A Technique for Specifying Interface Modules for Real-Time Systems

Yingzi Wang and Dennis K. Peters

Electrical and Computer Engineering
Faculty of Engineering and Applied Science
Memorial University of Newfoundland
St. John's, Newfoundland
Canada
E-mail: wang@engr.mun.ca, dpeters@engr.mun.ca

Abstract

Documentation plays a key role as a component of design process, and a preview of a task before it comes to be executed. A well-specified task might not take less implementation time than one without documents, but one of the obvious advantages is that misunderstandings are avoided and readable specification makes it easy for the successive developers to exploit or modify the software or hardware design. Such merit is particularly useful for aviation and military applications in which reliability and maintainability are very important aspects for judging the success of a project.

Interface Modules (IM) are modules that encapsulate input or output device hardware and the related software, so that the application software can be written without specific knowledge of the particular devices used. Replacing or modifying an interface device will only lead to changes in the IM, rather than changing the other modules in the whole system. In real-time and embedded systems, an IM will often relate real-valued external quantities (e.g., time, positions in space) with discrete valued software quantities. An IM specification must therefore use a combination of notations and formalisms.

In this paper, we present a technique for IM specification that is an extension of the System Requirements Documentation technique presented in [1], which is based on the Software Cost Reduction (SCR) method. An IM is specified as a "sub-system" that interacts with both the external environment and other software modules in the larger system. The interface quantities are modeled as functions of time and the behaviour is described in terms of conditions, events and mode classes. This technique facilitates concise and formal description of the module behaviour, including tolerances and delays.

Keywords: *device interface module, module specification, real-time system*

1 INTRODUCTION

Documentation is an essential product of the design process, and plays several key roles. For example,

- a requirements specification can provide an overview of the system before it is implemented;
- a complete and precise system design document can be used to help determine the feasibility of the system, or to verify other essential system properties;
- accurate and precise documentation improves maintainability and reliability by acting as a guide and reference for current and future developers; and
- precise specifications can be used to help to detect, isolate and remove faults earlier in the project lifecycle, which reduces costs.

Inadequate documentation causes software quality to degrade over time because the changes are inconsistent with the original(undocumented) design concept; such changes result in unnecessarily complex programs.

To reduce the complexity of the system, a system can be decomposed into a set of modules, each of which performs a certain task in the system.[2] These modules can be implemented by individual developers without communicating with others very often, for the task is defined explicitly in the documents of the module specification. A well-specified task might not take less implementation time than one without documents, but one of the obvious advantages is that misunderstandings are avoided and readable specification makes it easy for the successive developers to exploit or modify the software or hardware design. Such merit is particularly useful for example in aviation and military applications which consider reliability and maintainability very important aspects judging the success of a project.

A real-time system is a system that must produce its results within specified time intervals. Common examples of real-time systems include flight control programs, patient monitoring systems, and weapons systems. To be acceptable the behaviour of such systems must not only to be functionally correct, but also be temporally correct—satisfying the timing constraints.

1.1 Interface Modules

Interface Modules (IM) are modules that communicate between the application software and the environment in which it operates.[3, 4] They typically encapsulate software and hardware that implement an interface with either human users (i.e., a human-machine interface) or some other environmental quantity (e.g., temperature, robot arm position). Since environmental quantities may be either continuous or discrete, but software quantities are discrete, the IM is classed as a "hybrid" system.

An interface module reduces the complexity of the system design by isolating the interface details from the rest of the system software. This is particularly important in embedded systems, where the IM will often contain special purpose hardware devices (e.g., actuators or sensors): replacing or modifying a device should only lead to changes in the IM, rather than requiring changes to other modules in the system. If interface hardware is not explicitly encapsulated, when a device changes, programs depending on it will also need to change, so the change could have surprising and widespread ramifications.

Consider, for example, a system for making signs that uses a robotic arm as illustrated in Figure 1. As illustrated, the system consists of three modules (each of which may be sub-divided into other modules): User interface, processing, and robot interface. The User interface and robot interface modules are both examples of interface modules, which isolate the processing software from the specific details of the input or output device hardware and software. If, for example, the mechanical properties of the robot arm were to be modified, it would probably require that the software controlling it also change. The robot interface module limits the impact of these changes.

