Atomic Specifications and Controlware Design

ILYA LEVIN[1], VADIM E. LEVIT[2] and HANANIA T. SALZER[1]
[1]School of Education, Tel-Aviv University, Tel-Aviv, Ramat-Aviv 69978, ISRAEL
[2]Computer Science Department, Holon Academic Institute of Technology, Holon, ISRAEL
i.levin@ieee.org http://muse.tau.ac.il/ilya/, levitv@hait.ac.il, salzerha@post.tau.ac.il

Abstract: - In this paper we introduce Atomic Requirements (ATR) based specifications for designing logic controllers. We use the concept of Controlware, which is a toolkit introduced earlier for designing controllers.

A set of ATRs specifies a controller's functionality. Each ATR is translated into one transition formula. A controller is described as a set of transition formulae.

With Controlware, intelligent behavior of controlled systems is created through the composition of component subcontrollers. The composition of the subcontrollers' behaviors may result in conflicts among their actions. A second set of ATRs is used to specify the priority arbitration on this set of actions. The action-priority ATRs too are translated into transition formulae.

The proposed ATR based approach allows specifying both the basic functionality of controllers and the action-priority arbitration in a framework of unified design methodology.

An example of the ATR based design for a specific mobile robot is presented.

Key-Words: - Atomic requirement; controlled system; Controlware; decision table; natural language; specification.

1 Introduction

Controlware [1] is specified as a combination of formal models of individual control specifications and a formal model of the controller composition. According to our approach, the formal model of a control specification is a *transition formula*; and the formal model of a controller's composition is a partially ordered layer-architecture. Controlware includes: a decision table based language for defining controllers' and subcontrollers' behavior; a transition formulae based notation enabling transformation of corresponding subcontrollers' descriptions; subcontrollers' composition rules and a priority arbitration mechanism for resolving the conflicts between subcontrollers.

In this paper we propose to use natural language *atomic requirements (ATRs)* [2,3] for providing an appropriate Controlware design style. The atomic requirements can be considered as a universal, non-formal language for specifying the controller's behavior. We investigate ATRs properties, and emphasize the correspondence between the ATRs and transition formulae. We describe a specific subclass of transition formulae being in one-to-one correspondence with a set of ATRs. Each ATR can be considered as a verbal equivalent of the corresponding transition formula of this subclass.

This work is intended as an attempt to provide a unified methodology of designing controlled systems. This methodology is built on the ATR based description of controllers, on the transition formulae formal notation and on the partial ordering priority arbitration on the set of microoperations.

In this paper we present a spreadsheet-based working implementation of the method for control automata realization by decision tables to demonstrate this kind of control system [1].

The rest of the paper is organized as follows. The formal model of a controller is discussed in section 2. *ATR* based specification is introduced in section 3. The formal composition of controllers and subcontrollers is presented in section 4. A

case study is provided in section 5 and conclusions – in section 6.

2 Formal Model of the Controller

A broad spectrum of controlled systems can be represented as compositions of control and operation units [4]. The controller (control unit) is the part of the system responsible for taking the timely decisions that control the system's behavior. The operation unit is defined as all system components, except the controller. The set $X=x_1,x_2,...,x_L$ of binary input signals is transferred from the operation unit to the controller. The set of binary output signals $Y=y_1,y_2,...,y_N$ is the set of control microoperations, transferred from the controller of the system to its operation unit. The controller generates control microinstructions that are subsets of the microoperations set Y. The operation unit performs microoperations in one-to-one correspondence with the set Y.

