COMPUTER AIDED PROGRAMMING EDUCATION (CAPE)

Khalid Abdallah* and Abdallah Al-Sukairi**
*Saudi Aramco, Dhahran, Saudi Arabia
**King Fahd University of Petroleum & Minerals, Dhahran, Saudi Arabia

Computer Aided Education (CAE) system is a computer ABSTRACT. software that aids in the education of a student. Computer-based educational systems are valuable tools to improve learning. It is thus worth endowing programming environments with educational capabilities to help students learn programming.. Program development process has three distinct phases: understand the problem, design the solution, and code the program. Much of the creativity in programming is concentrated in designing the solution rather than implementing it in a selected language. We have developed a model, called Computer Aided Programming Education (CAPE), that supports the program development process. CAPE consists of problem definition and algorithm construction modules. Algorithm construction module is currently supported with a facility to map an algorithm(s) into FORTRAN or Pascal code. CAPE motivates students to study the algorithmic problems operationally and provides a mechanism for rapid prototyping in various programming languages.

INTRODUCTION.1

Computer Aided Education (CAE) system is a computer software that aids in the education of a student [1]. Most educators have become convinced that the computer's role in supporting teaching process is a valid and exciting one. It was recognized very early that educating was so complex that computers could only provide help in teaching rather than "replace" teachers. The system should be designed to teach students how to build abstract models independent of a specific programming language with facilities guiding users to map the model into an effective program in a specific language [2].

Computer-aided Instruction (CAI) is the process of teaching by computer [3]. CAI systems range from book analogies in which a student “turn pages” to systems that use multiple media to offer intelligent interactive assessment. In addition to text, students would like to have sound, pictures, maps, diagrams, animation and humor [4]. Multimedia involves the combination of different representations and offers something new to areas that deal with how ideas are represented and how students explore them. Multimedia design involves the grouping of information resources and the controls needed to manipulate them [5].

Computer Aided Programming Education (CAPE) is a system model that, we believe, aids students and instructors of an introductory course to procedural programming languages. Teaching a programming language is not our main goal; rather we are interested in providing a computerized environment where students are aided in a program development process. We need to teach problem solving and program methodologies that can be used in dealing with almost any procedural programming language. The model is designed for an academic environment where programming concepts are introduced in lectures and programming development process is performed during laboratory sessions. The model is also designed for a general purpose procedural programming language, currently defined to support Pascal and FORTRAN.

CAPE MODEL PHILOSOPHY.2

A program is an algorithm that has been translated into a programming language. We regard a program as a specific implementation of an algorithm. The point is that the algorithm is written first, in an informal (but still precise) English-like language. The program comes later, by a process of translation rather than creation. The hardest part of program development is the planning and design of the program, rather than the translation of that design into the statements of a programming language. Students who have difficulty in programming often believe their problem is unfamiliarity with the language, when actually it is inability to find a suitable design for the program. For our purposes, a program is "correct" if there exists some set of test data for which the program will yield the correct answer. Instructors could apply the problem's test data to students developed solutions and review the output against expected results.

When focus is changed to the coding and debugging of programs, it is advisable to map a skeleton code into an external environment that is intended for that purpose. CAPE provides the required environment to describe the problem and construct the algorithmic solution and then map it externally. There are a number of coding environments that are equipped with facilities to compile, execute, and debug programs written in a selected procedural language.

Problem Definition.2.1

A problem can be described by presenting a sequence of textual or animated steps. Each problem has its own approach that makes it unique to be classified as a programming problem. Classical programming problems include average calculation, sorting numbers, and matrix multiplication. A number of exercises can be generated from a single problem. For example, calculate the average of ten numbers, of *N* numbers, or of numbers delimited by a zero, are exercises of calculating average problem. The essence of the problem is in every exercise with differing parameters. An exercise can have one or more sample input data with their expected results. A sample of input and output always help to understand a problem. An evaluation mechanism to execute a set of programs using the exercise sample's data would allow instructors easily identify incorrect solutions.