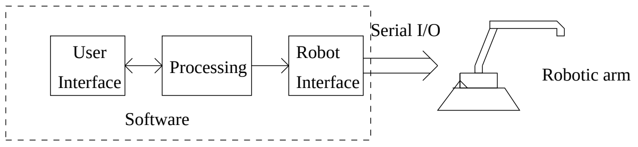

Figure 1. Robot System

Form	Meaning
$^{c}\mathsf{x}$	Controlled variable
$^{m}\mathsf{x}$	Monitored variable
$^{Md}\mathsf{x}$	Mode
$^{Cl}\mathsf{x}$	Mode class
$^{C}\mathsf{x}$	Constant
$^{p}\mathsf{x}$	Condition

Table 1. Identifier Annotations

The ideal Interface Module will:

- be the only component that needs to change if the devices change;
- not need to change unless the devices change;
- be relatively small and simple structured so that it can be easily changed if necessary.

Section 3 is a specification of an interface module for controlling a robotic arm, similar to that illustrated in Figure 1. The arm has five motors to position the tip and open or close the 'hand', and is controlled by software on a PC via a serial link. For this illustration we will consider the tip position in two dimensions only, $^{c}\mathsf{armPos}$ (the position on a drawing surface), and a Boolean, $^{c}\mathsf{armUp}$, to represent if the tip is above the surface or touching it.

Throughout this paper we use superscripted annotations as described in Table 1 to help clarify the meaning of identifiers.

2 SPECIFYING INTERFACE MODULES

Interface Module Specifications (IMS) are components of the System Design Document, as described in [1, 5]. Each treats a module as a "black box", identifying those programs that can be invoked from outside of the module(access programs), and describing the externally-visible effects of using them. Like other module specifications, the IMS tells the module designer what behaviour is required of the module, and allows it to be implemented without communicating with other module designers. Also the IMS can be used to verify that the module internal design is correct. Designers of other modules in the system can use the IMS to understand what behaviour they can expect from the module. As a part of system design process, the IMS and the system architecture can be used to verify that the design satisfies the system requirements.

Since interface modules interact with both other software modules and the environment external to the system, they are examples of hybrid systems, which contain both discrete and continuous components. Thus, they present new challenges for specification. Previous work [6], outlines a method for specifying interface modules by extending some of the notation for System Requirements documentation presented in [1] to allow the IM to be viewed as a *system* in the sense of that work. This paper extends [6] to consider real-time behaviour of the interface module.

2.1 Module Interface

As stated in [1] *environmental quantities* are quantities that are external to the system, “independent of the chosen solution and are apparent to the ‘customer’.” From the point of view of the IM, the ‘customer’ is the designer of the software that will use the IM to communicate with the external environment, and the quantities of interest are both internal (software) and external quantities. The internal quantities are software quantities that form the interface between the IM and other system modules, including, for example, parameters to access programs. The external quantities are the environmental quantities relevant to the system and represent such things as temperature, switch settings, or the position of a robot arm. All these quantities can be represented by functions of time. Note that for real-time systems, time, itself, is a relevant environmental quantity.

The IMS must describe the behaviour of the IM in terms of these quantities. We divide the quantities into two, not necessarily disjoint, sets: the *controlled quantities* are those that the IM may change the value of, and the *monitored quantities* are those whose value may effect the current or future behaviour of the IM. The IMS, then, must give the value of the controlled quantities depending on the current and past values of the monitored quantities. The interface to the robotic arm module is given in Section 3.1. Note that, implicitly, the input parameters of access programs are monitored quantities, and output parameters and return values are controlled quantities.

2.2 Conditions, Events and Mode Classes

The relevant properties of the monitored and controlled quantities can often be succinctly characterized by predicates, called *conditions*, which are Boolean functions of time defined in terms of the monitored and controlled quantities. For an interface module access program, we use the access program name and parameters to denote the condition that is true only when the access program is executing. For example, if `foo` is an access program then $\texttt{foo}(\texttt{x})$ is true if and only if `foo` is executing, and $\texttt{foo}(\texttt{x}) \wedge \texttt{x} < 0$ is true if and only if `foo` is executing and its parameter was less than 0 when it was called.

