The meaning of requirements

Michael Jackson

101 Hamilton Terrace, London NW8 9QX, England
E-mail: jacksonma@attmail.com, mj@ic.ac.uk

We use the term *requirements* to denote what are often called *functional* requirements. Requirements are located in the *environment*, which is distinguished from the *machine* to be built. A requirement is a condition over *phenomena* of the environment. A *specification* is a restricted form of requirement, providing enough information for the implementer to build the machine (by programming it) without further environment knowledge. To describe requirements appropriately we must fit our descriptions into an appropriate structure. This structure must respect the distinction between the machine and the environment, and the distinction between those environment properties that are given (*indicative descriptions*) and those that must be achieved by the machine (*optative descriptions*). Formalisation is a fundamental problem of requirements engineering. Since most environments are parts of the physical world, and therefore informal, the formalisation task is inescapable. Some techniques are discussed for tackling this task. In particular, the use of *designations* is explained, and the distinction between *definition* and *assertion*. By using the smallest possible set of designated terms, augmented by appropriate definitions, the developer can create a *narrow bridge* between the environment and its descriptions in the requirements. In this way a sufficiently faithful approximation to the informal reality can be obtained.

1. Introduction

1.1. The importance of requirements

Reliance on computer-based systems, especially in safety-critical applications, involves serious risks and dangers. Some failures result from relatively straightforward programming errors, as in the case of the Therac-25 radiotherapy machine [Leveson and Turner 1993]. Others result from a mismatch between the designed behaviour of the computer part of the system and the effects in the environment that this designed behaviour is intended to achieve. The computer hardware may perform correctly, and the software may satisfy its specification; but the results are not what was intended, and may be disastrous.

Many examples of this second kind of failure are reported in Neumann [1995]. Such failures are properly attributed to errors in the engineering of the system *requirements*: the true requirements of the system were not correctly identified; or were obscurely or imprecisely expressed; or were based on faulty reasoning about the environment or on faulty approximations to the reality of the phenomena and properties of the environment.

The theme of this paper is that significant improvements in requirements engineering can be obtained by careful attention to the meaning of requirement statements. The paper explains a distinction between requirements and specifications, and discusses techniques of formalising and describing requirements in a way that allows them to be more clearly expressed and more easily understood and validated. The ideas underlying the techniques can be applied both to the engineering of new requirements documents and to the analysis and understanding of existing documents. The central goal of the paper is to offer useful insight into the problems of requirements description and understanding and how they can be successfully addressed.

1.2. A concept of requirements

The term *requirements* has many meanings. Arguably, all of the following are requirements:

- The computer must not weigh more than 0.25 Kg.
- The system must be completed by 1st January 1998.
- The programs must be written in Ada.
- The system specification must be formally accepted by the steering committee.
- The operator interface must be easy to learn.
- The system must produce a monthly report of outstanding debts.
- If passenger in the lift presses the *open-doors* button while the lift is stationary at a floor, the doors should begin to open within 0.5 secs.

This paper is about requirements in a narrow sense, in which we would include at most the last three of the examples above, but more probably only the last two. In effect, we are concerned with what are often called *functional* requirements. However, we do also include under this heading such requirements as real-time response [Jackson and Zave 1995] and those properties of operational safety that can be precisely stated in terms of system behaviour. Requirements of these kinds are functional; but they are often excluded from the corpus of functional requirements for no better reason than the technical difficulty of treating them in a sufficiently formal way.

1.3. The machine and the environment

In this paper we are concerned with the description of requirements for systems whose construction is primarily a software development task. That means that our eventual goal is the provision and installation of a *machine* – all or part of one or more computers programmed to behave in a way that ensures satisfaction of the requirements. We do not construct the computer hardware; but in effect we construct the machine because we construct the software that transforms a general-purpose computer into the machine we need.