2.1 Transition Formulae

We use a special language of *transition formulae* [1] as a formal model within Controlware. A controller (which is not divided into several subcontrollers, as described below) is associated with one transition formula. A transition formula is constructed as follows [1,3]. Each product term α_i, depending on a set of variables $X=x_1,x_2,...,x_L$, is put into correspondence with a control microinstruction Y_i, which is a subset of the microoperations set Y. Product term α_i is assumed to be equal to 1 if and only if control microinstruction Y_i should be performed. Transition formula F associated with a controller is defined as:

$$F = \sum_{i=1}^{n} \alpha_i Y_i \tag{1}$$

where

$$\alpha_i Y_i = \begin{cases} Y_i & if\ \alpha_i = 1 \\ 0 & if\ \alpha_i = 0 \end{cases} \tag{2}$$

Assume the following example:

$$F = \alpha_1 Y_1 + \alpha_2 Y_2 + \alpha_3 Y_3 = x_1 x_2 y_1 y_2 + \bar{x}_1 y_1 + x_1 \bar{x}_2 y_3 \tag{3}$$

where $\alpha_1 = x_1 x_2$, $\alpha_2 = \bar{x}_1$, $\alpha_3 = x_1 \bar{x}_2$, $Y_1 = y_1 y_2$, $Y_2 = y_1$ and $Y_3 = y_3$.

In this example, when $x_1 x_2 = 1$ then the microoperations $y_1 y_2$ are transferred from the controller to the operation unit. When $\bar{x}_1 = 1$ or when $x_1 \bar{x}_2 = 1$ then the microoperation y_1 or y_3 is transferred, respectively.

2.2 Subcontrollers

A system's controller may be composed of several *subcontrollers*. It is possible to represent simultaneous functioning of several different subcontrollers. To represent simultaneous functioning of n subcontrollers described by the corresponding formulae F_1 through F_n, we define a product operation on the set of transition formulae as follows:

$$F_{controller} = F_1 \times \ldots \times F_n \overset{def}{=} \sum_{i_1,\ldots,i_n} \alpha_{i^1} \ldots \alpha_{i^n} Y_{i^1} \ldots Y_{i^n} = \sum_{i_1,\ldots,i_n} \{\alpha_{i^1} \ldots \alpha_{i^n}\} \{y_{i_1^1} y_{i_2^1} \ldots y_{i_M^1} y_{i_1^2} y_{i_2^2} \ldots y_{i_N^2}\} \tag{4}$$

This formula exploits the usual approach to the idea of priority based on the supposition that the pair-wise priorities are organized as an order on the set of microoperations Y. This means that for every pair of microoperations $\{y_k, y_l\}$ there exist three options, shown as follows:

$$\{y_k, y_l\} = \begin{cases} y_k & if \quad y_k > y_l \\ y_l & if \quad y_k < y_l \\ y_k y_l & if \quad y_k = y_l \end{cases} \tag{5}$$

Assume that the three following formulae describe three subcontrollers of a controller:

$$F_1 = x_1 y_1 + \bar{x}_1 \bar{x}_2 y_2 + \bar{x}_1 x_2 y_3$$
$$F_2 = x_1 y_2 + \bar{x}_1 y_3$$
$$F_3 = x_2 y_3 + \bar{x}_2 y_2, \tag{6}$$

and that the following partial order [5] exists on the set of microoperations:

$$y_3 > y_1, \quad y_3 > y_2 \tag{7}$$

The product $F = F_1 \times F_2 \times F_3$ describes the mutual functioning of the subcontrollers corresponding to the formulae F_1, F_2 and F_3 as well as to the order on the set of microoperations:

$$F = F_1 \times F_2 \times F_3 = (x_1 y_1 + \bar{x}_1 \bar{x}_2 y_2 + \bar{x}_1 x_2 y_3) \times (x_1 y_2 + \bar{x}_1 y_3) \times (x_2 y_3 + \bar{x}_2 y_2) = (x_1 \{y_1, y_2\} + \bar{x}_1 \bar{x}_2 \{y_2, y_3\} + \bar{x}_1 x_2 y_3) = x_1 y_1 + \bar{x}_1 \bar{x}_2 y_2 + \bar{x}_1 x_2 y_3 \quad (8)$$

2.3 Decision Tables

Each transition formula can be presented in a decision table [7]. Columns appearing in decision tables are marked by inputs $x_1, x_2, \ldots, x_L$, and by microoperations $y_1, y_2, \ldots, y_N$. Fig.1 illustrates this decision table for a subcontroller containing the transition formula $F = x_1 x_2 y_1 y_2 + \bar{x}_1 y_1 + x_1 \bar{x}_2 y_3$.