Algorithm Construction.2.2

To aid students in the planning and design of a program, we should provide them with a tool where they could construct such plan. Flowcharting is normally used in most cases, but it does not abstract language specific instructions. Flowcharting is not considered as an effective tool to represent algorithms. If we use a procedural programming language to express an algorithm then we are not abstracting the solution. Students can learn more by having an algorithmic environment that separates solution design from implementation issues. CAPE has four command statements: Set, Call, Get, and Put; and three structure statements: Simple, Select, and Repeat. We believe that these statements are sufficient to represent algorithms for introductory level students. The focus is towards using these statements in expressing steps of execution for every algorithm. Instructors can also use it during a lecture to explain algorithms and to have a consistent mechanism in developing programs.

Code Specification 2.2.1

Once the algorithm has been constructed, then comes the coding phase. CAPE is currently designed to map an algorithm or a set of algorithms into Fortran or Pascal code. The way CAPE is structured allows the incorporation of other procedural languages, such as C or PL/I, in the future. We are addressing coding by providing a facility to map an algorithm into a skeleton code of a selected programming language. The generated code is not expected to be free of syntax errors since the specification of algorithm's expressions is not constrained to a specific language. The algorithm designer would follow a valid language specific method in coding expressions or naming variables while inserting statements. Gained benefit is to generate code that is free of syntax errors. There is a requirement that students must be able to practice their coding skills at certain times. CAPE allows the mapping of a selected set of statements as a comment for the student to complete. The derivation of code is totally based on the algorithm specification. An exercise solution might consists of multiple algorithms mapped into one source code. Constructing a solution allows the selection of algorithms to be merged together into an exercise solution coded in a selected language.

CAPE'S OBJECT AND DYNAMIC MODELS.3

The object model is the main frame work around which the design is constructed. We have converted the actions and activities of a dynamic model and the processes of a functional model into operations attached to classes in the object model. CAPE consists of two modules: a Problem Definition module and an Algorithm Construction module. The problem definition module consists of a description component and a set of exercises. The algorithm construction module consists of a block of statements, storage requirements, and a programming language code representation. Figure 1 provides a high level object diagram of CAPE model.

CAPE dynamics are highlighted with its storage declaration and code derivation methods. Over time, the objects stimulate each other, resulting in a series of changes to their states. For example, adding a statement to a block changes the block's state. An individual stimulus from one object to another is an *event*. An event flow diagram summarizes events between objects without regard for sequence. The event flow diagram is a dynamic counterpart to an object diagram. Paths in the object diagram shows possible information flows where paths in the event flow diagram show possible control flows. Figure 2 shows the algorithm's dynamic model.

