A Tale of Two Reference Models

Alwyn E. Goodloe

So What is a Reference Model?

A division of functionality together with data flow between the pieces.

Examples:

- ISO 7 layer model for protocols.
- The model for compilers found in the Dragon book.

We shall look at two such models for software development,

What should such a model consist of? How about:

- Artifacts such as documents and programs that must be produced.

- Guide to what each artifact should contain.

- A set of relations between the artifacts which form a refinement theory.

- Those relationships should be expressed as proof obligations. They could be discharged by formal proof, model checking or less formally by walkthroughs and reviews.

We shall look at two such models.

- The Four Variable Model of David Parnas.
- Reference Model of Gunter,Gunter, Jackson and Zave.

The Four Variable Model is somewhat more comprehensive in that it specifically mentions the documents that must be produced throughout the development process.

Some basic assumptions that the models have in common (taken from Parnas):

- The Almost Right software that the proponents of *Good Enough Software* (virtually all the commercial industry) advocate is **WRONG**.
- Programmers must be told what the software should do.
- Artifacts should be subject to reviews and walkthroughs.
- Imprecise notations for describing software requirements almost always leads to almost right software.
- Notations must be mathematically based. (The next talk will give an in depth overview of one such notation - SCR.)

A Running Example: Patient Heart Monitor

- Given a patient in a hospital. We would like to build a device which monitors the patient's heart and notifies the nurse if it stops beating.

- We can't measure the heartbeat directly. But we can listen for the sound of the heart beat.

- There is a computer which has as devices:

  – To listen for the patient's heartbeat. This is clearly an analog measuring device so some signal processing (filtering or whatever those guys do) but we will not be concerned with those details here.

  - Alarm which should be sounded by the computer when the patient's heart has stopped beating.

- At various points in this talk we shall apply the two reference models to this examples.

- We shall not be fully formal but only use it to illustrate how a problem can be decomposed using the model as a guideline.

Four Variable Reference Model

Specifies a series of documents that need to be produced as part of the software development lifecycle.

- System Requirements Specification - Treats the computer system as a black box. It describes the environment and identifies a set of quantities that are of concern to systems users. Describes the relationship between these quantities as well as documenting any constraints on the values of these environmental variables.

- System Design Document - Identifies the computers within the system, how they communicate and describes the peripheral devices. The Input and output registers are denoted as variables. The relationship between these and the environment variables.

$$\text{Software Requirements} = \text{Sys. Design Doc} + \text{Sys. Requirements Doc.}$$

We shall give give more details on the formalisms proposed for representing the various relationship between the entities in these documents below.

- Software Behavior Specification - This document is needed when externally distinguishable s/w products would satisfy the system requirements. It records actual systems behaviour and design decisions.

- Software Module Guide - Describes the decomposition of the system into modules.

- Module Interface Specification - We view the module as implementing one or more finite state machines (objects). This description of the module interface is a blackbox description of these objects.

- Uses Relation Document - A table representing the relation (P, Q)- P uses Q, where P and Q are the public access methods of a module.

- Module Internal Design Document - A clear-box specification of each module's implementation.

- Data Flow Document - View software as a set of concurrent communicating processes. Each process transforms an input sequence into an output sequence.

Black-Box Module Interfaces

If the module is wholly internal, we can describe the history via *traces*.

Identify a set of *Canonical Traces*. Trace Assertions Specifications - comprise three relations:

- Fns, with domain - a set of pairs (canonical traces,event), range - set of canonical traces. $((T_1, e), T_2) \in f$ if T_2 is T_1 extended by e.

- Relations which associate each canonical trace with an output function.

- Functions which define the information returned.

Internal Module Design

- A complete description of the data structures.

- An *abstraction function* relating the data structures and events to traces and events.

- A *Program Function* specifying each method in terms of mappings from states before to after execution. Example - traditional Hoare assertions.

Back to Requirements
Environment and Systems Variables

- Monitored quantities $(m_1 \ldots m_p)$ - Those environmental variables that the system monitors.

- Controlled quantities $(c_1 \ldots c_q)$- Those environmental variables controlled by the system

- Environment variables can be both monitored and controlled.

- Input Variables $(i_1 \ldots i_r)$ - The values in the input registers.

- Output Registers $(o_1 \ldots o_s)$ - The values in the output registers.

Note: Environmental variables have a function that is recorded as a function of time, v^t, which has domain $\mathcal{R}$ and $v^t(s)$ is the value at time s.

Note: $\underline{v}^t \stackrel{\text{def}}{=} (v_1^t, \ldots, v_p^t)$

Example decomposed into the variables of the Four Variable Model:

- m - Sounds from patient's chest.
- c - Buzzer at nurse station.
- i - Value read from sensor.
- o - Value to write to the buzzer.

We document the System via the following five relations.