Input		Output		
x_1	x_2	y_1	y_2	y_3
1	1	1	1	
0		1		
1	0			1

Fig. 1: The decision table presenting the transition formula $F = x_1 x_2 y_1 y_2 + \bar{x}_1 y_1 + x_1 \bar{x}_2 y_3$

A controller defined by a set of transition formulae can be presented using a *two level structure* of decision tables, as described earlier [1]. The "Formal Model of Controller Composition" section presents the two-level structure in detail.

3 Atomic Requirement Specifications of a Subcontroller

It is possible to specify a system component completely by a list of specifications. A controlled system's controller is no exception. We use specifications of *atomic requirements*, or ATRs [2,3], to specify control functionality. Then, we convert each ATR into a transition formula. Consequently, each natural language specification (ATR) is associated with the respective formal specification (transition formula).

This paper uses the term *requirement* to denote any specification that a system, or a component thereof, such as a controller, must accomplish [8]. *ATRs* are, primarily, "well-formed requirements" *à la* ANSI/IEEE Standard 1233-1996 [9]. "Well-formed requirements" are abstract, unambiguous, traceable and validateable (testable). In addition to being well-formed requirements, ATRs are also the result of splitting complex, evolving requirements into elementary requirements that are indivisible at the abstraction level in which they are being considered. ATRs at different abstraction levels of controller design are discussed in [*9]. In this paper we concentrate on the description of a controller at two abstraction levels – ATRs and transition formulae. The significance of these particular two abstraction levels is that we show a straightforward translation between natural language and formal language. ATRs at the lowest non-formal abstraction level are translated into transition formulae, which represent a formal implementation language.

Usually, an ATR takes the form of a single sentence using non-formal language, nevertheless precisely defining a specification. Every ATR deals with only a single functionality. Therefore, chances are considerably better to achieve unambiguity with a set of ATRs than with an equivalent non-atomic specification [2].

We define an ATR as a requirement specification having an indivisible condition and resulting in some microoperations. Therefore we suggest that each ATR can be formally expressed as a transition formula having the following general format:

$$F_i = \alpha_i Y_i + \bar{\alpha}_i Y_0 \quad (9)$$

where the α_i is a product term on the subset of the input variables $X = x_1, x_2, \ldots, x_L$, Y_i is a subset of the set $Y = y_1, y_2, \ldots, y_N$ of microoperations, and Y_0 is the empty microinstruction. Thus:

$$F_i = x_{i_1}^{e_1} \ldots x_{i_n}^{e_n} y_{i_1} y_{i_2} \ldots y_{i_m} + \overline{x_{i_1}^{e_1} \ldots x_{i_n}^{e_n}} Y_0 = x_{i_1}^{e_1} \ldots x_{i_n}^{e_n} y_{i_1} y_{i_2} \ldots y_{i_m} + \left(x_{i_1}^{1-e_1} + \ldots + x_{i_n}^{1-e_n}\right) Y_0, \quad (10)$$

where

$$x_i^e = \begin{cases} x_i, & if\ e = 1 \\ \bar{x}_i, & if\ e = 0. \end{cases} \quad (11)$$

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 expressed by the product term α_i

materializes, but refrains from explicating what should happen otherwise. When the product term 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 empty microinstruction (Y_0) is executed.

Usually, the specifications at the abstraction level of a subcontroller include several ATRs. Therefore, a subcontroller can be represented by a transition formula that is the sum of the transition formulae corresponding to all ATRs of the subcontroller:

$$F = \sum_i \alpha_i Y_i + \left(\overline{\sum_i \alpha_i} \right) Y_0 \qquad (13)$$

4 Formal Model of Controller Composition

Representation of a controller could be too involved to comprehend all details at the same time. Furthermore, different subject mater experts (SMEs), responsible for separate portions of the requirements, may have a hard time coordinating conflicting requirements. It is possible to check requirement specifications *post factum* for self-consistency using formal methods, however the technique that we suggest here addresses during requirements elicitation the possible conflicts among microoperations (output). The suggested technique simplifies this aspect of the requirements specification process for a controller with n input signals reducing complexity from $O(2^n)$ to only $O(n^2)$ [1].