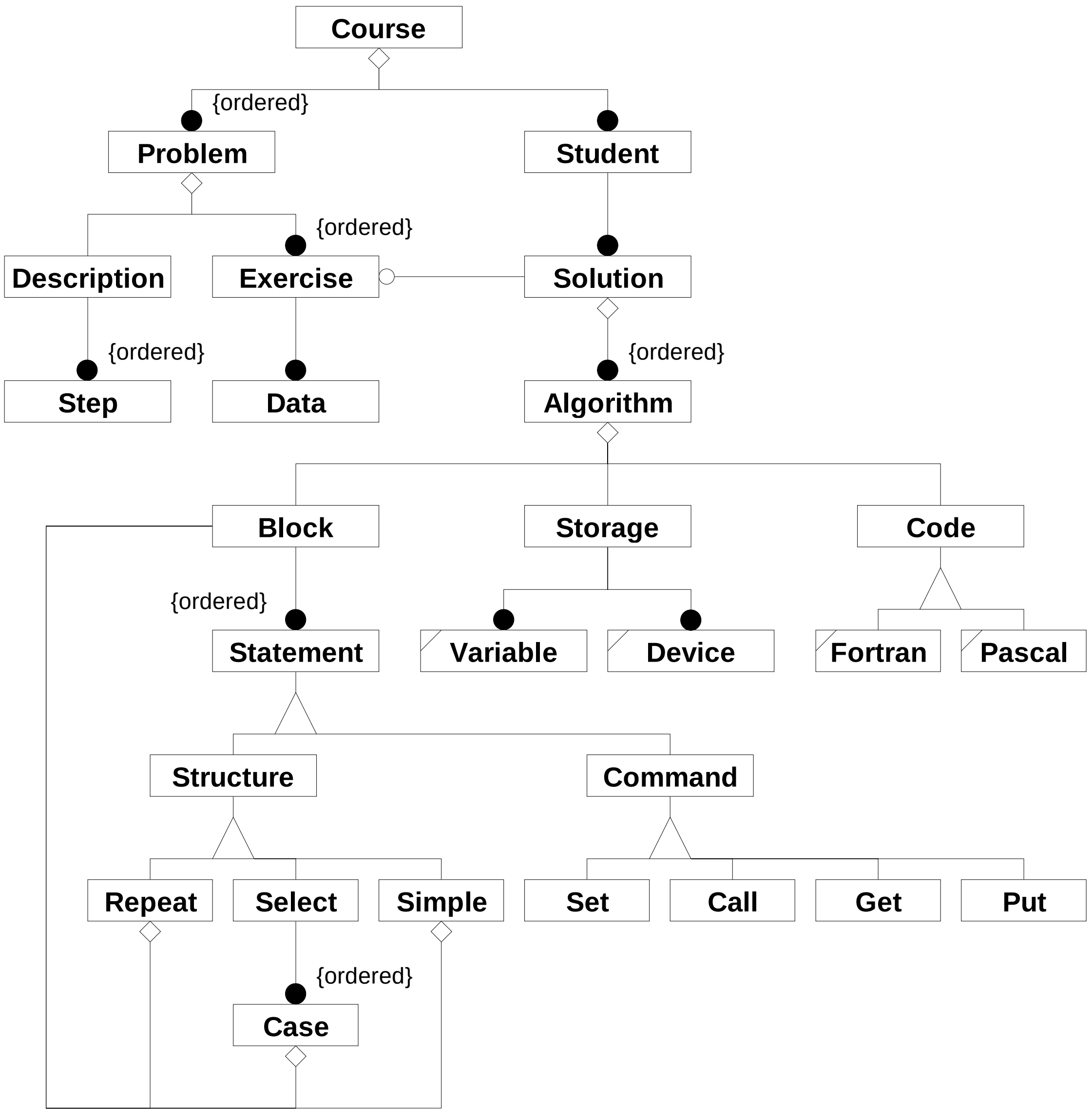


Figure 1 : CAPE Object Model

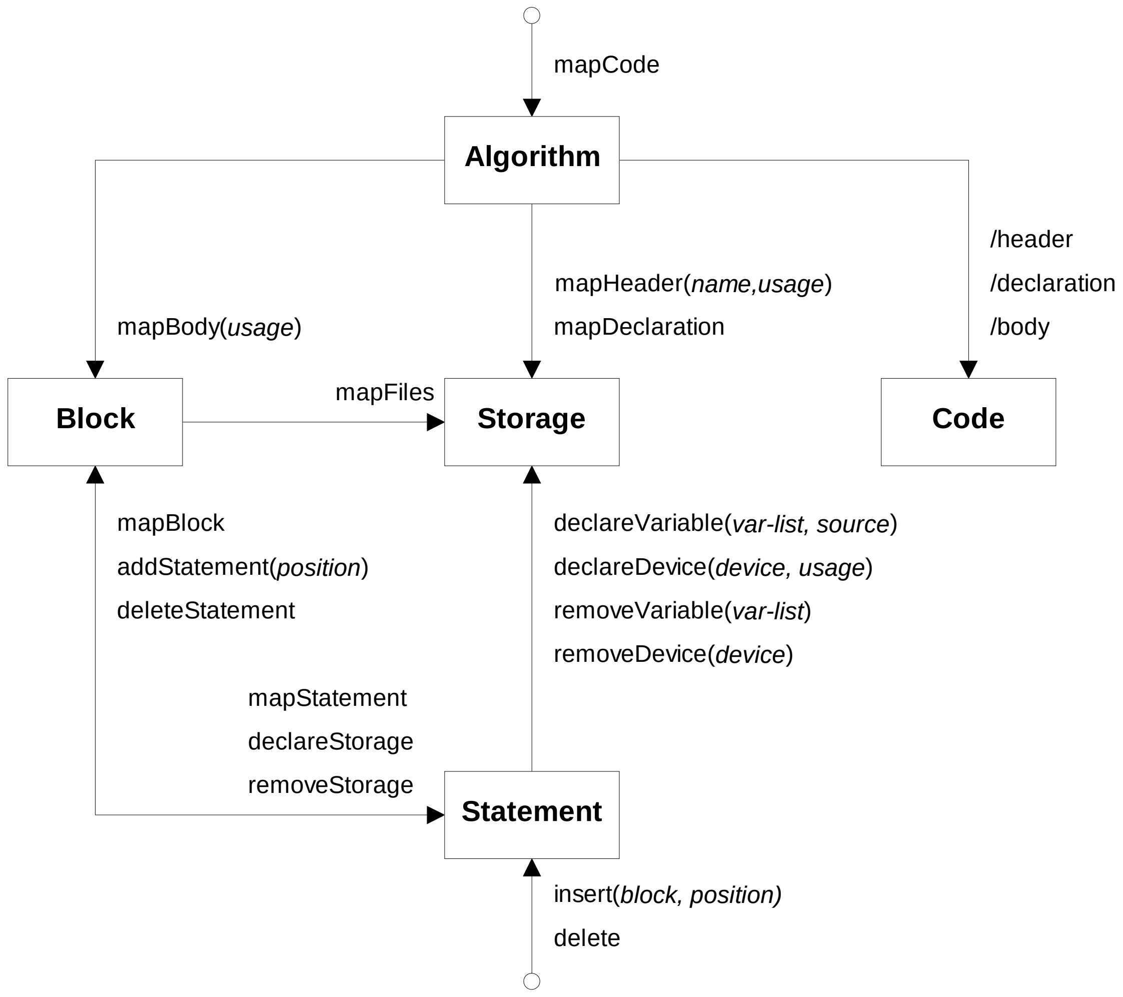


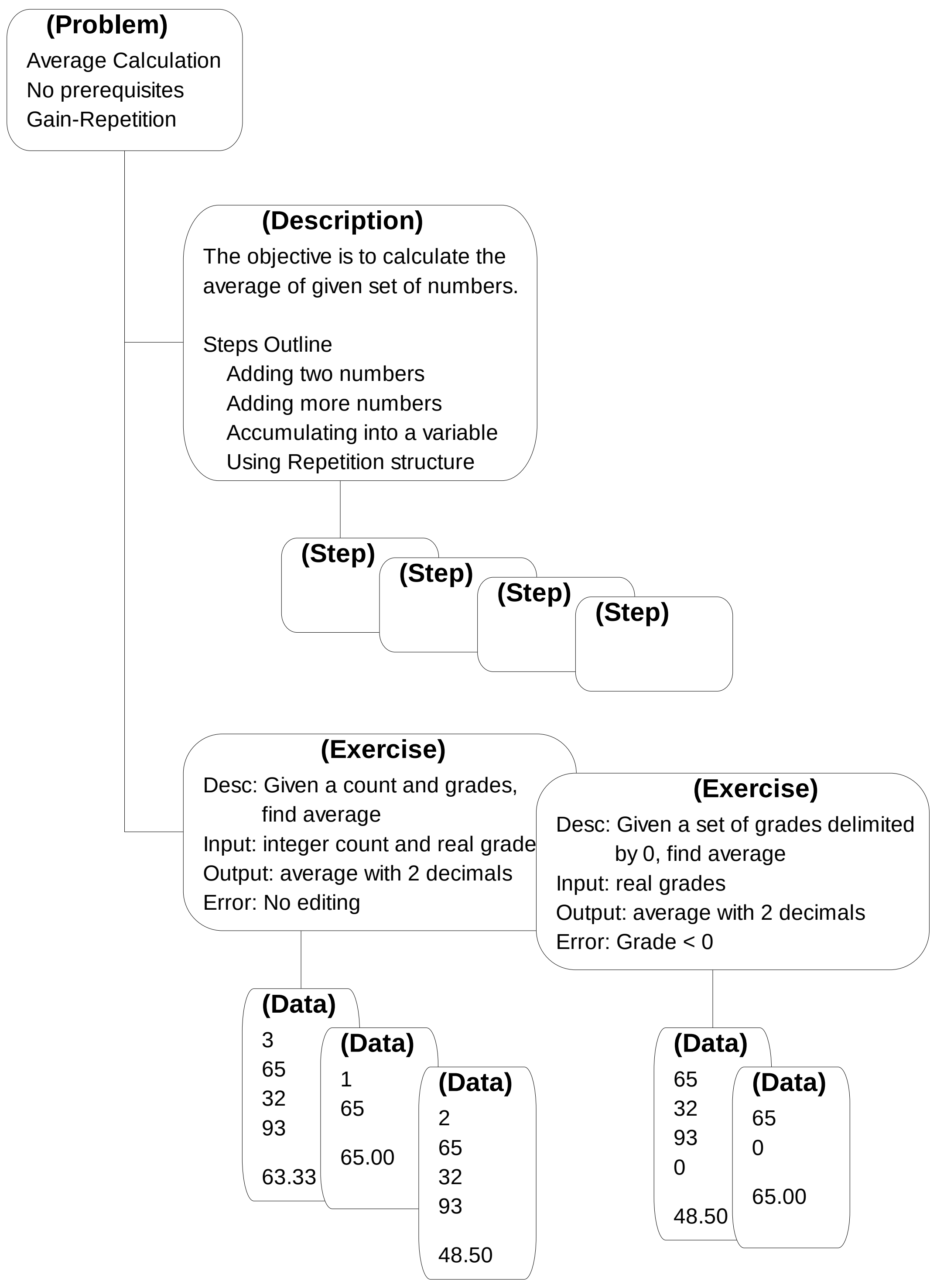
Figure 2 : Algorithm's Dynamic Model

CASE STUDY

An instance diagram describes how a particular set of objects relate to each other. The following discussion describes an instance of a problem definition object and its related components, solve one of the problem's exercises using the algorithm construction module, and then show how the algorithm is represented in Pascal and FORTRAN.

Problem Definition.4.1

Figure 3 shows an instance diagram for problem definition object. Problem name, prerequisite, and gained knowledge is indicated in the problem object. The description component is open to support various explanation strategies. Description object contains the objective of the problem's lesson and the outline of steps to be presented. Calculating an average could be described starting with two numbers then more numbers to show the need for an accumulator. The last step is to show the usefulness of repetition construct to implement accumulation process. The problem has two exercises as shown in the figure. An exercise has its own description, input, output, and error specification. The figure shows a short description of an exercise. The first exercise requires the development of a program that reads a count of grades followed by their set. Three