Specification language

A **specification language** is a formal language in computer science used during systems analysis, requirements analysis, and systems design to describe a system at a much higher level than a programming language, which is used to produce the executable code for a system.[1]

Contents

Overview

Specification languages are generally not directly executed. They are meant to describe the *what*, not the *how*. Indeed, it is considered as an error if a requirement specification is cluttered with unnecessary implementation detail.

A common fundamental assumption of many specification approaches is that programs are modelled as algebraic or model-theoretic structures that include a collection of sets of data values together with functions over those sets. This level of abstraction coincides with the view that the correctness of the input/output behaviour of a program takes precedence over all its other properties.

In the *property-oriented* approach to specification (taken e.g. by CASL), specifications of programs consist mainly of logical axioms, usually in a logical system in which equality has a prominent role, describing the properties that the functions are required to satisfy—often just by their interrelationship. This is in contrast to so-called model-oriented specification in frameworks like VDM and Z, which consist of a simple realization of the required behaviour.

Specifications must be subject to a process of *refinement* (the filling-in of implementation detail) before they can actually be implemented. The result of such a refinement process is an executable algorithm, which is either formulated in a programming language, or in an executable subset of the specification language at hand. For example, Hartmann pipelines, when properly applied, may be considered a dataflow specification which *is* directly executable. Another example is the actor model which has no specific application content and must be *specialized* to be executable.

An important use of specification languages is enabling the creation of proofs of program correctness (*see theorem prover*).

Languages

- Attempto Controlled English[2]
- CASL
- VDM

- Z notation
- TLA+
- LePUS3 (a visual, object-oriented design description language)
- Perfect
- Alloy
- LOTOS
- E-LOTOS
- MML
- Refine Language[3]
- SequenceL
- SMV
- SDL
- B-Method

See also

- Formal specification
- Language-independent specification
- SDL
- Unified Modeling Language

References

1. Joseph Goguen "One, None, A Hundred Thousand Specification Languages" Invited Paper, IFIP Congress 1986 pp 995-1004
2. Fuchs, Norbert E.; Schwertel, Uta; Schwitter, Rolf (1998). "Attempto Controlled English—not just another logic specification language" (ftp://nozdr.ru/biblio/kolxo3/Cs/CsLn/Logic%20Programming%20Synthesis%20and%20Transformation,%208%20conf.,%20LOPSTR'98(LNCS1559,%20Springer,%201990)(ISBN%203540657657)(340s).pdf#page=10) (PDF). *International Workshop on Logic Programming Synthesis and Transformation*. Lecture Notes in Computer Science. **1559**. Springer. pp. 1–20. doi:10.1007/3-540-48958-4_1 (https://doi.org/10.1007%2F3-540-48958-4_1). ISBN 978-3-540-65765-1.
3. Linden, Theodore; Lawrence Markosian (1989). "Transformational Synthesis Using Refine" (https://books.google.com/books?id=iMUfTzVuasUC&q=Simkit+Intellicorp&pg=PA245). In Richer, Mark (ed.). *AI Tools and Techniques*. Ablex. pp. 261–286. ISBN 0-89391-494-0. Retrieved 6 July 2014.