4.1 Conflict Resolution

A *conflict* between microoperations is defined as microoperations that should never be performed simultaneously. For example, an SME may determine that the microoperations "stop" and "turn left" should never be performed simultaneously. We do not deal, in this paper, with the conflicts that are defined for specific conditions only.

In this section we compare three alternative approaches to resolve such conflicts: no priority, layer-driven priority and microoperation-driven priority.

With the *no priority architecture* all 2^n possible input vector combinations are examined at design time. SMEs must agree among them, and explicitly tell for all combinations what the corresponding output microoperations should be. This architecture may not be practical for a controller with a large set of input signals (a large n), or when different input or output subsets are defined by different SMEs.

The *layer-driven priority architecture* comprises a family of basic independent controllers, organized in layers [10]. Each such controller, or layer, is capable of creating a sub-system behavior. Since different layers' microoperation outputs may be in conflict with each other, the relative level of each layer defines its priority; a higher layer's microoperation output overrides any conflicting output made by all lower layers. The layered design is intended for evolutionary development of the control logic, that is, a new, higher priority layer is added on top of older layers, without modifying the latter [10]. A limitation of this architecture is that all microoperations of a layer are in the same priority relative to the other layers' microoperations.

The first of the two levels in the *microoperation-driven priority architecture* is constructed of independent subcontrollers. The second level is a single-component arbitrator that resolves conflicts among the microoperations generated in the first level. The two levels, combined, comprise the controller. This architecture is convenient from a requirements engineering point of view, because the subcontrollers of the first level can be defined fairly independently, and the individual components, which deal with only a few inputs each one, have a reasonable complexity. Arbitration over potentially conflicting microoperations is defined as a partial order among the first-level subcontrollers' output signals; hence each control output has its own priority. The complexity of a partial order is manageable even with a relatively large number of microoperations and with several SMEs having independent concerns. This architecture is described below.

4.2 Controller Specifications

A common approach to designing a complex system is to decompose it into components with specific interactions among them, then to design the details for each component. This plot is iterative. Only the most detailed design work products are exhaustive enough for constructing the actual system components [8].

A controlled system's design involves the identification of the control logic as one of its components, and the definition of interactions between the control logic and the rest of the system. We design the subcontrollers by following these steps: (i) Identify the control logic specifications. (ii) Decompose specifications into ATRs. (iii) Cluster specifications into groups within which specifications do not contradict each other according to SMEs' claims. The arbiter will resolve contradictions among clusters. (iv) Use each cluster of specifications to synthesize a subcontroller.

4.3 Arbitration Specifications

We propose to complete the ATR based controller's specifications by an additional information concerning competitions between microoperations that can appear on different subcontrollers' output as a result of particular ATRs' functioning. Indeed, two or more ATRs, being defined properly and specifying different subcontrollers, can cause different subcontrollers to produce contradicting microinstructions. Consequently, arbitration between produced microoperations is required.

In this paper we develop an approach based on the ATR paradigm to specify priorities among microoperations. We propose to define the ATRs for priority specifying. We transform the natural language ATRs comprising the priority specifications into the rigorous notion of a partial order on the set of microoperations.

To construct the partial order, let $Y=y_1,y_2,\ldots,y_N$ be the set of all microoperations and the pair $\{Y,<\}$ be the partial order (*poset*) formalizing the natural idea of priority. We represent the $\{Y,<\}$ partial order by matrix A of the corresponding relation (*partial order matrix*), where $A_{(i,j)}=1$ means that $y_i>y_j$ or $y_i=y_j$, $A_{(i,j)}=0$ means that $y_i<y_j$, and $A_{(i,j)}=$"-" means that y_i and y_j are incomparable, i.e., can be performed simultaneously (for example, all microoperations comprising a microinstruction have to be incomparable). Below in the "Case Study" section we present an example of the partial order in a diagram and in the matrix forms.

