Int. J. Engng Ed. Vol. 20, No. 1, pp. 46–51, 2004
Printed in Great Britain.

0949-149X/91 $3.00+0.00

Atomic Requirements in Teaching Logic Control Implementation*

HANANIA SALZER and ILYA LEVIN
Tel-Aviv University, School of Education, Israel. E-mail: salzerha@post.tau.ac.il

The paper introduces an innovative approach for teaching design with logic control. The proposed approach is based on: (a) a representation of the controlled system in a form of its control and operational interacting portions; (b) atomic requirement (ATR) specifications of the system. The ATR-based approach can be supported by a formal notation of transition formulae. Each of the ATRs corresponds to its specific formula. ATRs with the algebra of transition formulae can be considered as a theoretical basis in teaching logic control. While ATRs have been successfully used in large, commercial software development projects, they have never been applied to logic control teaching. In this paper we propose using ATRs for specifying logic control. This paper exposes specific properties of ATRs. Owing to these properties, the ATRs will guide students to partition specifications between control and operation, and to correctly identify control signals. As a direct result, students will apply information hiding between the control unit and the operational components.

INTRODUCTION

BUILDING PHYSICAL artifacts is an important educational tool in learning the principles of logic control [12]. However, knowledge of an artifact's structure and function may be insufficient for the comprehension of the intangible 'stuff' that traverses between its components [2, 22]. Students tend to ignore the signals that operational and control components send to each other [14].

Students, when given the freedom of creativity, may engage into tasks that they do not have the skills to make into reality. Students may start off their mobile robot projects with high-level plans for control only to find out rather late in their project's lifecycle that it was too complex to implement [13], and that goals must be repeatedly scaled back as the complexity of seemingly simple behavior is revealed [3].

The first author of this paper has implemented atomic requirement (ATR) specifications for large, commercial software development projects [19], and trained software developers in composing ATRs, such as in [20]. The main goal of the present paper is to apply the industrial experience to undergraduate education. Encouraged by the experience accumulated along the years, we propose overcoming the above two problems faced by students (control signals' ignorance and unbalanced complexity), by providing them with a technique that uses ATRs to articulate system functionality. The technique will lead students to partition a particular system's functionality between control and operation, and will lead to control signals sorting themselves out.

We define the notion of atomic requirement (ATR) specifications and explain its difference from non-atomic requirement specifications. We show that a controlled system can be viewed as consisting of two components, the control unit (CU) and the operational unit (OU). Then we show how ATRs are superior to non-atomic specifications in recognizing control functionality, and in helping devise control and operational modules. We also outline the suggested steps that students should take when designing a controlled system.

ATOMIC REQUIREMENT (ATR) SPECIFICATIONS

Atomic requirements (ATRs) are, primarily, 'well-formed requirements' *à la* ANSI/IEEE Standard 1233–1998 [8]. 'Well-formed requirements' are abstract, unambiguous, traceable and validatable (testable). In addition to being well-formed requirements, ATRs are also the result of splitting complex requirements into elementary, or indivisible, requirements. Usually, an ATR takes the form of a single sentence using non-formal language, nevertheless precisely expressing a specification. The chances are considerably better to achieve unambiguity with a set of ATRs than with an equivalent, non-atomic specification [19].

The implications of the ATR notion are wide, encompassing requirements and design specifications since both are documented in a similar way [18], and since a specification at any abstraction level can be viewed as both requirement and design [5, 9]. Indeed, atomic requirements (ATRs) have demonstrated their benefits in the hi-tech industry [19] when were used to document data processing applications in software requirements specifications documents, as well as in individual programs' detailed design documents.

* Accepted 28 September 2003.

An ATR in the logic control context relates binary input signals entering the control unit (CU) with its binary output signals. Following is a description of a special language of *transition formulae* that we use as a formal model for the logic control [10].

