On the Development of CAMILA

J. J. Almeida, L. S. Barbosa, J. B. Barros and F. L. Neves

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

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

Abstract. CAMILA is a research project on constructive specification methods and reification calculi which resorts to the Functional Programming technology to provide a *rapid prototyping* environment for software models. This paper provides a brief overview of the CAMILA platform and discusses some of its current extensions.

1 Introduction

CAMILA [1] [2] [ABNO97a,ABNO97b] is an experimental platform for formal software development, oriented towards model-oriented specification methods. A major goal in its 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, document processing [RAH95] and the design of a *building description language* [Oli95a].

The CAMILA project has now reached a maturity stage which calls for a deep reflection on its evolution as well as on the pragmatics of its introduction on industry. This paper provides a brief overview of the system and discusses some of its current extensions.

2 Specification

As an (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 combinations of regular functors

[1] CAMILA has been developed by JNICT consortium 169/90 (INESC Group 2361 hosted by the University of Minho, Braga, Portugal, and the Computer Science Department of the same University) by a team led by Prof. José N. Oliveira. Further information is available at `http://camila.di.uminho.pt/`.

[2] CAMILA is named after a Portuguese 19^{th}-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.

plus the direct powerset functor $\mathcal{P}(\_)$, 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.

Its kernel is a *Functional Prototyper* which 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:

- *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 $\mathcal{P}(A)$ (*finite subsets*), $A \rightharpoonup C$ (*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 $\mathcal{P}(\_)^*$ and $\mathcal{P}(\_^*)$. 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.*, $\mathbb{N} \xrightarrow{\lambda x . x^2} \mathbb{N}$), or *category-theoretic*. The last group includes arrows classified as *implicit* (*e.g.*, $A \times B \xrightarrow{\pi_1} A$), *functorial*, *i.e.* the action of some functor on another arrow (*e.g.*, $A \rightharpoonup X \xrightarrow{A \rightharpoonup f} A \rightharpoonup Y$), *universal*, *i.e.* arising as the unique arrow in an universal construction (*e.g.*, $A + B \xrightarrow{[f,g]} C$ or $C \xrightarrow{<f,g>} A \times B$) or *natural*, *i.e.* regarded as a component of a natural transformation (*e.g.*, $X^* \xrightarrow{\text{elems}} \mathcal{P}(X)$). The Set 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 λ-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 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.

Camila is particularly oriented to the development of systems involving some degree of distribution. Facilities are provided to prototype software components as communicating agents [BA97].

3 Refinement

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.

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 platform includes an *Animator* of the SETS calculus whose purpose is to help in the process of finding by calculation concrete data-structures modeling the specification sorts, 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 $\mathcal{P}(A \times B)$ or a mapping in $A \rightharpoonup B$, for A and 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 ϕ 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

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

[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. Besides the relational database paradigm, including a re-interpretation of *normalization theory* as a subset of the SETS theory [Rod93], *Hash tables*, or, more generically, "fractal" types, are a class of implementation devices which has been carefully calculated in SETS [Oli98].

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. SETS is pregnant of useful laws for data reification. Among these, laws which, like the one above, "push the $\times$ construct outwards" make for *horizontal refinement* [Gog86] and data oriented distribution [OBM98] in a natural way.

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 α amounts to prove (constructively) the commutativity of the refinement diagram for α. 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 SETS 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,NO98].

4 Some Questions, Some Extensions

4.1 Prototyping *vs* Programming

CAMILA 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].

Certainly there is a clear difference between a specification and a program. There is also a shift of perspective between the formal methods and the functional programming communities that can be illustrated by the kind of data modeling primitive selected as "first choice" by each of them: a *finite mapping* in the the former, a *tree-like structure* in the later.

However, modern specification and programming languages are getting closer all the time. The decrease in the gap between these two worlds has risen the question whether there still exists a place for using a purpose-built, instead of a general-purpose language in the writing of prototypes. From our experience

[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.

with CAMILA we believe that there will continue to be a place for such systems but this debate should be maturated and better documented.

A point to be taken into consideration is the kind of support available to the software development, in particular the control of the runtime coexistence between different levels of abstraction. 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 the different levels mentioned above.

Both this *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. The relationship between this sort of *prototyping embedding* in the CAMILA methodology, and the concept of staged or meta-programming [She98] is under study.

4.2 (Re-)Designing the Type System

When compared with the majority of type-systems used in programming languages, CAMILA's type-system is quite cumbersome for it provides a vast amount of basic type constructions. In particular it allows the definition of generic constructions abstracted with respect to a (large) class of data types.