Let us assume that a controller generates the set of microoperations $Y=y_1,y_2,\ldots,y_N$. Obviously, any microinstruction performs its microoperations concurrently. In the case, when the behavior of the system is described by a set of subcontrollers, two or more microinstructions originating from different subcontrollers can be activated simultaneously, and consequently, a certain couple y_i, y_j of microoperations are able "to co-exist". It can result in three ways: execute y_i and suppress y_j; execute y_j and suppress y_i; and execute both y_i and y_j in parallel. In general, all microoperations of a controller are members of one or more partial orders (*posets*) where y_i may be covered by one or more other microoperations $y_j,y_k,\ldots$. The specifications need to specify the *posets* representing the controller's desired functionality.

Indeed, the specifications include also action arbitration, or prioritization information. The definition of priority between an action y_i and several other actions $y_j,y_k,\ldots$ means that y_i conflicts with the subset $y_j,y_k,\ldots$, and that the conflict is resolved by explicitly specifying which of the actions will be executed. This type of specification, called *action-priority ATR*, states that action y_i is allowed to execute only when neither of the actions $y_j,y_k,\ldots$ have to be executed. Each action-priority ATR can be expressed as an inequality that has the following general format:

$$y_i > y_j. \qquad (14)$$

As has been said earlier, we can transform a priority, hence the action-priority ATR, into a partial order on the microoperations, where each of the actions $y_j,y_k,\ldots$ covers y_i:

$$y_j > y_i,\quad y_k > y_i,\quad \ldots \qquad (15)$$

The set of all action-priority ATRs for a controller comprises one or more partial order sets, or *posets*. This is demonstrated with an example in the "Case Study" section.

Comment: Kohavi uses the notation $\leq$ (as opposed to <).

Implicitly, any pair of microoperations can coexist, unless an action-priority ATR otherwise specifies.

All action-priority ATRs comprise the specifications of a subcontroller that takes care of the microoperation arbitration. The output of all other subcontrollers is the input for the arbitration subcontroller, comprising a two level architecture presented in Fig.2. The first level implements all regular ATRs in several subcontrollers. The second level implements the action-priority ATRs in a single arbitration subcontroller. At this point we can construct the actual subcontrollers from the corresponding sets of "regular" ATRs and action-priority ATRs.

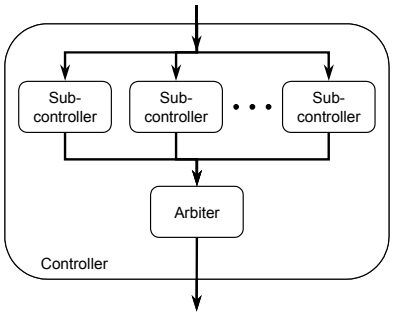

Fig. 2: The two levels of the microoperation-driven priority architecture. (Arrows indicate control signal flow.)

5 Case Study

Here we describe the above concepts through the process of constructing a working (albeit extremely simple) mobile robot system. The system consists of a controller composed of several subcontrollers and an operation unit. *Microsoft Excel* spreadsheets implement the controller [11]. A PC connected via *Lego Interface B* to *Lego Technique* components implement the operation unit [12]. The Microsoft Excel workbook is downloadable from [13].

In this case study we focus on the controller's implementation process, which is comprised of (i) specifications identification, using ATRs, (ii) ATRs translation into transition formulae, and (iii) executable decision tables' implementation in the Excel workbook. With a little experience, step (ii) can be skipped.

5.1 Specifications

We describe a mobile robot moving along a desk's surface in the direction of a light source, or beacon. The robot's goal is to reach as close as it can to the light source.