The set $X = x_1, x_2, \ldots, x_L$ of binary input signals is transferred from the operational unit (OU) to the CU. The set of binary signals $Y = y_1, y_2, \ldots, y_N$ is the set of control micro-operations, transferred from the CU of the system to the OU. The CU generates control micro-instructions that are subsets of the micro-operations set Y, which are executed concurrently. The OU performs micro-operations in one-to-one correspondence with the set Y.

A CU is associated with a set of transition formulae. A transition formula is constructed as follows. Each product term α_i, depending on a set of variables $X = x_1, x_2, \ldots, x_L$, is put into correspondence with a control micro-instruction Y_i, which is a subset of the micro-operations set Y. Product term α_i is assumed to be equal to 1 if and only if control micro-instruction Y_i should be performed. The resulting transition formula F_i associated with ATR_i is defined as:

$$F_i = \alpha_i Y_i + \bar{\alpha}_i Y_0$$

where

$$\alpha_i Y_i = \begin{cases} Y_i & \text{if } \alpha_i = 1 \\ 0 & \text{if } \alpha_i = 0 \end{cases}$$

The expression $\alpha_i Y_0$ in this formula tells explicitly that the ATR specifies only the actions that should be taken when the condition α_i materializes, but refrains from explicating what should happen otherwise. When the condition in one ATR does not realize, then the action specified in that ATR does not take place. The transition formula conveys this information by stating that the micro-instruction Y_0 (the empty micro-instruction) is executed.

We demonstrate below the correspondence between an ATR and its transition formula. Consider the following example ATR for a mobile robot that should avoid touching obstacles:

- *ATR-1: Keep turning left as long as facing an obstacle that is too close.*

ATR-1 is one of the many specifications that define the robot's logic control. The threshold distance that is considered to be 'too close' is defined in another ATR. The robot's CU receives from the OU two binary input signals: $x_1 = 1$ means that the robot faces an obstacle. $x_1 = 0$ means that the robot does not face an obstacle. $x_2 = 1$ means that the robot is in a safe distance from any obstacle. $x_2 = 0$ means that the robot is within dangerous proximity to an obstacle. The CU transmits to the OU a binary signal, indicating a micro-operation: $y_1 = 1$ signals the OU to make a right turn. $y_1 = 0$ signals the OU not to make a right turn.

Transition formula F_1 corresponds to ATR-1:

$$F_1 = x_1 \bar{x}_2 y_1 + (\bar{x}_1 + x_2) Y_0$$

Every ATR in the context of logic control implementation is in a one-to-one correspondence with a specific transition formula. The non-formal text in the ATR and the formal transition formula ($F_i = \alpha_i Y_i + \bar{\alpha}_i Y_0$) carry the same information. The Boolean function α_i consists of one Boolean product (product term). Each product term α_i represents the condition that the ATR describes. The control micro-instruction Y_i represents the operation that the ATR describes.

At the foundation of our approach is the reality that an ATR carries the specification of the smallest meaningful quantum of functionality. The one-to-one association between an ATR and the formal representation of a corresponding transition formula makes evident the ATR's indivisibility. The direct consequence of an ATR's oneness is that it cannot carry a functionality that is both control and operation. Therefore, after atomizing any specification containing control functionality, the resulting ATRs can be segregated unambiguously into two groups, control and operation.

Consequently, we are ready now to formulate a procedural definition for an ATR in the context of logic control.

> *A control-related atomic requirement specification (control-related ATR) is a requirement or design specification that is (a) associated with the system's control functionality, (b) is well-formed, (c) consists of a condition and of a corresponding operation, and (d) the condition and the operation are indivisible at the abstraction level where the specification is being considered.*

See Fig. 1 for a concept map of the notion of ATR.