Even though this large number of constructors gives rise to an increased expressive power, it also makes it difficult to devise efficient and useful type-checking mechanisms.

On the other hand CAMILA's formal notation has been studied with the view to devise a metrics for component classification, so that architectural relationships in the *Repository* such as is-a, is-used-by, is-implementation-of, is-special-case-of are formally decided rather than fixed by the users intuition. The associated SETS calculus is used as a means for classify and compare software components and, by unification up to isomorphism, as a base for the reuse mechanism [OC93]. Some mathematical difficulties are still to be solved concerning non monotonic inheritance associated to component classification.

The revision of CAMILA's type-system is currently under study, including an appropriate notion of subtyping.

4.3 (Re-)Implementing the Prototyping Kernel

The back-end language in which CAMILA is presently implemented is xmetoo. This language was developed at Universidade do Minho and uses a part of XLISP: mainly the reduction engine and some memory management routines. There were two main reasons to choose xmetoo as the back-end language:

- The availability of the source code made it extremely easy to add new features and functionalities to the system. This has proved to be a major point, in particular during the design stage of CAMILA.
- The simplicity of this system and its availability for a wide range of platforms.

The technology with which xmetoo was developed is now 10 years old. This has serious disadvantages, namely inefficient memory management, very limited string operations, poor implementation of lazy evaluation, and absence of any kind of type-checking mechanisms

Current work in the project includes the development of two alternative implementations of the CAMILA kernel using modern, standard, and widely-used programming languages as back-end – HASKELL and `perl`.

The choice of HASKELL, a non-strict, general purpose, purely functional and widely available language, is quite obvious. It was motivated, in particular, by

- HASKELL adoption as the standard functional programming language for educational purposes in many universities around the world;
- availability of a large collection of tools, libraries, and modules that could be incorporated in the writing of CAMILA specifications;
- neat implementation of monads, a concept that seems quite appropriate in the design of a specification language based on a notion of state.
- availability of efficient and up-to-date implementations, thus providing easily maintainable programs;
- non-strictness, enabling the definition of infinite structures and thus offering an alternative implementation for networks of prototype's processes, a feature that is achieved in an imperative way in CAMILA's current release.

On the other hand, `perl` is a language for processing text which incorporates sophisticated pattern matching capabilities, straightforward I/O, and a flexible syntax. By borrowing heavily from C, `sed`, `awk`, and the UNIX shells, `perl` has become the language of choice for many I/O, file processing, process management and system administration tasks.

Availability of interfaces with the majority of programming environments, pre-compilation mechanisms and a large collection of tools, paves the way to the efficient implementation of CAMILA development strategies, namely prototype embedding at different levels of abstraction. Furthermore, `perl` provides robust memory management mechanisms (required *e.g.* to run application prototypes on real data, which involve dealing with very large data structures). In the area of document prototyping [RAH95] `perl` powerful text processing libraries may become quite relevant.

4.4 Integrating with a Proof Assistant

Another major extension to the CAMILA platform is its integration with the COQ proof assistant [CCF+95], which is being done in the context of the LOGCOMP project. In the overall the project aims to integrate, at both the conceptual and implementation levels, Proof Assistant systems with computational systems for symbolic manipulation.

By a Proof Assistant should be understood, in this context, a generic theorem prover supporting implementations of type theory, such as COQ, ALF [AGNS94] or ISABELLE [PN90]. In a sense the success of such systems rely on their capacity to represent the most general programming paradigm: the one based on an inference system for an almost arbitrary logic.

Of course, such flexibility presents some drawbacks which manifest themselves with respect to efficiency and the lack of a user population with enough expertise in the technicalities of the methodologies used. They can be overcome, however, when combined with systems, like the ones used in the formal methods community. In fact, such systems, being more oriented towards a particular domain or class of problems, are supposed to possess efficient implementations and to be familiar to a broad range of users.

From the CAMILA point view, on the other hand, the incorporation of such a Proof Assistant will provide for a semi-automatic proof environment embedded in the platform. This can be used, not only to carry on some of the classical proof obligations (*e.g.* invariant preservation), but also to produce "explanations" about the CAMILA *Animator* behaviour in a semi-automatic way, therefore documenting the calculations performed.