Assume that the robot has five sensors corresponding to binary signals (input variables) $x_1,\ldots,x_5$. Three of these signals, x_1, x_2 and x_3, correspond to light sensors positioned on the robots' front, left and right sides, respectively. Binary input variable x_4 corresponds to a sensor that detects the approach of the robot to the edge of the desk. The fifth sensor detecting the robots' arrival to the light source corresponds to variable x_5. Assume also that the robot is provided with a transport mechanism, which can be controlled by the following microoperations: y_1 - to move the robot forward; y_2 - to turn the robot to the left; y_3 - to turn the robot to the right; y_4 - to stop the robot; and microoperation y_5 sounds a buzzer mounted on the robot.

According to the approach suggested in this paper, the list of ATRs below comprises the example mobile robot's control specifications. Each ATR is followed by a corresponding transition formula or by a partial order, as appropriate:

ATR-1. When the beacon light source is to the left of the robot, the robot turns left. $F_1 = x_2 y_2 + \bar{x}_2 Y_0$

ATR-2. When the beacon light source is to the right of the robot, the robot turns right. $F_2 = x_3 y_3 + \bar{x}_3 Y_0$

ATR-3. While the robot does not sense any light source, it keeps turning to the left. $F_3 = \bar{x}_1 \bar{x}_2 \bar{x}_3 y_2 + (x_1 + x_2 + x_3) Y_0$

ATR-4. When the robot senses light from two opposite directions, left and right, it stops. $F_4 = x_2 x_3 y_2 + (\bar{x}_2 + \bar{x}_3) Y_0$

ATR-5. When the beacon light source is in front of the robot, the robot moves forward. $F_5 = x_1 y_1 + \bar{x}_1 Y_0$

ATR-6. The robot does not go beyond the desk's edge. $F_6 = x_4 y_4 + \bar{x}_4 Y_0$

ATR-7. When the robot reaches the light source, it stops and sounds a buzzer. $F_7 = x_5 y_4 y_5 + \bar{x}_5 Y_0$

ATR-8, ATR-9, ATR-10. Stopping has priority over any maneuver across the desk.

(In fact, this non-atomic specification statement encompasses three ATRs). $y_4 > y_1;\quad y_4 > y_2;\quad y_4 > y_3$.

ID	ATR	Light comes from the front (x_1)	Light comes from the left (x_2)	Light comes from the right (x_3)	Desk surface edge reached (x_4)	Contact with light source (x_5)	Move forward (y_1)	Turn left (y_2)	Turn right (y_3)	Stop (y_4)	Sound a buzzer (y_5)
1	When the beacon light source is to the left of the robot, the robot turns left.		1	0				1			
2	When the beacon light source is to the right of the robot, the robot turns right.		0	1					1		
4	When the robot senses light from two opposite directions (left and right), It stops.		1	1						1	

ATR-11, ATR-12. Turning toward the light source has priority over moving forward. (This non-atomic specification statement comprises two ATRs). $y_2 > y_1;\quad y_3 > y_1$

ID	ATR	x_1	x_2	x_3	x_4	x_5	y_1	y_2	y_3	y_4	y_5
3	While the robot does not sense any light source, it keeps turning left.	0	0	0				1			
5	When the beacon light source is in front of the robot, the robot moves forward.	1					1				

ID	ATR	x_1	x_2	x_3	x_4	x_5	y_1	y_2	y_3	y_4	y_5
6	The robot does not go beyond the desk's edge.				1					1	

ID	ATR	x_1	x_2	x_3	x_4	x_5	y_1	y_2	y_3	y_4	y_5
7	When the robot reaches the light source, it stops and sounds a buzzer.					1				1	1

Fig. 3: Four decision tables implementing ATRs number 1 through 7 in four subcontrollers

.

5.2 Controller Synthesis

The decision tables in Fig.3 represents the first layer of the mobile robot's control, obtained by implementing ATRs number 1 through 7. These decision tables do not need to be orthogonal relative to each other; the second, arbiter layer resolves any conflicts, as described below.