The instants when conditions change value are significant to the behaviour of the system, and these instants are referred to as *events*. Formally, an event e, is a pair (t,c), where e.t is a time at which one or more conditions change value and e.c denotes the status (i.e., true, false, becoming true, becoming false — denoted $\mathrm{T}, \mathrm{F}, @\mathrm{T}, @\mathrm{F}$, respectively) of all conditions at e.t. For real-time systems, the amount of time between events will be relevant to the module behaviour.

The history that is relevant to the behaviour of a module can thus be described by the initial conditions and the sequence of events that have occurred since the initial state. It is often the case that many histories are the same with respect to current and future behaviour, so we group these together into a *mode*. A set of modes that partition the possible histories — forming an equivalence relation on the set of histories — is known as a *mode class*. If the behaviour is specified for every mode in a mode class, then it is fully specified.

Sections 3.2 and 3.3 describe the mode classes for the robotic arm module. Mode class ${}^{Cl}\mathsf{Motion}$ relates to the motion of the robot arm. The initial mode is ${}^{Md}\mathsf{uninitialized}$. When the event $@\mathrm{T}\,(\texttt{moveInitialPos}())$ occurs (i.e., the access program is called), the IM enters the mode ${}^{Md}\mathsf{movingTo}({}^{C}\mathsf{X_i}, {}^{C}\mathsf{Y_i}, \underline{\mathit{true}})$—the arm is moving toward its initial position. Similarly, $@\mathrm{T}\,(\texttt{moveLinear}(x, y, u))$ initiates movement towards the postion (x, y, u). When the arm reaches its destination, $@\mathrm{T}\,({}^{p}\mathsf{onPosition}(x, y, u))$ occurs and the IM enters mode ${}^{Md}\mathsf{stopped}(x, y, u)$—it is stopped at that position.

Mode class ${}^{Cl}\mathsf{Gripper}$ relates to the opening and closing of the gripper. The mode change is initiated by either $@\mathrm{T}\,(\texttt{graspGripper}())$ or $@\mathrm{T}\,(\texttt{releaseGripper}())$, to close or open the gripper, respectively. The mode changes to ${}^{Md}\mathsf{grasped}$ or ${}^{Md}\mathsf{released}$ when the access program returns ($@\mathrm{F}\,(\texttt{graspGripper}())$ or

@F (`releaseGripper()`))—indicating that the call will not return until the gripper has completed the operation.

2.3 Controlled Value Functions

The behaviour of the IM is thus described by giving the values of the controlled quantities in terms of the history relevant to the module. The characteristic predicate of the acceptable values of controlled values is given using the standard predicate and relational operators and tabular expressions. These are expressed in terms of the previous behaviour, current mode in one or more mode classes, and condition values.

In Section 3.5 the value of controlled quantities are specified in terms of each mode in the relevant mode class. The controlled quantities ${}^{c}\mathsf{armPos}$, ${}^{c}\mathsf{armUp}$ are defined in terms of ${}^{Cl}\mathsf{Motion}$, and ${}^{c}\mathsf{gripPres}$ is defined in terms of and ${}^{Cl}\mathsf{Gripper}$.

2.4 Timing Constraints

In real-time systems, time relative to some initial time is always a monitored variable, and the elapsed time between some events will be constrained by the specification. For a condition p, $Drtn(p,t)$ denotes the duration of time that p has been continuously true until current time t.

In Section 3.5, the amount of time that the arm is permitted to take to reach its destination is constrained by the expression $Drtn({}^{Md}\mathsf{movingTo}(x,y,u)) \leq {}^{C}\mathsf{MOVE\_TIME}$ in the first table. Thus the real-time requirements on this module are specified.