SYSTEM SYNTHESIS FROM CONTROL UNIT AND OPERATIONAL UNIT

A controlled system can be viewed as composed of two high-level components, the control unit (CU) and the operational unit (OU), as shown in Fig. 2. The CU is the part of the system responsible for taking the decisions that control the system's behavior. The OU is defined as all system components, except the CU. The communication between the system and its environment is only across the OU-environment interface. The CU does not interact directly with the system environment; it communicates only with the OU. The OU can be viewed as an interface between the CU and the system's environment. Figure 2 presents two input and output pairs. One pair is between the environment and the OU, and the other pair is between the OU and the CU. From the CU design's point of view, this representation fully complies with the Four-Variable Model [6, 18].

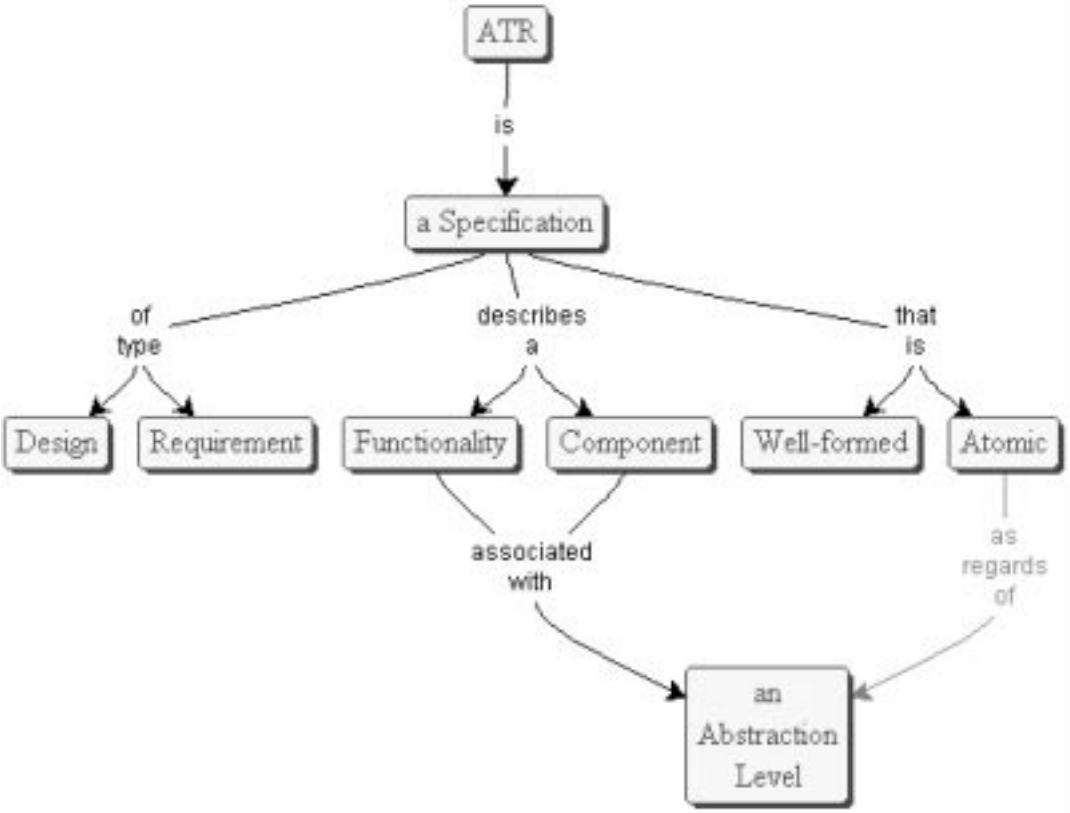

Fig. 1. Concept map of ATR's properties.

The system partition into a CU and an OU is a special case of system modularity. Modular design constructs a system from a number of modules with well-defined interfaces; each one is small enough and simple enough to be thoroughly understood and well programmed [17]. Modular design brings with it great productivity improvements [7]. Parnas coined the term 'information hiding' [16], and suggested that the design of each module will be independent of other modules designs [17]. Two of the factors contributing to module independence are *coupling* and *cohesion* [15].

