Examples of how to Determine the Specifications of Control Systems

Joey W Coleman and Cliff B Jones

School of Computing Science
University of Newcastle upon Tyne
NE1 7RU, UK
email: {j.w.coleman,cliff.jones}@ncl.ac.uk

Abstract. Creating the specification of a system by focusing primarily on the detailed properties of the digital controller can lead to complex descriptions that have no coherence. An argument put forward in a recent paper by Hayes, Jackson, and Jones gives reasons to focus first on the wider environment in which the system will reside. This paper informally explores two examples so as to illustrate this approach to system specification.

1 Overview of approach

The general idea of the "Hayes/Jackson/Jones" approach [HJJ03] is simple: for many technical systems it is easier to derive their specification from one of a wider system in which physical phenomena are measurable. Even though the computer cannot affect the physical world directly, it is still worthwhile to start with the wider system. The message can be stated negatively: don't jump into specifying the digital system in isolation. If one starts by recording the requirements of the wider (physical) system, the specification of the technical components can then be *derived* from that of the overall system; assumptions about the physical components are recorded as rely-conditions for the technical components.

In order to be able to write the necessary specifications, some technical work derived from earlier publications of Hayes, Jackson and Jones has to be brought together. The process of deriving the specification of the software system involves recording assumptions about the non-software components. These assumptions are recorded as rely conditions because we know how to reason about them from earlier work on concurrency (e.g. [Jon81,Jon83,Jon96]). In most cases, we need to reason about the continuous behaviour of physical variables like altitude: earlier work by Hayes (and his PhD student Mahony) provides suitable notation [MH91]. The emphasis on "problem frames" comes from Jackson's publications [Jac00].

A trivial example of the HJJ approach is a computer-controlled temperature system: one should not start by specifying the digital controller; an initial specification in terms of the actual temperature should be written; in order to derive the specification of the control system, one needs to record assumptions (rely-conditions) about the accuracy of sensors; there will also be assumptions about

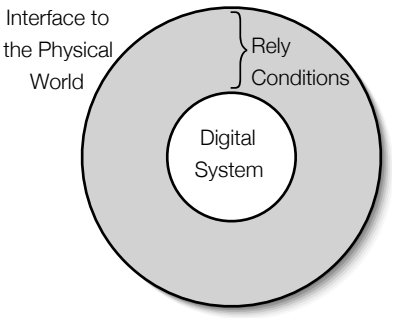

Fig. 1. Bridging from the physical world to a digital control system

the fact that setting digital switches results in a change in temperature. Once the specification of the control system has been determined, its design and code can be created as a separate exercise. At all stages — but particularly before deployment — someone has to make the decision that the rely conditions are in accordance with the available equipment. Figure 1 gives an abstract view of the HJJ approach. The referenced [HJJ03] outlines this procedure for a "sluice gate" controller. The analysis includes looking at tolerating faults by describing weaker guarantees in the presence of weaker rely conditions.

Notice that it is not necessary to build a complete model of the physical components like motors, sensors and relays: only to record assumptions. But even in the simple sluice gate example of [HJJ03], it becomes clear that choosing the perimeter of the system is a crucial question: one can consider the physical phenomena to be controlled as the height of the gate, or the amount of water flowing; or the humidity of the soil; or even the farm profits. Each such scope results in different sorts of rely-conditions.

2 Pushing out the boundaries of the system

2.1 The gas-burner

The need to start the specification phase without considering the digital system can be illustrated by examining the gas-burner example used in [HRR91]. The (interesting) physical components of the gas-burner system are:

- a processor to run the control software
- a heat request interface
- a flame sensor
- a gas valve
- an ignition transformer

The requirements, taken verbatim from [HRR91], are:

1. In order to ensure safety the gas concentration in the environment must at all time be kept below a certain threshold

2. The gas-burner should burn when heat request is on, provided the gas ignites and burns without faults
3. The gas-burner should not burn when heat request is off

And three assumptions, also verbatim from [HRR91], are given:

1. When no gas is released, the flame is extinguished after at most 0.1 seconds
2. Gas cannot ignite unless the ignition transformer is operating
3. The gas concentration will stay below the critical threshold if gas never leaks for more than 4 seconds in any period of at most 30 seconds

These requirements and assumptions, on their own, give a very sparse description of what the system is supposed to be doing. Moreover, the description hides a number of assumptions which could, on the one hand, make deployment dangerous and, on the other hand, make the specification arbitrary. The referenced paper gives the first step in formalising requirements as constructing a formal model, and defines five state variables in the digital system. They are $Heatreq$, $Flame$, Gas, $Ignition$, and $Conc$. The first four are boolean-valued, and the final one is a real-valued percentage.