3 EXAMPLE: ROBOT ARM MODULE

3.1 Module Interface

Access Programs

Name	Parameter Types
`moveInitialPos`	
`moveLinear`	**int**, **int**, **Boolean**
`graspGripper`	
`releaseGripper`	

Environmental Quantities

Variable	Description	Value Set
${}^{m}\mathsf{t}$	current time	**Real**
${}^{c}\mathsf{armPos}$	position of the arm tip(x, y in mm)	**Real** × **Real**
${}^{c}\mathsf{armUp}$	true if the tip is not touching the surface	**Boolean**
${}^{c}\mathsf{gripPres}$	pressure applied by the gripper (Pa)	**Real**

3.2 Mode Class ${}^{Cl}\mathsf{Motion}$

Modes : ${}^{Md}\mathsf{uninitialized}$, ${}^{Md}\mathsf{movingTo}(x,y,u)$, ${}^{Md}\mathsf{stopped}(x,y,u)$

Initial Mode : ${}^{Md}\mathsf{uninitialized}$

Transition Relation :

$\mathrm{p_T} : H_1 \wedge G$ $\mathrm{r_T} : H_3$ Decision	`moveInitialPos()`	${}^{p}\mathsf{onPosition}(x,y,u)$	`moveLinear`(x,y,u)	
${}^{Md}\mathsf{uninitialized}$	@T	$*$	$*$	${}^{Md}\mathsf{movingTo}$ $({}^{C}\mathsf{X_i}, {}^{C}\mathsf{Y_i}, \underline{true})$
${}^{Md}\mathsf{movingTo}(x,y,u)$	$*$	@T	$*$	${}^{Md}\mathsf{stopped}(x,y,u)$
${}^{Md}\mathsf{stopped}(x,y,u)$	$*$	$*$	@T	${}^{Md}\mathsf{movingTo}$ (x,y,u)

Maximum Delay : ${}^{c}\mathsf{RT\_MOVING}$

3.3 Mode Class ${}^{Cl}\mathsf{Gripper}$

Modes : ${}^{Md}\mathsf{grasping}$, ${}^{Md}\mathsf{grasped}$, ${}^{Md}\mathsf{releasing}$, ${}^{Md}\mathsf{released}$

Initial Mode : ${}^{Md}\mathsf{released}$

Transition Relation :

Mode	Event	New mode
${}^{Md}\mathsf{released}$	@T (`graspGripper()`)	${}^{Md}\mathsf{grasping}$
${}^{Md}\mathsf{grasping}$	@F (`graspGripper()`)	${}^{Md}\mathsf{grasped}$
${}^{Md}\mathsf{grasped}$	@T (`releaseGripper()`)	${}^{Md}\mathsf{releasing}$
${}^{Md}\mathsf{releasing}$	@F (`releaseGripper()`)	${}^{Md}\mathsf{released}$

Maximum Delay : ${}^{c}\mathsf{RT\_GRIPPER}$

3.4 Conditions

$${}^{p}\mathsf{onPosition} : \mathbf{Real} \times \mathbf{Real} \times \mathbf{Boolean} \rightarrow \mathbf{Boolean}$$

$${}^{p}\mathsf{onPostion}(x,y,u)$$
$$\stackrel{\mathrm{df}}{=} |{}^{c}\mathsf{armPos.x} - x| < {}^{C}\epsilon \wedge$$
$$|{}^{c}\mathsf{armPos.y} - y| < {}^{C}\epsilon \wedge$$
$${}^{c}\mathsf{armUp} = U$$

3.5 Controlled Value Functions

$(^c\mathsf{armPos}, {}^c\mathsf{armUp})$

$\mathrm{p_T} : H_1$ $\mathrm{r_T} : H_2\ G$ Vector	$^c\mathsf{armPos}\ \vert$	$^c\mathsf{armUp} =$
$^{Md}\mathsf{uninitialized}$	$\vert\frac{d}{dt}(^c\mathsf{armPos})\vert = 0$	$\underline{true}$
$^{Md}\mathsf{stopped(x,y,u)}$	$\vert\frac{d}{dt}(^c\mathsf{armPos})\vert = 0$	u
$^{Md}\mathsf{movingTo(x,y,u)}$	$\frac{d}{dt}(\vert{}^c\mathsf{armPos} - (x,y)\vert) < 0 \wedge$ $Drtn(^{Md}\mathsf{movingTo}(x,y,u)) \leq {}^C\mathsf{MOVE\_TIME}$	u