The mobile robot's control includes, in addition to the decision tables in Fig.3, also an arbitration function over the set Y of microoperations, as implied by ATRs number 8 through 12, above. The arbiter comprises the controller's second layer. The microoperations that are the output values of the above, first layer decision tables are the input values for the second layer decision table corresponding to the partial order $\{Y,<\}$.

From ATRs number 8 through 12, above, we construct the following partial order on the set of microoperations:

$$Y = \{y_1, y_2, y_3, y_4, y_5\}:\quad y_4 > y_2 > y_1,\quad y_4 > y_3 > y_1,\quad y_5. \tag{16}$$

We can illustrate the above *poset* with the partial order (or Hasse) diagram in Fig.4.

For each local conflict the arbiter chooses one of these alternatives. To dissolve the global conflict Y under the given family of priority constraints imposed by the partial order the arbiter has to choose only maximal elements of the set Y. Traversing the tree corresponding to the partial order Y, one can easily find all the maximal elements needed by an efficient polynomial algorithm [1].

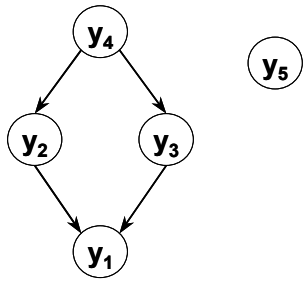

Fig. 4: The partial order diagram of priority constraints over the set of microoperations {y1,y2,y3,y4,y5}

		y_j				
		y_1	y_2	y_3	y_4	y_5
y_i	Move forward - y_1	1	0	0	0	-
	Turn left - y_2	1	1	-	0	-
	Turn right - y_3	1	-	1	0	-
	Stop - y_4	1	1	1	1	-
	Sound a buzzer - y_5	-	-	-	-	1

Fig. 5: The partial order matrix of the priority constraints over the set of microoperations {y1,y2,y3,y4,y5}

The above partial order is presented by its corresponding partial order matrix in Fig.5, as has been described in the "Formal Model of Controller Composition" section. In a partial order matrix A:

$$A_{ij} = \begin{cases} 1 & if \quad y_i > y_j \\ 0 & if \quad y_i < y_j \\ - & if \quad y_i = y_j \end{cases} \qquad (17)$$

Arbiter Input					Arbiter Output
y_1	y_2	y_3	y_4	y_5	
1	0	0	0	-	y_1 - Move forward
-	1	-	0	-	y_2 - Turn left
-	-	1	0	-	y_3 - Turn right
-	-	-	1	-	y_4 - Stop
-	-	-	-	1	y_5 - Sound a buzzer

Fig. 6: The poset decision table of the priority constraints over the set of microoperations {y_1,y_2,y_3,y_4,y_5}

Now our intent is to show a technique for transforming the above matrix of partial order on the set Y to the corresponding poset decision table. Every decision table row corresponds to an element of the set of outputs Y. The left part of the decision table columns (see Fig.6) corresponds to the same set Y as the partial order matrix (Fig.5), and is interpreted as a set of inputs to the decision table. The right part of the decision table columns corresponds to the set of outputs Y. To accomplish this transformation task we assign to the left part of the decision table all the corresponding values of the partial order matrix, except any non-diagonal "1", which is changed to "–".

The right part of the poset decision table (Fig.6) is the output microoperation. This decision table is an example for an arbitration subcontroller that comprises the second level in the microoperation-driven priority architecture (Fig.2). Note that there is no need for this decision table to be orthogonal, as the example below will demonstrate.

We demonstrate the decision table's use with an example. Suppose that at a certain point of time the following intermediate microoperations' vector is generated: $\{y_3,y_4,y_5\}$. This vector matches two of the decision table's rows: the rows with y_4 and y_5 in the output column. Hence, the controllers' output for this particular intermediate vector will be the resulting vector of microoperations $\{y_4,y_5\}$. Indeed, one would expect the microoperation y_4 ("stop") to override y_3 ("turn left").