- NAT - Constraints on the environment variables placed by the environment.
  - $\mathcal{DOM}$(Nat) $-\, \underline{m}^t$ values allowed by environment.
  - $\mathcal{RAN}$(Nat) $-\, \underline{c}^t$ values allowed by environment.
  - $(\underline{m}^t, \underline{c}^t) \in$ Nat iff Nature permits.

- Req - Requirements. Reflects the further constraints imposed on environment variables by a correctly working program.
  - $\mathcal{DOM}$(Req) - Values of $\underline{m}^t$.
  - $\mathcal{RAN}$(Req) -Values of $\underline{c}^t$ allowed by a correctly working system.

- $(\underline{m}^t, \underline{c}^t) \in$ Req iff the system should permit that behavior.

- In - Input values when a measured value is read.
  - $\mathcal{DOM}$(In) - Values of $\underline{m}^t$.
  - $\mathcal{RAN}$(In) - Values of $\underline{i}^t$.
  - $(\underline{m}^t, \underline{i}^t) \in$ In iff $\underline{i}^t$ is a value in input register when $\underline{m}^t$ is read.

- Out - Output values resulting in a controlled value being set.
  - $\mathcal{DOM}$(Out) - Values of $\underline{o}^t$.
  - $\mathcal{RAN}$(Out) - Values of $\underline{c}^t$.
  - $(\underline{o}^t, \underline{c}^t) \in$ Out iff $\underline{c}^t$ is a possible value when the output register is $\underline{o}^t$.

- SOF - The program output given an input.
  - $\mathcal{DOM}$(SOF) - Values of $\underline{i}^t$.
  - $\mathcal{RAN}$(SOF) -Values of $\underline{o}^t$.
  - $(\underline{i}^t, \underline{o}^t) \in$ SOF iff The software produces $\underline{o}^t$ given $\underline{m}^t$.

Why aren't all these relations functions

- Nat - We wouldn't be able to vary the controlled variable w/o effecting the monitored variable.

- Req - One can tolerate small errors in the value of the controlled quantities.

- In - Imprecision in measurement and transducer.

- Out- Device imperfections.

- SOF- Is a deterministic function as you would expect.

Decomposing the Patient Monitoring System

- Nat - Would describe the range of sounds detected or nondetected by the sensor and the possible range of values of the actuator controlling the buzzer.

- Req - A warning that notifies the nurse if the system detects heart stops beating. The document would formalize the statement: *if the sound being monitored falls below a certain threshold, then the system sound the buzzer.*

- In - The input registers holding the data read from the sensor monitoring the sounds of the heart beat.

- Out - The output registers which are read by the actuator that can sound the buzzer.

- SOF - If the input register doesn't show signs of a heart beat for more than some specified time then the output register indicates the alarm to ring.

- Parnas' model is strongly influenced by control applications.

- Examples: A7 avionics, Darlington Nuclear Reactor Shutdown system.

- This may seem too limited for many non-control applications.

- For non-realtime systems some of this model can be simplified.

- For any application it is helpful to formalize the environment and the system and their interaction.

Proof Obligations

Here Comes the Math

Feasibility:

$$\begin{array}{l} \forall \underline{m}^t.(\exists \underline{c}^t.\mathsf{NAT}(\underline{m}^t, \underline{c}^t)) \;\; \Rightarrow \\ \qquad (\exists \underline{c}^t.\mathsf{NAT}(\underline{m}^t, \underline{c}^t) \;\; \wedge \;\; \mathsf{Req}(\underline{m}^t, \underline{c}^t)) \end{array}$$

In must Handle all cases possible under Nat:

$$\forall \underline{m}^t.(\exists \underline{c}^t.\mathsf{Nat}(\underline{m}^t, \underline{c}^t)) \Rightarrow (\exists i.\mathsf{In}(\underline{m}^t, \underline{i}^t))$$

Acceptability:

$$\begin{array}{rcl} \forall \underline{m}^t \; \underline{i}^t \; \underline{o}^t \; \underline{c}^t.[\mathsf{In}(\underline{m}^t, \underline{i}^t) & \wedge & \mathsf{SOF}(\underline{i}^t, \underline{o}^t) \wedge \\ \mathsf{Out}(\underline{o}^t, \underline{c}^t) & \wedge & \mathsf{Nat}(\underline{m}^t, \underline{c}^t) \\ & \Rightarrow & \mathsf{Req}(\underline{m}^t, \underline{c}^t) \;] \end{array}$$

When an antecedent term governing behavior fails, then a devise is broken and the s/w can not be required to satisfy the requirements.

WRSM Reference Model

- WRSM model also rigidly separates the environment control from the system control in problem analysis.

- Each artifact in the reference model reflects how the division between the system and the environment is viewed in that artifact.

- who controls what (environment or system).

- This model is not as detailed as the Four Variable model.

- Some differences between this and the Four Variable model may be viewed as improvements.