The requirements, however, do not directly concern the machine. They concern the environment into which it will be installed. The environment is the part of the world with which the machine will interact, in which the effects of the machine will be observed and evaluated. For a lift-control system, the environment includes the floors served, the lift shaft, the motor and winding gear, the doors, the lift car, the buttons and indicator lights, and the passengers. For a theatre booking system it includes the theatres and their seats, the audiences, credit cards used to buy tickets, the tickets themselves, and the performances together with their postponements and cancellations. For an avionics system it includes the pilot, the airframe and engines, the control surfaces, the surrounding atmosphere, the landing gear and the airport runways.

The distinction between the environment and the machine is partly a distinction between what is given and what is to be constructed. The terms are therefore to be understood in a relative sense. For the developers of an operating system kernel, for example, the kernel is to be constructed; with the hardware resources it uses it will constitute the machine. What is given – and is therefore the environment – is the rest of the same computer, and the population of programs running in it whose executions will be supported and controlled by the kernel.

1.4. Shared phenomena

The machine can affect, and be affected by, the environment only because they have some *shared phenomena* in common. That is, there are some events that are events both in the machine and in the environment; and there are states that are states of both.

In the lift system, for example, the machine is directly connected to the motor switch and to the sensors that detect the presence of the lift car at the floors. A *turn-motor-on* event is an event both in the motor switch and in the machine. So too is a *set-motor-polarity-upwards* event. The event will, very likely, be differently named in the machine's programming language and in the switch manufacturer's equipment manual; but it is the same event. Similarly, the state *up-sensor-2-on* is a state shared by a bit in the machine's store and a sensor located in the lift shaft at floor 2. On the other hand, the departure of a disgruntled passenger from a floor lobby after waiting fruitlessly for the lift to arrive is a *private* event of the environment that is not shared with the machine; and the position of the read-write heads of the disk drive is a private state of the machine that is not shared with the environment.

In considering shared phenomena, it is essential to distinguish between those that are controlled by the machine and those that are controlled by the environment. The *turn-motor-on* event and the *set-motor-polarity-upwards* event are controlled by the machine. The value of the state *up-sensor-2-on*, by contrast, is controlled by the environment.

2. Requirements and environment properties

2.1. Requirements are in the environment

Requirements, in the sense in which we are using the word, are located in the environment. That is to say, they are conditions over the events and states of the environment. The customer for the lift control system requires that when a button is pressed the lift should come to the floor as requested. The customer for the theatre reservations system requires that seats should not be double-booked, and that the best seats at each price should be allocated first. Requirements, in this sense, can be stated entirely without reference to the machine. It may be that some of the events and states that are the subject of requirements are shared with the machine, but this is purely accidental: what is essential is that they are phenomena of the environment.

The question immediately arises: how can the machine ensure satisfaction of a requirement that concerns private phenomena of the environment, in which it does not participate? The answer lies in the given properties of the environment. These given properties constitute a nexus of constraints and causal chains. They guarantee that by directly affecting shared phenomena the machine can indirectly affect private phenomena of the environment; and that a machine possessing direct sensitivity to shared phenomena will have an indirect sensitivity to certain private phenomena. This general relationship between the environment and machine phenomena is pictured in Fig. 1.

In the figure, the requirements are concerned only with private phenomena. This is not necessary: requirements may be concerned also with shared phenomena. But it is typical, because the customer for a system is usually interested in parts of the environment that are some way from the machine in the causal chain.