$^c\mathsf{gripPres}$

$\mathrm{p_T} : H_1$ $\mathrm{r_T} : H_2 \mid G$ Vector	$^c\mathsf{gripPres}$
$^{Md}\mathsf{releasing}$	$\frac{d}{dt}(^c\mathsf{gripPres}) < 0$
$^{Md}\mathsf{released}$	$\frac{d}{dt}(^c\mathsf{gripPres}) = 0 \wedge {}^c\mathsf{gripPres} = 0$
$^{Md}\mathsf{grasping}$	$\frac{d}{dt}(^c\mathsf{gripPres}) > 0$
$^{Md}\mathsf{grasped}$	$\frac{d}{dt}(^c\mathsf{gripPres}) = 0 \wedge {}^c\mathsf{gripPres} \geq {}^C\mathsf{GRIP\_PRES}$

3.6 Constants

Constant	**Description**	**Range**
$^C\mathsf{X_i}$	Home x position (mm)	$(-10, 10)$
$^C\mathsf{Y_i}$	Home y position (mm)	$(0, 20)$
$^C\epsilon$	Tolerance on positions (mm)	$(0, 2)$
$^C\mathsf{RT\_MOVING}$	Maximum delay on mode class $^{Cl}\mathsf{Motion}$ (s)	$(5, 10)$
$^C\mathsf{RT\_GRIPPER}$	Maximum delay on mode class $^{Cl}\mathsf{Gripper}$ (s)	$(0, 5)$
$^C\mathsf{GRIP\_PRES}$	Pressure applied by the gripper (Pa)	$(15, 18)$
$^C\mathsf{MOVE\_TIME}$	Maximum moving duration (s)	$(30, 35)$
$^C\mathsf{ARM\_RATE}$	Maximum moving rate (mm/s)	$(5, 8)$

3.7 Environmental Constraints

The following assumption can be made for environmental quantities in addition to the range limits on the constant values given in the above table.

$\left|\frac{d}{dt}(^c\mathsf{armPos})\right| \leq {}^C\mathsf{ARM\_RATE}$

4 CONCLUSIONS AND FUTURE WORK

This paper has briefly illustrated a technique for interface module specification using notation similar to that presented in [1]. The use of events and mode classes provides a foundation for concise descriptions of the required behaviour. Since events are instants, we can express real-time aspects of the behaviour using simple constraints on the time elapsed between events.

Application of these techniques to the specification of other interface modules will further illustrate their usefulness and may allow us to draw more conclusions on specifying real-time systems.

Acknowledgments

We are thankful to the our colleagues in Electrical and Computer Engineering at Memorial University, and in particular to An Zhiwei for many insightful discussions. The Natural Sciences and Engineering Research Council and the Faculty of Engineering at Memorial University have supported this work.

References

[1] D. K. Peters, *Deriving Real-Time Monitors from System Requirements Documentation.* PhD thesis, McMaster University, Hamilton ON, Jan. 2000.

[2] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *Communications ACM*, pp. 1053–1058, Dec. 1972.

[3] D. L. Parnas, "Use of abstract interfaces in the development of software for embedded computer systems," NRL Report 8047, Naval Research Laboratory, June 1977.

[4] K. H. Britton, R. A. Parker, and D. L. Parnas, "A procedure for designing abstract interfaces for device interface modules," in *Proc. Int'l Conf. Software Eng. (ICSE)*, pp. 195–204, 1981.

[5] D. L. Parnas and J. Madey, "Functional documentation for computer systems," *Science of Computer Programming*, vol. 25, pp. 41–61, Oct. 1995.

[6] Y. Wang and D. K. Peters, "Interface module specifications for real-time systems," in *Newfoundland Electrical and Computer Engineering Conference (NECEC)*, IEEE, Newfoundland and Labrador Section, Nov. 2001.