Module coupling

Module coupling is the degree of connection between modules; hence it is a measure of module interdependence. Designers should strive for *data coupling*, where two modules communicate by passing arguments [15].

Mobile robot construction is a popular educational tool for exposing students to hands-on experience with controlled systems, sometimes peaking in a competition, such as in [21]. Acquaintanceship with a program, which trains students towards mobile robot competitions [4], reveals that rarely a team designs the software with control segregated from operation. Students on this program build autonomous mobile robots that explore apartment corridors, peaking into rooms in search for a fire. A typical software program section monitors a fire detection sensor, and manipulates two motors connected to the robot's two active wheels. In this overly simplified example, the program section implements the non-atomic, and rather cryptic, requirement Req-2, below:

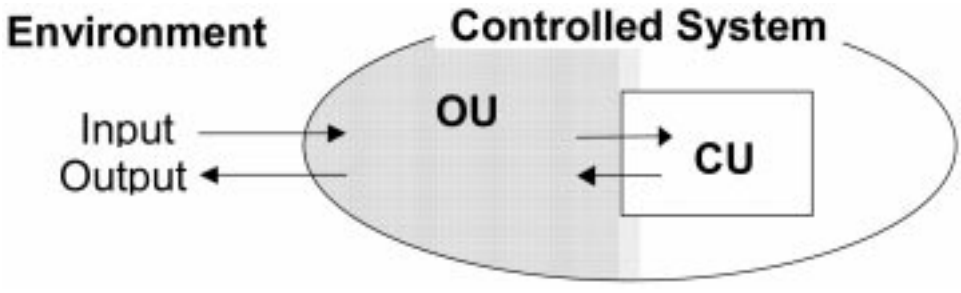

Fig. 2. The system partitioning into an operational unit (OU) and a control unit (CU).

Req-2: When the value at the memory address allocated to the Pyroelectric Sensor drops below 102, or peaks above 153—then set to 1 the two Motor Direction bits, and set to 160 the two T-on (cycle time on) registers allocated to the left and to the right motors.

The program section directly references memory locations in a Motorola 68HC12 family microcontroller. In contrast, a well-designed CU should not be aware of hardware locations or of the threshold value determining a certain state, because these are 'secrets' of OU components. Instead, the CU should communicate with the rest of the system (the OU) only via binary input and output signals [1, 11] thus providing pure data coupling.

The set of ATRs below is the result of splitting the non-atomic requirement Req-2 into atomic requirements. The CU implements ATR-3:

- ATR-3: *When flames are detected at the front—move forward in medium speed.*

The OU implements ATR-4 through ATR-7:

- ATR-4: *The robot is facing flames when the pyroelectric sensor's output drops to below +2V or peaks above +4V.*
- ATR-5: *The value at the memory address allocated to the pyroelectric sensor's measurement is: 51×(sensor output in volts)*
- ATR-6: *To move forward, release brakes and turn forward both left and right motors.*
- ATR-7: *Medium motor speed is achieved by a duty cycle of 63%.*

According to these ATRs the OU sends to the CU binary signals indicating whether it is facing flames. The CU sends to the OU a binary signal whether to move forward or not. Thus, the ATRs have facilitated the design of data coupling between the CU and the OU.

Module cohesion

Module cohesion is the degree of inner self-determination of the module; hence it measures the strength of the module's independence. Designers should strive to design *functionally cohesive* modules. In such a module all of the elements are related to the performance of a single function [15].

The above-mentioned procedure carries out three functions: it taps a certain hardware location (memory), then it determines whether the received input has exceeded a certain threshold, and finally, it sets absolute values at certain memory locations according to the predicate's outcome. This is an example of sequential cohesion.

Replacing the non-atomic requirement Req-2 with the six ATRs (ATR-2 through ATR-7), results with the different functions implemented by separate modules. Some OU modules handle inputs from certain sensors, and make the derived information available for the CU in the form of binary signals. Other OU modules respond to CU signals by operating their respective actuators (such as certain motors). Finally, the CU has only one function: to decide which OU functions to activate at any point of time.