sample data is assigned for that exercise. The first data checks the basic requirements, the second is used to validate the execution using only one input, and the third is used to check if reading will go beyond the specified count. The second exercise is similar to the first, except now the data is delimited by a given value and additional error handling is required.



(Problem)
Average Calculation
No prerequisites
Gain-Repetition
(Description)
The objective is to calculate the average of given set of numbers.
Steps Outline
Adding two numbers
Adding more numbers
Accumulating into a variable
Using Repetition structure
(Step)
(Step)
(Step)
(Step)
(Exercise)
Desc: Given a count and grades, find average
Input: integer count and real grade
Output: average with 2 decimals
Error: No editing
(Exercise)
Desc: Given a set of grades delimited by 0, find average
Input: real grades
Output: average with 2 decimals
Error: Grade < 0
(Data)
3
65
32
93
63.33
(Data)
1
65
65.00
(Data)
2
65
32
93
48.50
(Data)
65
32
93
0
48.50
(Data)
65
0
65.00

Figure 3 : Problem Definition Instance Diagram

Algorithm Construction

Figure 4 shows an instance diagram of the constructed algorithm that solves exercise one of the defined average calculation problem. The constructed algorithm is named 'Avg1', used as a 'Main' program, with a short description of what it does. The date and owner of the algorithm is maintained by the system. The algorithm consists of a block object, a derived storage object, and a code object which is discussed in the following section. The algorithm's block, with 'Main' as description, has six statements. The accumulator *Sum* is set to zero followed by a read of the count *N* of grades from *Input*. A repeated block *Total grade* is executed for an index *I* varying from *1* to *N* by *+1* steps. The repeated block has two statements to get *Grade* from *Input* then add it to *sum*. After repetition, *Average* is calculated in statement four. A select statement is used to conditionally put a status message. The first condition *a* where calculated average > 90 simply put an 'Honor' message. The second condition *b* where average is between 90 and 70 reports that the average is upgraded by 10 points. After this select statement, the average is reported to *Output* device.

Storage variables and devices objects were derived from the algorithm's block statements. A variable has a name, type, size, usage, and a derived count attribute. When a variable is created, a default type of *{type}* is used with its count set to one. Note how variable *Sum* is counted twice due to its dual usage. A device has a name, usage, filename, and a derived attribute of count. In our example the only device used is the input/output terminal. Output device is used in three statements where input device is used twice.