Nothing in that specification constrains the use of those variables, and their relationship to the physical system is left undefined. These relationships are critical: should those variables be used as sensors, so that their value is relied upon to reflect the physical world, or are they used as a channel to send commands to the physical components of the system?

The $Heatreq$ and $Flame$ variables appear to be inputs — $Heatreq$ is the input that tells the gas-burner to turn on, and $Flame$ appears to be tied to the flame sensor component in the physical system. The $Conc$ variable, used to denote the relative gas concentration around the burner, is most likely a "ghost" variable, as the physical system has no sensor to measure gas concentration. The Gas and $Ignition$ variables must then be outputs from the system, used to control the gas valve and ignition transformer respectively.

2.2 Extending the system boundaries

What is the actual purpose of the gas-burner? The specification as developed gives the impression that the purpose is to burn gas — when the $Heatreq$ signal is on — given certain time-related constraints.

Moving the boundary outwards from that, one could say that a more accurate description of the purpose of the gas-burner is to burn gas safely. The adjective "safely" is used informally here and simply means that no explosions occur and nobody is asphyxiated or intoxicated from high concentrations of gas in the environment.

Pushing the boundary of the system out further, the purpose of the gas-burner is probably to generate heat. Perhaps this is obvious; after all, one of the signals in the referenced model is called $Heatreq$. However, that merely prompts us to ask about the precise relationship between the $Heatreq$ signal and the operation of the gas-burner. Even at this level we do not know what it is that we are trying to heat, that is, what the use of the gas-burner is.

2.3 Back to the example

One of the first things to do is look at the real requirements of the system. If we take the purpose of the system as simply to generate heat, we can quickly come up with some general requirements.

The machine's behaviour, during "normal" operation, would have requirements like:

- If $Heatreq$ signal comes on at some point in time means that the gas-burner will start to generate heat soon after.
- When the gas-burner is generating heat the $Heatreq$ signal must be on and must have come on in the relatively recent past.
- When the $Heatreq$ signal turns off then the gas-burner will stop generating heat soon after.

These requirements would be based on assumptions like:

- A flame in the gas burner generates heat.
- The presence of gas and a spark will cause a flame.
- Gas is present if the gas valve is turned on.
- The ignition transformer generates sparks.
- The gas-burner can sense the state of the $Heatreq$ signal in a timely manner.

The assumptions tend to be very simple, but each can be easily formalized if necessary. Note that the sample requirements here are not intended to cover unusual situation — they are intended for a perfect environment.

The requirements for the machine when faced with an imperfect environment could include:

- The machine does not cause explosions.
- The machine does not cause toxic concentrations of gas in the environment.

This requirement forces us to consider assumptions like:

- A large concentration of gas can cause an explosion.
- Small concentrations of gas can not cause an explosion.
- The environment cannot change in such a way so that the maximum safe concentration of gas is less than some specific amount.
- The concentration of gas in the environment increases when the gas is on without a flame.
- The concentration of gas in the environment cannot increase when the gas is off.
- The environment causes concentrations of gas to dissipate over time.
- The machine will only have to deal with a single type of gas.
- The characteristics of the gas — volatility, ignition temperature, etc. — are constant during operation.
- The ignition transformer is the only source of sparks.
- There is no other source of gas in the environment other than the gas-burner.

– The environment does not actively inhibit gas-burning, but it is possible for the environment to extinguish the flame even while the gas is on.
– The gas valve cannot fail to close.
– It is also assumed that the rate of flow of gas is constant, or has a constant maximum. This is dependent on nozzle size, gas pressure and so on.

All of these assumptions are important, though this is not intended to be an exhaustive list. While many may seem trivial, violating any of them can cause a situation where the machine cannot meet its guarantee-conditions, and thus — potentially fatally — fail to meet the requirements.

From all of the requirements and assumptions above we can consider the behaviour of our machine. The observable behaviour is given through the use of guarantee-conditions, i.e.:

– The ignition transformer generates a spark after gas is turned on.
– The time between turning the gas on and the ignition transformer generating a spark is much less than the amount of time it would take for the concentration of gas in the environment to exceed a certain threshold.
– If the gas fails to ignite then the gas will be turned off, and will not be turned back on for a period of time.

Among others, there would also be guarantee-conditions that covered the specific relationship between the $Heatreq$ signal and the actions of the gas-burner.