We have shown that ATRs facilitated the design of a functionally cohesive CU module, and functionally cohesive modules in the OU.

LEARNING ACTIVITY

Encouraged by the success of professional software designers and programmers in writing ATRs, we suggest a process that incorporates the same technique. Although ATR lists are useful for concisely specifying any component of a system, the subject matter of this paper limits our scope to the CU and to the OU components that communicate with the CU.

To prepare the students, they learn the notions of cohesion and coupling, and the phrasing of well-formed requirements. Finally, they learn atomizing non-atomic requirements.

In the following process the students design a CU and a set of software components that communicate directly with the CU:

1. Atomize the requirements in the manner shown for the example above, and thus identify the pure logic control ATRs.
2. Separate the control ATRs from the operational ATRs.
3. The control ATRs dictate explicitly the binary input signals to the CU and the binary output signals (micro-instructions). List these signals.
4. List and design the software components that generate the input signals necessary for the CU, and the software components that act in response to the micro-instructions.

If complexity of student-invented systems may be unsuitable for the course goals, assess the complexity of the CU compared to the operational components as reflected by their respective ATRs. We would like to encourage the use of ATRs to specify also the rest of the system.

CONCLUSIONS

This paper suggests incorporating the use of atomic requirement (ATR) specifications [19] into the teaching of logic control design. It presents an example in the context of mobile, student-built robots.

A control related ATR is defined as a requirement or design specification that is associated with the system's control functionality, is well-formed, consists of a condition and of a corresponding operation, and the condition and the operation are indivisible at the abstraction level where the specification is being considered.

The control unit (CU) is defined as the system component whose sole role is to decide what actions the operational unit (OU) takes at any moment. For this effect the CU sends the OU

signals called micro-operations and receives signals about the OU's state. The micro-operation is a binary signal that tells the OU whether to do something or not to do it. Similarly, the messages from the OU to the CU are also binary.

An ATR carries the specification of the smallest meaningful quantum of functionality. From this fact stems the conclusion that after atomizing a sufficiently detailed set of system specifications, each one of the resulting ATRs can be unambiguously allocated to either the CU or to the OU.

Segregation between the CU and the OU should lead students to discover the need for a communication between the two components, and hence, to the need for some kinds of signal. Not only that, but also the abstract nature of control-related ATR's text explicitly suggest what messages the binary signals carry.

This paper argues that ATRs will help students to design CUs that feature functional cohesion and that are connected to the system's OU through data coupling.

This paper tackled a few, known problems in the teaching of logic control. It tailored the suggested solution to the mechanism of those problems by capitalizing on the ATR's unique properties, and by recruiting ATRs to play a key role in implementing well-established system engineering practices. The authors are on the way of performing the appropriate experiments focused on the potential of ATR-based logic control design by students.

Acknowledgment—We are thankful to Eli Kolberg for providing valuable information from his experience in leading robot-building student projects.

