CONCEPTS AND TASKS RELATING TO SYSTEM DESCRIPTION

K. Nygaard

Abstract

Modern organisations for production and administration are becomming networks of people, production equipment and information processing equipment. The design, implementation, operation and modification of information processing systems are essential parts of the development of these organisations.

Direct human interaction with the social and physical environment is to a rapidly increasing extent being substituted by interaction through computing and telecommunication equipment, the interfaces being defined by rather fixed, computer based models of the surrounding reality.

In Norway new laws and negotiated, nationwide agreements between the Trade Unions and the employers specify that:

1. Employees have the right to participate in the development of information processing systems which may affect their employment, their job content and their work environment.

2. Information on such systems should be clearly stated in a language understandable to other than system specialists.

Similar conditions will probably be imposed upon system development work in Denmark and Sweden in the course of the next few years.

These imposed conditions are now beginning to influence, in a very direct sense, both the system development process itself, the tools used, and the research and education process relating to information processing systems.

One important implication is that the process of communicating information about systems must be considered more carefully, and in a wider context than before.

The lecture examines the system concept and the communication process, particularly the part which may be named the system description process. Various categories of system description and system exposure are being discussed.

The need for and role of system description languages oriented towards a range of specified tasks in the development process and the operation of systems is explored, with a number of examples.

Discussion

Professor Page considered that acceptance of the responsibility to train and educate is good; but he was not sure what should be done about workers and students who cannot master the computing technicalities. Professor Nygaard suggested that the less able students and workers could be trained in only those aspects of the computing directly relevant to their jobs, and very well trained in those aspects. The more able workers, who might already be in system design, must be retained to work with computer system design, but this presents problems for which there are no answers yet.

Dr. Lauer observed that making something automatic often implies making yourself or others redundant, so provision is necessary for moving people around in jobs.

Professor Nygaard agreed that this is a problem. The primary motivations of manufacturers to automate are finance and convenience. It is sometimes good to automate jobs in a polluted environment, but manufacturers may use this reason to disguise their true motivation, as only some polluted jobs may reasonably be automated.

Professor Neuhold thought that the training appropriate to use a system is very different from that required to build the system. He wondered who really benefits from retraining and who should be consulted about the content of such training.

Professor Nygaard said that this was a long subject. The retraining of employees affects the jobs done by the programmers and system workers. Since the Data Agreements were between the employers and employees, the programmers resented not being involved. However, in retrospect they have seen the benefits of the Agreements.

Professor Wells pointed out that the resistance of the programmers and systems workers was one aspect of what the agreements were intended to prevent.

An Outline of DELTA, a SIMULA-Inspired Language for System Description.

Abstract

SIMULA I and SIMULA 67 were developed to be, at the same time, system description languages and high-level programming languages and are being used in practice for both purposes. As a system description tool, SIMULA assists in the researcher's development of his own understanding of the system being considered (the "referent system"), and in his communication with other researchers or other people concerned with the system.

SIMULA has, however, a number of shortcomings as a system description language, because it also is a programming language.

When we consider systems in our environment, most actions are regarded as time consuming. Changes of state take place continuously, often in a continuous interplay between components. Other actions are regarded as instantaneous (for example, leaving a queue). The computer is a discrete device and has to portray such

time consuming actions involving continuous changes of state by a sequence of instantaneous actions.

Also, it is only to a limited extent, dependent upon the available hardware configuration, possible to portray parallel action sequences.

Description of parallelism and time consuming actions necessitates the use of an interrupt concept which in SIMULA does not exist in a sufficiently powerful version.

The DELTA language is an attempt to generalise the notion of a programming language to create a more comprehensive tool for system description. DELTA was developed in 1973-75 by Petter Handlykken, Erik Holbaek-Hanssen and the lecturer, all employed by the NCC. Since DELTA is not a programming language and cannot be used for instructing computers, its semantics is defined in relation to an "idealised system generator", a generalisation of a computer.

