Architectural Specifications in CASL*

Michel Bidoit¹, Donald Sannella² and Andrzej Tarlecki³

¹ Laboratoire Spécification et Vérification, CNRS & ENS de Cachan, France
² Laboratory for Foundations of Computer Science, University of Edinburgh, UK
³ Institute of Informatics, Warsaw University and Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland

Abstract. One of the most novel features of Casl, the Common Algebraic Specification Language, is the provision of so-called architectural specifications for describing the modular structure of software systems. A brief discussion of refinement of Casl specifications provides the setting for a presentation of the rationale behind architectural specifications. This is followed by some details of the features provided in Casl for architectural specifications, hints concerning their semantics, and simple results justifying their usefulness in the development process.

Keywords: Algebraic specification; Architectural specifications; CASL; Formal software development; Specification of program structure

1. Introduction

A common feature of present-day algebraic specification languages (see e.g. [SaW83, EhM85, GuH93, CFI96, SaW99]) is the provision of specification-building operations for building large specifications in a structured fashion from smaller and simpler ones. This was pioneered by Burstall and Goguen in their seminal work on the Clear specification language [BuG77, BuG80]. Less usual in specification languages are features for describing the modular structure of software systems under development. This paper is about the facilities for this that are provided in Casl, the new Common Algebraic Specification Language [ABK03, CFI01b] that has been developed under the auspices of the Common Framework Initiative [Mos97, CFI01a] in an attempt to create a focal point for future joint work on algebraic specifications and a platform for exploitation of past and present work on methodology, support tools, etc.

Following earlier practical experiences [FiJ90, FAC92] and foundational work [Bid88, SaT89, SST92, BiH93], we argue that mechanisms for structuring specifications are not the same as and cannot suffice for describing the modular structure of software under development. Casl, therefore provides a separate kind of specifications, so-called architectural specifications, for this purpose. An architectural specification consists of a list of unit declarations, indicating the component modules required with specifications for each of them, together with a unit term that describes the way in which these modules are to be combined. Such architectural

*This is a revised and expanded version of Bidoit et al. [BST99].
Architectural Specifications in Casl

Architectural specifications are aimed at the ‘implementation’ modular structure of the system, much in the style of the features for programming in the large using generic modules in modern programming languages like Standard ML [Pau96], cf. Burstall and Lampson’s Pebble [BuL88]. This is related to but is not the same as software architecture in the sense of [AlG97] which deals with the organization of software into components and connectors, focusing on the ‘interaction’ relationships between modules (the latter to be considered when specifications of ‘reactive’ modules are introduced in a Casl extension, cf. [FiL97]).

The aim of this paper is to present motivation, intuition and technicalities related to this concept. We provide some basic information about Casl in Section 2, discuss the development of programs from specifications by stepwise refinement in Section 3 and then introduce architectural specifications in Section 4. We stress there how generic components arise naturally from the desire to allow separate but related modules to be developed independently. The semantics and correctness aspects of architectural specifications with the simplest ways of combining modules are discussed in Sections 5, 6 and 7. Further operators for combining modules are presented in Section 8. The development process in the presence of architectural specifications is briefly discussed in Section 9.

Architectural specifications are presented in the context of Casl. However, the overall ideas if not all the technicalities are applicable in any specification and development framework, as we explain in Section 10. We also venture there briefly into more advanced features of architectural specification and development, bringing in ideas of behavioural refinement.

2. Casl Preliminaries

Casl is a formalism to describe Casl structures: many-sorted algebras with subsorts, partial operations and predicates. Structures are classified by signatures, which give sort names (with their subsorting relation), partial/total operation names, and predicate names, together with profiles of operations and predicates. In Casl structures, subsorts and supersorts are linked by implicit subsort embeddings required to compose with each other and to be compatible with operations and predicates with the same names. For each signature Σ, the class of all Σ-structures is denoted \( \text{Mod}[\Sigma] \).

The basic level of Casl includes declarations to introduce components of signatures and axioms to give properties of structures that are to be considered as models of a specification. The logic used to write the axioms is essentially first-order logic (thus, with quantification and the usual logical connectives) built over atomic formulae which include strong and existential equalities, definedness formulae and predicate applications, with generation constraints added as special, non-first-order sentences. A basic Casl specification \( SP \) amounts to a definition of a signature Σ and a set of axioms \( \Phi \). It denotes the class \( \text{[}[SP]\text{]} \subseteq \text{Mod}[\Sigma] \) of its models, which are those Σ-structures that satisfy all the axioms in \( \Phi \): \( \text{[}[SP]\text{]} = \{ A \in \text{Mod}[\Sigma] \mid A \models \Phi \} \).

Apart from basic specifications as above, Casl provides ways of building complex specifications out of simpler ones by means of various structuring constructs. These include translation, hiding, union, and both free and loose forms of extension. Generic specifications and their instantiations with pushout-style semantics [BuG80, EhM85] are also provided. Structured specifications built using these constructs can be given a compositional semantics where each specification \( SP \) determines a signature \( \text{Sig}[SP] \) and a class \( \text{[}[SP]\text{]} \subseteq \text{Mod}[\text{Sig}[SP]] \) of models. We say that \( SP \) is consistent if \( \text{[}[SP]\text{]} \) is non-empty.

2.1. Example

Here is a sequence of definitions of Casl specifications. We intersperse them with comments to clarify the meaning of particular Casl constructs and notations. The example is small but it is not contrived in the sense that the way in which the specifications build upon one another seems quite natural.

```
spec MONOID =
sort Thing
ops null : Thing;
o : Thing × Thing → Thing, assoc, unit null
end
```

This is the usual specification of a monoid with a sort of elements, a constant, and a binary operation that is associative and has the constant as a neutral element.