SETS relies on an categorial view of set theory and is mainly concerned with *calculation* within it. An implementation of this approach to set theory in COQ is being developed. Well-known implementations of set theory in proof systems, such as [Pau93] or [CNSS94] follow closely the axiomatic view and are somewhat poor with respect to calculation skills. On top of it a codification of several properties related to the certification of a generic CAMILA model (*e.g.* invariant preservation and propagation along a refinement chain) will form the base of this major extension of CAMILA.

Acknowledgments

Current research on the CAMILA platform is funded by the LOGCOMP project under PRAXIS XXI contract nr. 2/2.1/TIT/1658/95.

References

[ABNO97a] J. J. Almeida, L. S. Barbosa, F. L. Neves, and J. N. Oliveira. CAMILA: Formal software engineering supported by functional programming. In A. De Giusti, J. Diaz, and P. Pesado, editors, *Proc. II Conf. Latino Americana de Programacin Funcional (CLaPF97)*, pages 1343–1358, La Plata, Argentina, October 1997.

[ABNO97b] J. J. Almeida, L. S. Barbosa, F. L. Neves, and J. N. Oliveira. CAMILA: Prototyping and refinement of constructive specifications. In M. Johnson, editor, *6th International Conference on Algebraic Methods and Software Technology (AMAST)*, pages 554–559, Sydney, Australia, December 1997. Springer Lect. Notes Comp. Sci. (1349).

[AGNS94] T Altenkirch, V. Gaspes, B. Nordstrom, and B. Sydow. A user's guide to ALF. Technical report, Chalmers University of Technology, Sweden, 1994.

[BA97] L. S. Barbosa and J. J. Almeida. Systems prototyping in CAMILA. Lecture Notes for the Bristol Course (1st ed. 1995) DI-CAM-97:11:1, DI (U. Minho), 1997.

[CCF+95] C. Cornes, J. Courant, J.-C. Filliâtre, G. Huet, P. Manoury, C. Muños, C. Murthy, C. Parent, C. Paulin-Mohring, A. Saibi, and B. Werner. The COQ proof assistant reference manual version 5.10. Rapport Technique 0177, INRIA Rocquencourt-CNRS-ENS Lyon, July 1995.

[CNSS94] T. Coquand, B. Nordstrom, J. M. Smith, and B. Sydow. Type theory and programming. TR 81, Programming Methodology Group, University of Goteborg and Chalmers University, 1994.

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

[Gog86] J. A. Goguen. Reusing and interconnecting software components. *IEEE Computer*, 19(2):16–28, 1986.

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

[Mic94] Z. 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.

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

[NO98] F. L. Neves and J. N. Oliveira. ART - um laboratório de reificação "genética". In *Proc. IBERAMIA'98 — Sixth Ibero-Conference on Artificial Intelligence*, Lisbon, Portugal, October 5-9 1998.

[OBM98] J N. Oliveira, L. S. Barbosa, and F. S. Moura. Can distribution be (statically) calculated? (in preparation), DI (U. Minho), 1998.

[OC93] J N. Oliveira and M. Cruz. Formal calculi applied to software component knowledge elicitation. Technical report c19-wp2d, project c.1.9. sviluppo di metodologie, sistemi e servizi innovativi in rete, INESC, 1993.

[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, UNU/IIST, May 1997. Lecture Notes of Course at UNU/IIST, Macau, 5-16 May 1997, 125p.

[Oli98] J. N. Oliveira. Fractal types: An attempt to generalise hash table calculation. In T. Sheard R. Backhouse, editor, *Proc. Workshop on Generic Programming*, Marstrand, Sweden, June 1998.

[Pau93] L. C. Paulson. Set theory for verification: I. from foundations to functions. *Jour. of Automated Reasoning*, 11(3):353–389, 1993.

[PN90] L. C. Paulson and T. Nipkow. ISABELLE tutorial and user's manual. Technical report, TR 189, University of Cambridge, 1990.

[RAH95] J.C. Ramalho, J.J. Almeida, and P.R. Henriques. Algebraic specification of documents. In A. Nijholt, G. Scollo, and R. Steetskamp, editors, *Algebraic Methods in Language Processing*. Twente University, Netherlands, 6–8 Dec. 1995. TWLT 10, AMiLP'95 to apear on TCS.

[Rod93] C. J. Rodrigues. Sobre o desenvolvimento formal de bases de dados. Master's thesis, University of Minho, 1993. (In Portuguese).

[She98] T. Sheard. Reflective metaprogramming systems as environments for generic programming. In T. Sheard R. Backhouse, editor, *Proc. Workshop on Generic Programming*, Marstrand, Sweden, June 1998.