The language has been used in practical description tasks, both in informal, semi-informalised and strictly formal versions.

The lecture will attempt to present some of the basic properties of DELTA and some examples on its use in different situations.

Discussion

Dr. Tanenbaum recalled the need for the systems descriptions to be understood by ordinary workers, and asked whether it was intended that the ordinary worker would eventually understand the system description language DELTA.

Professor Nygaard stated that there is definitely a need for a language such as DELTA, but that experience had shown a great syntactic freedom was required in order that the form could be close to natural language. A more natural appearing language would help avoid activating defence mechanisms within the users, and could be transformed into a more formalised description.

Professor Dijkstra suggested that maybe even natural language is not used accurately enough to enable it to be employed as a tool for system description. The best action was probably to apply teaching methods to overcome any defence mechanisms. In general it was inappropriate to imagine that there could be a good correspondence between a natural language and a formal language, and if it was not possible to communicate a system description in a formal language then maybe no attempt at description should be made.

Professor Nygaard agreed that there were many dangers of misunderstandings when using a natural language for communication, but stressed that it was very important to have some tool for describing and understanding systems. It was hoped that gradually a more formalised notation would become accepted, and that the current, less adequate, tools would no longer be necessary. He agreed with the problems Professor Dijkstra had raised, but felt unable to accept the conclusions.

An Outline of BETA, a DELTA-Inspired Language for Software Construction.

Abstract

In the spring of 1976 it was decided to embark upon a project to develop a high level programming language based upon DELTA and the present "state of the art" in programming research. The participating institutions were: The Department of Computer Science and the Computing Center at the University of Aarhus, Denmark; the Department of Computer Science at the University of Aalborg, Denmark; and the Norwegian Computing Center. This new language, tentatively named GAMMA, was intended to be a useful platform for a possible, later revision of SIMULA 67.

We needed, however, an implementation tool for GAMMA. Soon the development of this tool, a system programming language (or software construction language) named BETA, became an important objective in its own right. The team working on this main partial project consists of Bent Bruun Kristensen, Ole Lehrman Madsen, Birger Moller-Pedersen and the lecturer. The BETA language proposal will be completed this year and implementation projects started by the end of this year.

BETA is intended for use on a wide range of computers. NCC's first implementation is planned for the INTEL 8086 Micro Processor. Typical tasks for the intended use of BETA are: Development and implementation of user oriented languages, experiments with development of new block structured programming languages, operating systems, communication systems and data base systems.

The basic notion of BETA is the notion of a text <u>block</u> and its incarnations in the program executions: the block instances called <u>entities.</u> In the BETA development the emphasis is on the structures generated on the storage media of computing equipment during the execution of a program. Such a structure, generated by the execution of a program written in a language L, is called an <u>L-system.</u>

If the language L is BETA, a BETA-system thus consists of entities, described in the associated program by <u>entity descriptors,</u> being BETA program text blocks. BETA-entities may be either <u>autonomous</u> or <u>constituents</u> of other entities. A BETA-entity is described as <u>singular</u> by

<u>BEGIN</u> entity specification <u>END</u>

or by entity <u>patterns</u>, described by

<u>PATTERN</u> P: <u>BEGIN</u> entity specification <u>END</u>

An autonomous entity is generated and spends its life span as one, integrated whole, like those generated by "<u>NEW</u> C" (C being the title of a class declaration) and the procedure statement "Q" (Q being title of a procedure declaration) in SIMULA.

In BETA autonomous entities may be generated by three distinct constructional modes:

- objects, by the expression "OBJECT P", P being a pattern title or a singular entity descriptor. Objects may develop into stacks with an associated action sequence.

- instances, by the expression "INSTANCE P", P once more being a pattern title or a singular entity descriptor. Instances are members (but never bottom members) of object stacks.

- contexts, by the expression "CONTEXT P", (syntax not yet definitely settled), P being a pattern title or a singular entity descriptor. Contexts provide environments in which BETA-programs are executed.

