Interface Module Specifications 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
wang@engr.mun.ca, dpeters@engr.mun.ca

November 13, 2001

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. This paper presents a suitable method for IM specification, which is both precise and readable.

1 Introduction

Design 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 life-cycle, which reduces costs.

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.[1] 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

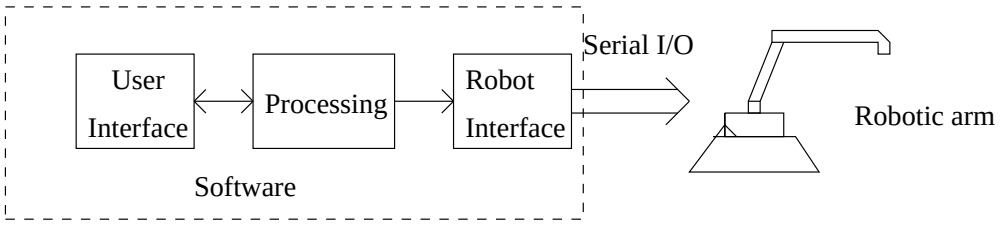

Figure 1: Robot System

useful for example in aviation and military applications which consider reliability and maintainability very important aspects judging the success of a project.

Interface Modules (IM) are modules that communicate between the application software and the environment in which it operates.[2, 3] They typically encapsulate software and hardware that implements an interface with either human users (i.e., a human-machine interface) or some other environmental quantity (e.g., temperature, robot arm position).

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.

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.

2 Specifying Interface Modules

Interface Module Specifications (IMS) are components of the System Design Document, as described in [4, 5]. 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 present new challenges for specification. In this section we briefly outline a method for specifying interface modules that extends some of the notation for System Requirements documentation presented in [4] to allow the IM to be viewed as a *system* in the sense of that work.

2.1 Module Interface

As stated in [4] *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 past values of the monitored 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 `foo(x)` is true if and only if `foo` is executing, and $\texttt{foo(x)} \wedge \texttt{x} < 0$ 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.

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.

3 Example: servoPositioning Package

The servoPositioning package is an interface module for 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 only the tip position in two dimensions, ${}^{c}\mathsf{armPos}$ (the position on a drawing surface), and a Boolean, ${}^{c}\mathsf{armUp}$, to represent if the tip is above the surface or touching it.

3.1 Module Interface

Access Programs

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

Environmental Quantities

Variable	Description	Monitor	Control	Value Set	Notes
${}^{m}\mathsf{t}$	current time	X		**Real**	1
${}^{c}\mathsf{armPos}$	position of the arm tip		X	**Real** $\times$ **Real**	2
${}^{c}\mathsf{armUp}$	true if the tip is not touching the surface		X	**Boolean**	

Notes:

1. Elapsed time measured from some arbitrary fixed time before the system is started.
2. Arm tip position is measured, in mm, in the plane containing the drawing surface with origin at the corner of the drawing surface closest to the robot arm base.

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 :

Mode	Event	New mode
${}^{Md}\mathsf{uninitialized}$	@T (`moveInitialPos`())	${}^{Md}\mathsf{movingTo}({}^{C}\mathsf{X_i}, {}^{C}\mathsf{Y_i}, \underline{true})$
${}^{Md}\mathsf{movingTo}(x, y, u)$	@T (${}^{p}\mathsf{onPosition}(x, y, u)$)	${}^{Md}\mathsf{stopped}(x, y, u)$
${}^{Md}\mathsf{stopped}(x, y, u)$	@T (`moveLinear`(x, y, u))	${}^{Md}\mathsf{movingTo}(x, y, u)$

3.3 Conditions

${}^{p}\mathsf{onPosition} : \mathbf{Real} \times \mathbf{Real} \times \mathbf{Boolean} \rightarrow \mathbf{Boolean}$
${}^{p}\mathsf{onPostion}(x, y, u)$
$\quad \stackrel{\text{df}}{=} |{}^{c}\mathsf{armPos.x} - x| < {}^{C}\epsilon \wedge |{}^{c}\mathsf{armPos.y} - y| < {}^{C}\epsilon$

3.4 Constants

Constant	Description	Range
${}^{C}\mathsf{X_i}$	'home' x position	$(-10, 10)$
${}^{C}\mathsf{Y_i}$	'home' y position	$(0, 20)$
${}^{C}\epsilon$	Tolerance on positions	$(0, 2)$

3.5 Controlled Value Functions:

	${}^{Md}\mathsf{uninitialized}$	${}^{Md}\mathsf{stopped}(x, y, u)$	${}^{Md}\mathsf{movingTo}(x, y, u)$
${}^{c}\mathsf{armPos}$ |	$\left\lvert \frac{d}{dt}({}^{c}\mathsf{armPos}) \right\rvert = 0$	$\left\lvert \frac{d}{dt}({}^{c}\mathsf{armPos}) \right\rvert = 0$	$\frac{d}{dt}\left(\lvert {}^{c}\mathsf{armPos} - (x, y) \rvert\right) < 0$
${}^{c}\mathsf{armUp} =$	$\underline{true}$	u	u

4 Conclusions and Future Work

This paper has briefly illustrated a technique for interface module specification using notation similar to that presented in [4]. The use of events and mode classes provides a foundation for concise descriptions of the required behaviour.

Further investigation will focus on the issues concerning the hybrid nature of interface modules: the quantities are both discrete and continuous in nature.

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.

References

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

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

[3] 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.

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

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