To put the structure of the overall system into perspective it is useful to create a problem diagram of the sort described in Jackson's book [Jac00]. The diagram then acts as an aid when identifying the assumptions and possible sources of interference about which the specification needs to be concerned. Figure 2 gives a possible problem diagram from the gas-burner.

The "Control Machine" domain is the digital system whose specification we want to determine and the "Gas-Burner" domain is the physical gas-burner. The "Environment" domain represents the environment in which the gas-burner is placed. The oval labelled "Requirements" shows the relationship between the three domains it connects and shows that the behaviour of the gas-burner is what is being constrained.

The last domain, "Control Signals", was left aside as its presence while working on the diagram highlighted an important omission from the original description in [HRR91]: precisely what is controlling the $Heatreq$ signal? Even more than just that single example, the diagram also makes the possibility of change in the environment more explicit and shows — by omission — that it is strictly not possible for the machine to inspect the concentration of gas in the environment. The combination of rely-conditions and problem diagrams provide a very good means of identifying the properties — assumed or otherwise — of the overall system.

The problem diagram has the useful effect of giving a visual representation of the possible sources of interference that need to be recorded by rely-conditions. Every variable shared between domains in the diagram will, at the very least,

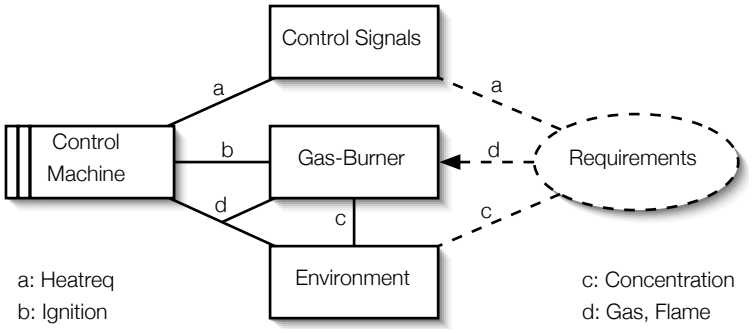

Fig. 2. Problem diagram for the gas-burner

need a rely-condition that describes the behaviour we assume it will have. Furthermore, we will also need a rely-condition for every situation where two (or more) variables have some relationship in their values.

Assumptions like the characteristics of the gas and nozzle — volatility, rate of flow, and so on — can be coded as rely-conditions fairly directly. This can even allow for some of the rely-conditions to be derived more-or-less automatically, rather than written down without any context.

The rely-conditions and the properties of the overall system are used to justify the set of guarantee-conditions that fulfill the requirements. The combination of rely- and guarantee-conditions, matched against the requirements, form the basis on which the user makes the decision as to whether or not the machine's behaviour is suitable.

Despite the linear presentation here, the construction of requirements, rely- and guarantee-conditions, problem diagrams, and the identification of assumptions is not done in a linear fashion. All of these specific tools should be used to influence the others.

3 Avoiding confusion between assumptions and requirements

The message of the general method (Section 1) as exemplified by the previous section applies to all examples: clarify the requirement in the real world before trying to specify the software which sits within the system. This process naturally identifies assumptions about the physical components which can be recorded (as in [HJJ03]) as rely-conditions.

As an indication that there is another danger of focussing too early on the computer system, we identify some reservations about one of the many specifications of the "Production Cell" example. This interesting problem is explored using many different approaches in [LL95]. The specification which we investigate is [MC94] (which is the journal version the paper by MacDonald and Carrington in [LL95]).

For the purposes of this workshop version of the paper, we assume that the reader is familiar with the overall problem.[1]

3.1 Normal operation

- Section 2 of [MC94] contains an argument for the assumption that the Feed Belt can contain only one metal block at a time (and a discussion of how changing this assumption would change the model). This is not presented as an assumption in the description; it becomes hidden in the state abstraction for *Component_Loaded*.
- There are several places (e.g. Sections 3, 4.1, 4.2, 5.1 of [MC94]) where assumptions are made on the initial state of the system.
- A specific concern about Z is that it does not specifically identify pre-conditions of operations; this raises the question whether this decision contributes to the confusions (e.g. Section 3.2 of [MC94])
- It can be concluded from the specifications of *Extend* and *Retract* (in Section 4.1 of [MC94]) that these operations are not allowed to change *load_pos* or *unload_pos* but it is unclear whether this is an assumption on the equipment or a requirement on the code.
- Similarly, the specifications of *Load* and *Unload* (in Section 4.1 of [MC94]) indicate in their predicates that these operations are only allowed in certain positions; in this case (unlike the previous one) it might well be a requirement on the code.
- Section 4.3 of [MC94] has requirements about not rotating the robot if either arm is extended but it is left to guesswork as to whether this is an assumption on the equipment or a requirement on the code.
- Section 4.2 of [MC94] makes statements about "the press must be empty" without clarifying whose responsibility it is to achieve this situation.
- Similarly for unloading requiring that there is something to unload.
- Section 7 of [MC94] states that "the pre-condition[2] ensures there is no collision between the loaded robot and the elevating rotary table"!
- usw. usw.