Six Artifacts

- W - Domain knowledge of the environment. Restricts the values of these variables.

- R - Requirements that indicate what the customer wants in terms of the environment.

- S - A specification, detailed enough for the programmer.

- P - A program implementing the specification.

- M - The programming platform.

- Designated Terminology - Provides names to describe the application domain (environment) and programming platform for software development and the interfaces between them.

Decomposing the Patient Monitoring System:

- R - A warning that notifies the nurse if the heart stops beating.

- W -There is always a nurse near enough to hear the buzzer. A heart which beats gives off detectable sound.

- S - Of the sound being monitored falls below a certain threshold, then the system sound the buzzer.

- P - The program which implements S.

- M - Program platform with a sensor which detects sound in patients chest and a programmable actuator that can sound the buzzer.

Classes of Phenomena

- e = belongs to environment and controlled by it.
  - e_h - environment variables hidden from the system.
  - e_v - environment variables visible to the system.

- s = belongs to system and controlled by it.
  - s_h - system variables hidden from the environment.
  - s_v - system variables visible to the environment.

$$s \cap e = \emptyset$$

- W, R - uses e_h, e_v, s_v.

- P, M - uses e_v, s_v, s_h.

- S - uses e_v, s_v.

The WRSPM variables applied to the patient monitoring example:

- e_h - Nurse and heart beat of patient.
- e_v - Sounds from patient's heart.
- s_v - Buzzer at nurse's station.
- s_h - Internal representation of data from sensor.

Proof Obligations (Environment System)

Consistency of domain knowledge to ensure nontrivality:

$$\exists \ e \ s.\mathsf{W}$$

Adequacy - the requirement allows all events that the environment (e_h, e_v) and the system (s_h, s_v) can perform simultaneously:

$$\forall \ e \ s.\mathsf{W} \wedge \mathsf{M} \wedge \mathsf{P} \Rightarrow \mathsf{R}$$

Relative Consistency - Any choice of environment variable visible to the system is consistent with $\mathsf{M} \wedge \mathsf{P}$ if it is consistent with our assumptions about the environment:

$$\forall e_v.(\exists e_h.\mathsf{W}) \Rightarrow (\exists \ e_h \ s.\mathsf{W} \wedge \mathsf{M} \wedge \mathsf{P})$$

Specifications

- Specification - Filters out the knowledge of W and R that the people who develop P must have to write the program.

- If S properly takes W into account in saying what is needed to obtain R and P is an implementation of S for M then we say P IMPLEMENTS R.

- As noted above, only s_v and e_v may appear as free variables in S.

Proof Obligations for Specification

Two Proof Obligations of Environment (buyer of software):

Adequacy wrt S:

$$\forall\ e\ s.\mathsf{W} \wedge \mathsf{S} \Rightarrow \mathsf{R}$$

Relative consistency for S wrt W:

$$\begin{aligned}\forall e_v.\ (\exists e_h s.\mathsf{W}) &\ \Rightarrow \\ (\exists s.\mathsf{S}) \wedge (\forall s.\mathsf{S} &\ \Rightarrow\ \exists e_h.W)\end{aligned}$$

System Proof Obligation (producer of software):
Relative consistency for M $\wedge$ P wrt S.

$$\begin{aligned}\forall e.(\exists s.\mathsf{S}) &\ \Rightarrow\ (\exists s.\mathsf{M} \wedge \mathsf{P}) \wedge \\ (\forall s.(\mathsf{M} \wedge \mathsf{P}) &\ \Rightarrow\ \mathsf{S})\end{aligned}$$

Relating WRSM with the Four Variable Model

4 Var	*WRSM*
NAT	W
Req	R + S
SOF	P
In + Out	M

4 Var	*WRSM*
i, o	s_h
m	e_v
c	s_v

Comparison of the two Models

Consider the following four variable model:

$$\begin{array}{rl} \text{Nat}: & (\forall t.c(t) > 0) \wedge (\forall t.m(t) < 0) \\ \text{Req}: & \forall t.c(t+3) = -m(t) \\ \text{In}: & \forall t.i(t+1) = m(t) \\ \text{SOF}: & \forall t.o(t+1) = i(t) \\ \text{Out}: & \forall t.c(t+1) = o(t) \end{array}$$

Each predicate is internally consistent.

These predicates satisfy all the obligations of the Four Variable Model. But:

$$\neg(\exists m\ i\ o\ c.\text{Nat}(m,c) \wedge \text{In}(m,i) \wedge \text{SOF}(i,o) \wedge \text{Out}(o,c))$$

But the WRSPM obligation:

$$\forall m.(\exists c.\text{Nat}(m,c)) \Rightarrow$$
$$(\exists i\ o\ c.\text{Nat}(m,c) \wedge \text{In}(m,i) \wedge \text{SOF}(i,o) \wedge \text{Out}(o,c))$$

would have caught the problem.