"OBJECT P" corresponds to "NEW P" in SIMULA. "INSTANCE P" corresponds to the procedure statement "P". "CONTEXT P" has no counterpart in SIMULA, but will provide a generalisation of that language's "system class" concept; for example, the classes SIMSET and SIMULATION.

In contrast to ALGOL 60, SIMULA and most other block structured languages, BETA has only one kind of block specifications - the singular entity specification and pattern declaration being instead used in different constructional modes.

An entity may contain a declaration part consisting of ("prefix", "infix", "insertion" explained below):

- a prefix constituent entity,

- any number of infixed, constituent entities,

- any number of entity pattern declarations,

- any number of references to objects (and possibly to contexts).

An entity may also contain an action part consisting of:

- a sequence of statements,

- among the statements some may be insertions, being constituent entities, analogous to macros in some other languages.

Constituent entities are integral, inseparable parts of other constituent entities and autonomous entities. All constituent enties are specified by constructional modes referring to patterns or singular entity specifications.

If P is a pattern title, then:

- "P BEGIN ..... END" specifies a P prefix entity, with properties similar to those of SIMULA prefixes.

- "X,Y: P" in the declaration part specifies two infixed P entities. Infixed entities are used to obtain variables of the type P or static subroutines described by P.

- "statement 1;
  P BEGIN ..... END;
  statement 2, ..."
  in the action part specifies an inserted singular entity having a P prefix constituent entity.

All constituent entities may be given a name, but only one.

Parameters are implemented as value or result parameters. The virtual concept of SIMULA is extended and provide the tool corresponding to procedure parameters. Repetitions of infixes (corresponding to arrays) and insertions and instances (corresponding to "for-loops") are given a unified treatment. Only few, basic and transparent control structures are provided, since more complex control structures usually are associated with specialised data structues and should be regarded as parts of their definition.

Basic constructs for handling parallel execution of objects are being developed, as well as tools for specifying the hardware and software environment of BETA-programs. Specification of an entity's "interface" with its dynamic environment in a program execution will be developed later (corresponding to, for example, "export" and "import" clauses of other languages).

Since BETA is to be used as a tool for implementing other languages means for linking program constructs in user-defined syntax with BETA-defined semantics will be associated with BETA compilers.

A GAMMA language will be defined when BETA is frozen and DELTA is being revised.

Discussion

Professor Katzenelson observed that the language development work at Delta was related to extensible languages, and asked what extensions would be allowed to operators and data structures through syntax.

Professor Nygaard replied that syntax extensions would be handled by providing a compiler generator to produce the analysis phase of a compiler. This would allow a user to define his own syntax.

Professor Katzenelson then asked whether additional features could be added to a language by the programmers.

Professor Nygaard pointed out that a problem oriented language would be defined for a particular project, with a syntax and semantics appropriate to that project. The defined language would then be implemented through the compiler generator.

Dr. Tanenbaum suggested that it was not appropriate to build a compiler for a particular microprocessor, the Intel 8086 or Zilog Z80, as Professor Nygaard had mentioned during his talk. A portable compiler would be a much better aim.

Professor Nygaard agreed completely, and stated that although the intention was to develop a portable compiler, the first example implementation would be for an Intel 8086. The eventual goal would be to move on to a more useful and economical package.

Professor Pyle said that since the system was to be designed for a microcomputer then the storage required by the run-time nucleus would be a dominant consideration. He enquired whether there were any preliminary ideas or bounds on the size of the run-time nucleus, and whether this would be a design consideration.

Professor Nygaard replied that obviously, in a more generally distributed package, the size would be a consideration. During the development the crucial factor considered was the amount of support required for the execution of programs, and compilation would be allowed to take what it needed. The run-time support required depended very much on the modes of entities within the programs, and the basic techniques and facilities used, but he did not foresee any great problems with this.

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]