4 Further work

The most obvious immediate objective is to completely formalise the examples discussed in this paper in Hayes-Mahoney logic [MH91]. Tackling these and similar further examples will inevitably refine the method described in [HJJ03]. Less immediately, further work includes creating a library of examples — including the two given here — to create a body of work that can serve as a guide to practitioners. These examples would need to be fully formal, and worked out up to the point where an implementation would be designed.

[1] Very briefly, the system has a press unit to which items are transferred from a belt by a lifting device.

[2] of $Move\_ERT\_to\_Loading\_Position\_1$

In the longer term, it should be possible to use such a library of examples to generate a set of "HJJ patterns", not unlike the design patterns [GHJV95] currently used by practitioners of object-oriented development. Even if a set of pattern-like structures cannot be developed, a full set of guidelines for using this method is required.

The composition of specifications given with this method, in senses of both subproblems and whole specifications, is a problem that remains to be fully explored. The task of creating a specification for a machine's "normal" operation seems well understood, and creating the specification with weaker rely-conditions for the "abnormal" machine behaviour is equally straightforward. However, the problem of combining such specifications is a problem that demands further study.

The basic ideas involved in the Jones' rely-conditions, while good at recording interference, leave gaps when it comes to notions such as ensuring that the system can make progress. Work such as Stølen's on wait-conditions [Stø91] addresses some of these issues, and should be included in this method.

The notation given in Jackson's [Jac00] for problem diagrams needs extension to be able to directly record interference notation. The current notation does not allow for more than a single domain to control a variable. Figure 2 is less detailed than it might have been because of this.

Acknowledgments
The authors are both supported in their research by EPSRC (UK) funding for the "Dependability IRC" (DIRC — see www.dirc.org.uk) and by EU-IST STREP funding for "RODIN" (see rodin.cs.ncl.ac.uk). The many discussions about this topic with Michael A. Jackson have been a wonderful source of insight into this material.

References

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[HJJ03] I. J. Hayes, M. A. Jackson, and C. B. Jones. Determining the specification of a control system from that of its environment. In Keijiro Araki, Stefani Gnesi, and Dino Mandrioli, editors, *FME 2003: Formal Methods*, volume 2805 of *Lecture Notes in Computer Science*, pages 154–169. Springer Verlag, 2003.

[HRR91] K. M. Hansen, A. P. Ravn, and H. Rischel. Specifying and verifying requirements of real-time systems. In *SIGSOFT '91: Proceedings of the conference on Software for Critical Systems*, pages 44–54, New York, NY, USA, 1991. ACM Press.

[Jac00] M. A. Jackson. *Problem Frames: Analyzing and structuring software development problems*. Addison-Wesley, 2000.

[Jon81] C. B. Jones. *Development Methods for Computer Programs including a Notion of Interference*. PhD thesis, Oxford University, June 1981. Printed as: Programming Research Group, Technical Monograph 25.

[Jon83] C. B. Jones. Specification and design of (parallel) programs. In *Proceedings of IFIP'83*, pages 321–332. North-Holland, 1983.

[Jon96] C. B. Jones. Accommodating interference in the formal design of concurrent object-based programs. *Formal Methods in System Design*, 8(2):105–122, March 1996.

[LL95] C. Lewerentz and T. Lindner, editors. *Formal Development of Reactive Systems - Case Study Production Cell*, volume 891 of *Lecture Notes in Computer Science*. Springer, 1995.

[MC94] A. MacDonald and D. Carrington. Z specification of the production cell. Technical Report 94-46, University of Queensland, 1994.

[MH91] B. Mahony and I. J. Hayes. Using continuous real functions to model timed histories. In P. Bailes, editor, *Engineering Safe Software*, pages 257–270. Australian Computer Society, 1991.

[Stø91] K. Stølen. An attempt to reason about shared-state concurrency in the style of VDM. In *VDM '91: Proceedings of the 4th International Symposium of VDM Europe on Formal Software Development-Volume I*, pages 324–342, London, UK, 1991. Springer-Verlag.