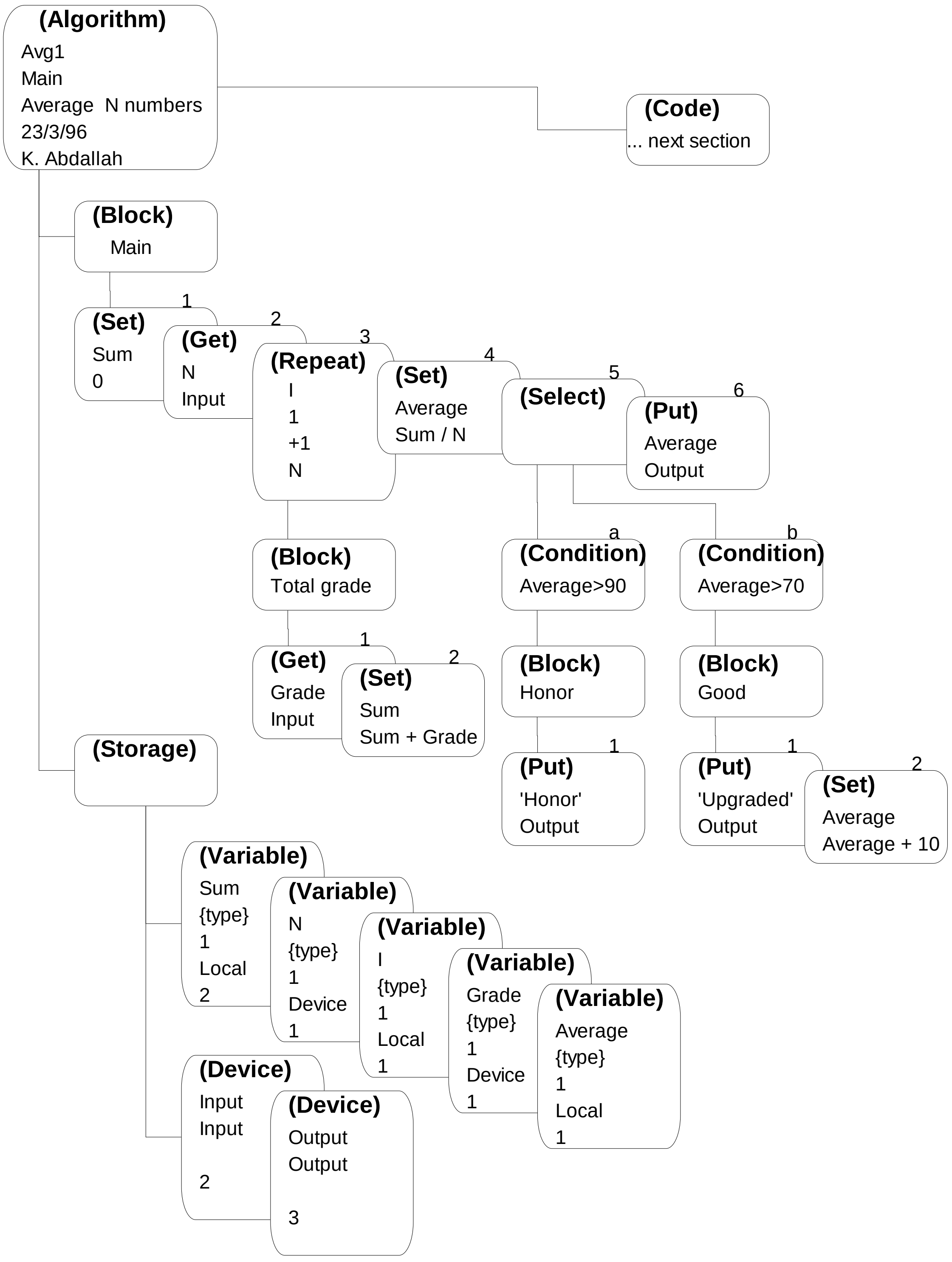


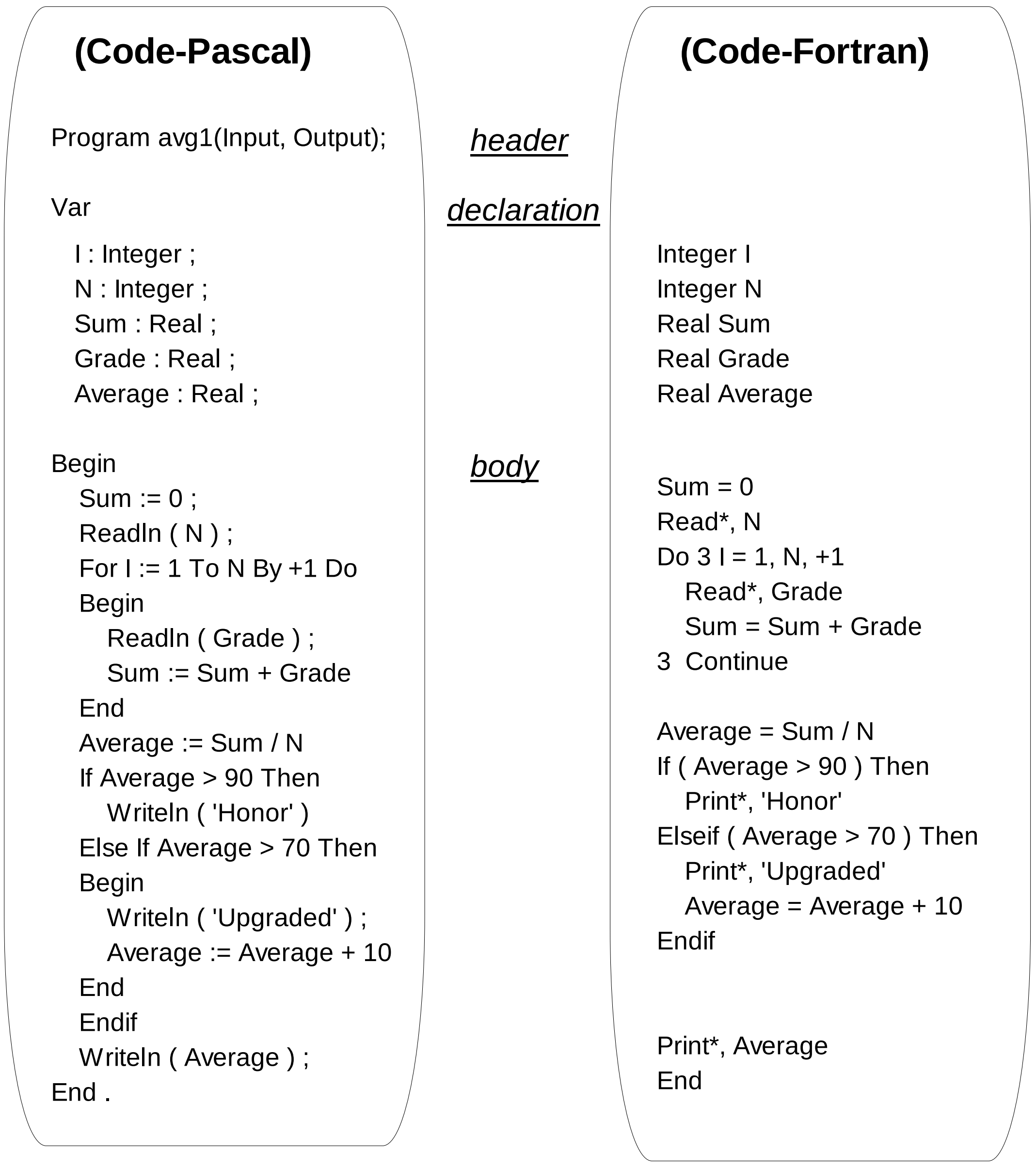
Figure 4 : Algorithm Construction Instance Diagram

Code Derivation

Once the algorithm has been constructed, the owner may complete the specification of defaulted attributes, such as variable type. The choice of language generates the corresponding code, as shown in Figure 5. The constructed code has a header, declaration, and body attributes. Storage object derives header and declaration content where block object derives the body content. Indentation is used to clarify the generated code since it was not explicitly specified in the model.

The generated code in both languages helps students view a simple representation of the constructed algorithm in different languages. Every language has its own naming convention and expression syntax. The code could be directly modified to have a valid compile run but these changes are not reflected in the algorithm. There are certain requirements that can not be represented using the algorithm construction, such as formatting an output or indenting the code. Our suggestion is to get the heart of the algorithm working first and then generate code that will be modified to include final touches.

