leads to what may be called a "hybrid" prototype: some system components already fully reified may cohabit (and communicate) with other parts still in a prototyping phase. Such temporary configurations of the system, which may require abstraction functions explicit at run time, cannot be expected to be particularly efficient ones. But they provide for smooth, stepwise reification and testing.

### 2.4 The Process Prototyper

The specification of distributed systems is not only concerned with the components functionality, but also with the local and global *behavior patterns*, which entails the need to combine set-based specifications with some sort of process calculi over the set of declared operations.

The *Process Prototyper* allows for the annotation of CAMILA components with *behavioral patterns*, which can be simulated and further analyzed by a typical process algebra tool, like the CWB [MS96]. It also includes a small configuration language enabling the (eventually dynamic) association of each node of the system to an independent process (*e.g.*, a UNIX process), communication being achieved through a set of specific primitives. Application dependent constraints of the communication network are themselves prototyped as another system component.

On going work in this area includes the development of a calculus for process refinement dualising SETS results to a category of transition systems.

The idea is that, as specifications of functionality may be regarded as algebras of some Set functors, similarly behaviors emerge as co-algebras of other functors [Bar98]. CAMILA is expected to cope with this broader notion of refinement in the near future.

**Acknowledgments**

The CAMILA project has been supported by the JNICT council under PMCT contract nr. 169/90.

## References

[Bar98] L. S. Barbosa. *Reification of Processes.* PhD thesis, Universidade do Minho (to appear), 1998.

[Dar82] J. Darlington. Program transformation. In *Funct. Prog. and Its Applications: An Advanced Course.* Cambridge Univ. Press, 1982.

[Geo91] C. George. The RAISE specification language: a tutorial. In *Proc. of VDM'91.* LNCS (551), 1991.

[Hen84] P. Henderson. me too: A language for software specification and model building. Preliminary Report, University of Stirling, 1984.

[Jon86] Cliff B. Jones. *Systematic Software Development Using* VDM. Series in Computer Science. Prentice-Hall International, 1986.

[Mic94] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, 1994. Second, Extended Edition.

[Mor90] C. Morgan. *Programming from Specification.* Series in Computer Science. Prentice-Hall International, 1990. C. A. R. Hoare, series editor.

[MS96] F. Moller and P. Stevens. The edinburgh concurrency workbench (version 7). User's manual, LFCS, Edinburgh University, 1996.

[NO95] F. Luís Neves and José N. Oliveira. Software Reuse by Model Reification. *Seventh Annual Workshop on Software Reuse*, August 1995.

[Oli90] J. N. Oliveira. A reification calculus for model-oriented software specification. *Formal Aspects of Computing*, 2(1):1–23, 1990.

[Oli92] J. N. Oliveira. Software reification using the SETS calculus. In *Proc. of the BCS FACS 5th Refinement Workshop, Theory and Practice of Formal Software Development, London, UK*, pages 140–171. Springer-Verlag, 8–10 January 1992. (Invited paper).

[Oli95a] J. N. Oliveira. Formal specification and prototyping of a building description language. In *Proc. CIVIL-COMP'95, Cambridge*, August 1995.

[Oli95b] J. N. Oliveira. Fuzzy object comparasion and its application to a self-adaptable query mechanism. In *Proc. IFSA'95, S. Paulo*, July 1995.

[Oli97] J. N. Oliveira. Sets: A data structuring calculus and its application to program development. Technical Report Lecture Notes for the Macau Course, UNU/IIST, May 1997.

[Spi89] J. M. Spivey. *The Z Notation: A Reference Manual.* Series in Computer Science. Prentice-Hall International, 1989.