Consider the example of the lift control system. The immediate phenomena of movement of the lift car – its position and velocity – are not shared with the machine. Yet it is a requirement that in certain circumstances the machine should cause the lift car to move upwards to a certain floor and stop there. The environment has these given properties (among others):

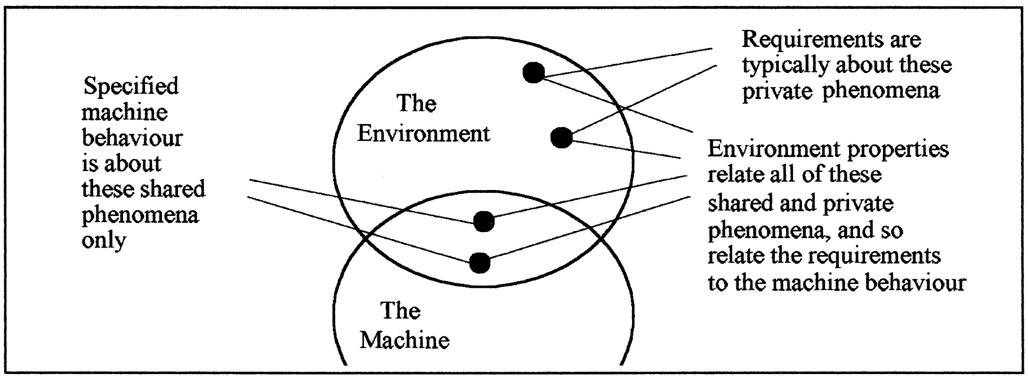
Specified machine behaviour is about these shared phenomena only
The Environment
The Machine
Requirements are typically about these private phenomena
Environment properties relate all of these shared and private phenomena, and so relate the requirements to the machine behaviour

Figure 1.

- If the lift motor is switched on while its polarity is set in the upwards sense, the car will start to rise within 0.2 seconds.
- The lift car is constrained to move vertically in the shaft from one floor to the next.
- When the lift car is at any point between -8.25 inches and $+0.25$ inch of a floor position, the up-sensor at that floor is on.
- If the lift motor is switched off while it is raising the car, the car will halt after rising a further 8 inches.

By virtue of these properties, the machine can cause the car to rise to a floor by controlling and sensing shared phenomena in the obvious way. It sets the motor polarity upwards and switches the motor on (both of these events are under machine control). It waits until the up-sensor at the floor flips on (this state change is under the control of the environment), and then switches the motor off.

2.2. Optative and indicative

The full description of a requirement therefore consists of at least two parts. We must describe the requirement itself – the *desired* condition over the phenomena of the environment. And we must also describe the *given* properties of the environment by virtue of which it will be possible for a machine, participating only in the shared phenomena, to ensure that the requirement is satisfied.

This distinction between the desired and the given must be reflected in a separation of descriptions:

- A customer requirement $\mathcal{R}$ expresses a condition over the phenomena of the environment that we wish to make true by installing the machine.
- An environment assertion $\mathcal{E}$ expresses a condition over the phenomena of the environment that we know to be true irrespective of the properties and behaviour of the machine.

A traditional grammarian would say that the requirement $\mathcal{R}$ is in the *optative* mood, expressing a wish; the environment assertion $\mathcal{E}$ is in the *indicative* mood, expressing what is claimed to be a known truth.

2.3. The indicative context

Almost always, introducing a new system will change the environment in significant ways. The environment assertion $\mathcal{E}$ describes the given properties of the environment as they will be when the machine has been installed and the system is in operation.

The distinction between indicative and optative is not therefore a distinction between environment properties holding at one time and environment properties holding at a later time. It is, rather, the distinction between two classes of environment properties holding at the same time: those that are guaranteed by the environment itself, and those that are to be guaranteed by the machine.

In some problems, system operation may bring about environment changes that are very hard to predict. This is notoriously true of traffic patterns and of human habits and skills, but less subjective examples can also be found. The indicative properties of interest to the requirements engineer are then the changed, not the original, properties. A failure of prediction results in environment assertions that prove eventually to be false.

The distinction between indicative and optative is applied here to descriptions of the environment. But it applies also to descriptions of some *agents* [Feather 1987; Dardenne *et al.* 1993]. An agent is an object which is a processor for some actions; examples of agents are human beings, physical devices, or programs that exist or are to be developed [Dardenne *et al.* 1993]. In all of these examples it is likely, if not certain, that an individual agent will have indicative, given, properties, and also that it will be required to exhibit some desired, optative, properties. The need to distinguish the two is clear [Van Lamsweerde *et al.* 1995].