6 Conclusions

In this paper we have described a design methodology for development of the control part of a system. This approach is based on the Controlware concept that combines a verbal system specification, a formal model of the specification and a method for easy implementation of the specifications. Since the control part of a system consists of a number of component subcontrollers working concurrently, a problem of their arbitration becomes essential. We have proposed to solve this problem by introducing a partial ordering relation on the set of possible microoperations, as the arbitration description. The arbiter depicts SMEs requirements thus providing a cheaper solution than explicitly defining operations

for any input conditions that have been left undefined.

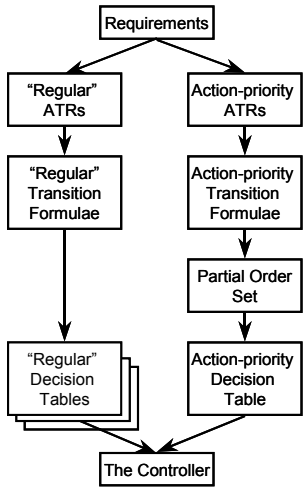

Fig. 7: The specification flow from requirements to implementation

Another significant contribution of the paper is the use of atomic requirements (ATR) as a high level, non-formal language of Controlware. Using ATRs as software specifications has several benefits for Controlware design. Designers can conveniently express their mental models' details in ATR statements. The ATRs, in turn can be transformed, one-to-one, into transition formulae. The translation of an ATR to a transition formula provides the smallest possible step between a natural language specification and a formal language specification, which is ready for direct implementation.

In the case of priority ATRs, the transition formulae transform into *poset*s, which transform into corresponding decision table rows. In the case of the "regular" ATRs, the transition formulae transform directly into the corresponding decision table rows. Finally, the controller can be synthesized from the decision tables (see Fig.7). Obviously, ATRs, as opposed to non-atomic requirement specifications, smoothly fit into the controller's formal model, because their transformation into transition formulae, and hence to decision table rows, is one-to-one.

We hope that the proposed approach can be considered as a universal methodology for designing logic control of controlled systems.

References:

[1] I. Levin and V. E. Levit, Controlware for Learning Using Mobile Robots, *Comp Sci Educ*, Vol.8, No.3, 1998, pp. 181-196.

[2] H. Salzer, ATRs (Atomic Requirements) Used Throughout Development Lifecycle, In: *Proceedings of Quality Week 1999*, San Jose, California, USA, 1999

[3] H. Salzer and I. Levin, Atomic Requirements in Teaching Logic Control Implementation, *International Journal of Engineering Education*, Vol.20, No.1, 2004.

[4] S. Baranov, *Logic Synthesis for Control Automata*, Kluwer Academic Press, 1994

[5] Z. Kohavi, *Switching and Finite Automata Theory*, McGraw-Hill, New York, 1970.

[*9] H. Salzer and I. Levin, Spreadsheet-based Logic Controller for Teaching Fundamentals of Requirements Engineering, *International Journal of Engineering Education*, (in press).

[7] E. Hamby, *Programs from Decision Tables*, MacDonald, London and American Elsener, New York, 1973

[8] H. Kilov and J. Ross, Contracts and Layers, In: *Information Modeling: An Object-oriented Approach*, Prentice-Hall, 1994, pp. 28-32.

[9] IEEE Std 1233, Guide for Developing System Requirements Specifications, *IEEE Standards, Software Engineering*, Vol.1, *Customer and Terminology Standards*, IEEE, Computer Society, 1998

[10] R. A. Brooks, A Robust Layered Control System for a Mobile Robot, *IEEE J of Robotics and Automation*, Vol.2, No.1, 1986, pp. 14-23.

[11] I. Levin, Matrix Model of Logical Simulator within Spreadsheet, *Int J Elect Eng Educ*, Vol.30, No.3, 1993, pp. 216-223.

[12] I. Levin, The State Machine Paradigm and the Spreadsheet Learning Environment, In: Smith AJ (ed), *Engineering Education, Increasing Students Participation*, Sheffield Hallam University, Sheffield, UK, 1994, pp. 351-355.

[13] http://www.tau.ac.il/~salzerha/demos/ArbiterCaseStudy.xls