REFERENCES

1. S. Baranov, *Logic Synthesis for Control Automata*, Kluwer Academic Press (1994).
2. J. de Kleer, and J. S. Brown, Assumptions and ambiguities in mechanistic mental models, in *Mental Models*, Gentner, D. and Stevens, A. L. (eds.), Lawrence Erlbaum Associates, Hillsdale, New Jersey (1983) pp. 155–190.
3. C. Hancock, Children's understanding of process in the construction of robot behaviors, *Varieties of Programming Experience*, AERA Seattle (2001).
4. C. G. Harrison and P. L. Jones, A creative class project based on VHDL, synthesis and FPGA Design., *Int. J. Elect. Enging. Educ.*, **34**, 1997, pp. 370–375.
5. R. Harwell, E. Aslaksen, I. Hooks, R. Mengot and K. Ptack, What is a requirement? *Proc. Third Int. Symp. NCOSE INCOSE* (1993).
6. C. L. Heitmeyer, R. D. Jeffords and B. G. Labaw, Automated consistency checking of requirements specifications, *ACM Trans. Software Eng. Methodol.*, **5**(3) 1996, pp. 231–261.
7. J. Hughes, Why functional programming matters, *Comput. J.*, **32**(2), 1989, pp. 98–107.
8. IEEE, Guide for developing system requirements specifications, in *IEEE Standards, Software Engineering, Volume One, Customer and Terminology Standards*, IEEE-Std-1233, Computer Society (1998).
9. H. Kilov and J. Ross, *Information Modeling: An Object-oriented Approach*, Prentice-Hall (1994) pp. 28–32.
10. I. Levin and V. E. Levit, Controlware for Learning with Mobile Robots, *Computer Science Education*, **8**(3), 1998, pp. 181–196.
11. I. Levin and D. Mioduser, A Multiple-Constructs Framework for Teaching Control Concepts, *IEEE Trans. Edu.*, **39**(4), 1996, pp. 488–496.
12. P. H. Lewis, Introducing discrete-event control concepts and state-transition methodology into control curricula, *IEEE Trans. Education*, **37**(1), 1994, pp. 65–70.
13. F. G. Martin. Ideal and real systems: a study of notions of control in undergraduates who design robots, in *Constructionism in Practice: Rethinking the Roles of Technology in Learning*, Y. Kafai and M. Resnick (eds.), Mahwah, NJ: Lawrence Erlbaum (1996) pp. 297–322.
14. D. Mioduser, R. L. Venezky and B. Gong, Students' perceptions and designs of simple control systems, *Computers in Human Behavior*, **12**(3), 1996, pp. 363–388.
15. G. J. Myers, *Reliable Software Through Composite Design*, Petrocelli/Charter, New-York (1975).
16. D. L. Parnas, Information distribution aspects of design methodology, *Proc. 1971 IFIP Congress*, pp. 339–344.
17. D. L. Parnas, On the criteria to be used in decomposing systems into modules, *Commun. ACM*, **15**(12), 1972, pp. 1053–1058.
18. D. L. Parnas, Functional documentation for computer systems, *Sci. Comput. Program.*, **25**(1) 1995, pp. 41–61.
19. H. Salzer, ATRs (atomic requirements) used throughout development lifecycle, *12th Int. Software Quality Week (QW99)*, (6S1), San Jose, CA (1999).
20. H. T. Salzer, *ATRs (Atomic Requirements) Writing Workshop at the 3rd International Software Quality Week Europe 1999*, Brussels, Belgium (1999).
21. Trinity College Fire Fighting Home Robot Contest website: http://www.trincoll.edu/events/robot
22. M. D. Williams, J. D. Hollan, and A. L. Stevens, Human reasoning about a simple physical system, in D. Gentner and A. L. Stevens (eds.) *Mental Models*, Lawrence Erlbaum Associates, Hillsdale, New Jersey, (1983) pp. 131–154.

Dr. Ilya Levin received the M.Sc. Degree in Electrical Engineering (Cum Laude) in Leningrad Transport Engineering University and Ph.D. degree in Computer Engineering

from the Latvian Academy of Science in 1976 and 1987 respectively. During 1985–1990 he was the Head of the Computer Science Department in the Leningrad Institute of New Technologies (Russia). During 1993–1996 he was the Head of the Computer Systems Department of the Center for Technological Education, Holon (Israel). Being presently a faculty of the School of Education of Tel Aviv University, he is a supervisor of Engineering Education program. He is an author of more then 50 papers both in Design Automation and in Engineering Education fields.

Hanania Salzer is a PhD student at the School of Education in the Tel-Aviv University. For the last 20 years he worked in the software industry. He earned his MSc in Zoology and BSc in Biology at the Tel-Aviv University.