Notes on translation problems for micros in education

Problem: Many different dialects of programming languages, many different machines and OSs; want to make production of educational software more efficient and results more uniform

Two basic approaches: (1) constrain form of software developed
- obligation on programmers to produce appropriate software

(2) develop tools to assist translation
- guarantee to handle whatever gets submitted within reason

Suggest: probably need a combination of both approaches: tools implicitly constrain software

Comments on (1):

Far from easy to make standards! Standardising BASIC is hard enough . Possible anyway that many translation difficulties stem from special features of micro used: some types of graphics/interface design cannot be supported on just any micro. What to do about mouse actions where machine has no mouse, for instance?

The only sensible view seems to be that some more abstract level of specification is required: need to be able to specify the I/O behaviour in a way that is independent of the language, machine, type of interfaces to be used. Traditional methods based on eg VDM or Z [Mathai Joseph]. Personal preference lies elsewhere, with a definition based approach, so far untried.

Idea is then that the software must be accompanied by an abstract specification that makes translation to various different contexts routine. Will need a toolkit for interpretation.

Comments on (2):

Unrealistic to expect that can develop methods for arbitrary translation irrespective of the form of the source program + machine configuration. At the same time, doesn't seem that we should be satisfied with a situation in which every translation is ad hoc.

Methods based on machine code translation, or traditional syntax-directed translation are unlikely to offer more than very limited palliatives. To alleviate the problem must achieve similar abstract grasp of the content of a program that is required in (1).

Assume that we can offer a sophisticated environment for supporting the translation process: small team of programmers + reasonably large computing resources + appropriate software tools. The software tool required would create a generic translator: from an abstract model inferred by an expert consultant from a specific program could generate source code for a variety of configurations automatically.

Possible scenario:

Software house prototypes on specific configuration.

Educational software consultant examines the prototype: describes it abstractly in terms of the abstract model supported by the generic translator, or indicates where modification is required to permit such an abstract description.

The abstract description is used as a blueprint from which to develop variants for many conifigurations.

[At this stage believe that initially collaboration between software house and educational software consultant is needed over the development of a particular piece of software; it is unrealistic to think that abstract guidelines for all kinds of software can be specified once and for all.]