FORTRAN does not recognize the specified 'If' conditions since they include '>' sign where it should have been 'GT' for greater than. Expressions specified in the algorithm are not validated by CAPE, the compiler does. After using the language for a while, methods of specifying expressions will eventually be consistent with what the language requires. The other note is that variable name 'Average' has more than seven characters, the compiler would not accept it.



(Code-Pascal)

```
Program avg1(Input, Output);

Var
  I : Integer ;
  N : Integer ;
  Sum : Real ;
  Grade : Real ;
  Average : Real ;

Begin
  Sum := 0 ;
  Readln ( N ) ;
  For I := 1 To N By +1 Do
  Begin
    Readln ( Grade ) ;
    Sum := Sum + Grade
  End
  Average := Sum / N
  If Average > 90 Then
    Writeln ( 'Honor' )
  Else If Average > 70 Then
  Begin
    Writeln ( 'Upgraded' ) ;
    Average := Average + 10
  End
  Endif
  Writeln ( Average ) ;
End .
```

(Code-Fortran)

```
Integer I
Integer N
Real Sum
Real Grade
Real Average

Sum = 0
Read*, N
Do 3 I = 1, N, +1
   Read*, Grade
   Sum = Sum + Grade
3  Continue

Average = Sum / N
If ( Average > 90 ) Then
   Print*, 'Honor'
Elseif ( Average > 70 ) Then
   Print*, 'Upgraded'
   Average = Average + 10
Endif


Print*, Average
End
```

Figure 5 : Algorithm Code Object

CONCLUSION

The variety of procedural programming languages widely used in practice stresses the importance of building computing environments aimed at helping students acquire the capability to solve programming problems independently of a specific language. Moreover, if such environments are endowed with the capability to produce code automatically in different languages, students are motivated to study algorithmic problems operationally, and are provided with a tool for rapid prototyping in various programming languages. On this basis we have designed the system described here to be used in the teaching/learning of programming with university students at an introductory level. For such students, understanding the role of algorithms and obtaining the knowledge on implementation techniques are issues in programming education and practice.

REFERENCES

Du Plessis, J. Van Biljon, C. Tolmie, and T. Wollinger, "A model for intelligent computer-aided education systems," *Computers & Education*, Vol. 24 no 2, Feb 1995. [1]

P. Forcheri and M. Molfino, "Software tools for the learning of programming: A proposal," *Computers & Education*, Vol. 23 no 4, Dec 1994. [2]

R. Price, *Computer-Aided Instruction: A Guide for Authors*, Brooks/Cole Publishing Company, Pacific Grove, California, 1991. [3]

M. Sammons, "Students assess computer-aided classroom presentations," *T.H.E. Journal*, Vol. 22 no 10, May 1995. [4]

S. Maddock, "A personal CAL workbook," *Computers & Graphics*, Vol. 18 no 3, May Jun 1994. [5]

نموذج تعليمي بمساعدة الحاسب الآلي للغات البرمجة الإجرائية

خالد عبد الله *، عبد الله السكيري **

* أرامكو السعودية، الظهران، المملكة العربية السعودية

** جامعة الملك فهد للبترول والمعادن، الظهران، المملكة العربية السعودية

المستخلص: أنظمة التعليم بمساعدة الحاسب الآلي هي برامج تدعم العملية التعليمية وتعتبر وسائل قيمة لتحسين التعلم . لذا من المناسب دعم بيئات البرمجة بقدرات تعليمية لمساعدة الطلاب لتعلم البرمجة . إن عملية تطوير أي برنامج على الحاسب الآلي تمر بثلاث أطوار مميزة: فهم المسألة ، تصميم الحل ، وكتابة البرنامج . وعليه فقد طورنا نموذجاً يدعى (CAPE) الذي يدعم عملية تطوير البرامج . يتكون هذا النموذج من وحدة تعريف المسألة ووحدة تصميم الحل الإجرائي مدعمة بوسيلة لوضع التصميم بلغتي (FORTRAN) و (Pascal) . من مميزات هذا النموذج أنه يحفز الطلاب لدراسة المسائل الإجرائية عملياً ويمدهم بآلية لإنتاج نموذج أولي سريع بعدة لغات للبرمجة .