2.4. Why environment assertions are necessary

Although the optative description $\mathcal{R}$ and the indicative description $\mathcal{E}$ are both relevant to requirements, it might seem at first sight that the work of the requirements engineer as such is complete when the customer requirement $\mathcal{R}$ has been written. From a narrow point of view this may indeed be true; but there are several reasons why the formulation of the accompanying environment assertion $\mathcal{E}$ is also an integral part of the requirements engineering task.

First, there is a simple human reason. Making an indicative description of the environment and validating it by careful study and by discussion with domain experts is a vital way of obtaining and demonstrating the necessary understanding of the environment in which the customer requirements are located and in which they make sense. This is a powerful and important reason, but we will not pursue it further here.

Second, it is necessary to show that the requirement is satisfiable by some machine. It may be that the environment does not embody enough constraints and enough causal chains to connect the shared phenomena appropriately to the phenomena that are of direct interest to the customer. In that case, the problem is not properly a software development problem: it is a problem of the kind that we may call *environment engineering*. For example, suppose that the customer for the lift system has the following unusual additional requirement:

- If a passenger in the lift requests travel to a floor, but then leaves the lift before that floor is reached, the request should be cancelled.

This requirement is easily formalised, but its satisfaction can not be ensured by any machine in an environment of conventional lift equipment. The shared phenomena are insufficient to allow the machine, however indirectly, to identify the individual passenger who participates in a departure event or in a request event. Satisfaction of this requirement, therefore, will demand a substantial change or enhancement of the environment. We regard this as taking us outside our chosen realm of requirements engineering for software development.

2.5. *Requirements and specifications*

To show that the requirements are satisfiable by some machine we derive a specification of the machine. A specification $\mathcal{S}$ is an optative description of a condition over the shared phenomena at the interface between the machine and the environment. A machine satisfying $\mathcal{S}$ will ensure satisfaction of the requirement. That is,

$$\mathcal{E}, \mathcal{S} \vdash \mathcal{R}.$$

If a machine whose behaviour satisfies $\mathcal{S}$ is installed in the environment, and the environment has the properties described in $\mathcal{E}$, then the environment will exhibit the properties described in $\mathcal{R}$.

The relationship among $\mathcal{E}$, $\mathcal{S}$ and $\mathcal{R}$ is an entailment, not an implication. The implication

$$\mathcal{E} \wedge \mathcal{S} \Rightarrow \mathcal{R}$$

(unless it were a tautology) would itself be a further assertion about the environment, in addition to the assertion $\mathcal{E}$. But the essence of the relationship is precisely that $\mathcal{R}$ can be deduced from $\mathcal{E}$ and $\mathcal{S}$ with no further knowledge of the environment.

2.6. *The nature of a specification*

A specification forms a bridge between requirements engineering, which is concerned with the environment, and software engineering, which is concerned with the machine. The distinction is of practical importance, because it clarifies the differing responsibilities of those whose expertise lies in acquiring and using knowledge of the environment – often called application or domain knowledge – and those whose expertise lies in the invention, design, and construction of computer software. In principle, a specification allows requirements engineers to reason about the requirement and its satisfaction in the environment, without mentioning the properties of the machine. It also allows programmers to reason about the software and its adequacy for its purpose without mentioning either the environment properties or the customer's requirement. This is why it has traditionally represented the intermediate product between requirements and programs.

To serve its purpose, a specification must be subject to a further constraint beyond its restriction to shared phenomena. Its satisfiability must be demonstrable without appeal to properties of the environment. That means that by appeal only to logic and to the properties of the general-purpose computer a specification must be formally refinable to a conjunction of liveness and safety descriptions respectively of the form:

"whenever c is true cause e to occur within time t"

and:

"whenever c is true do not permit e to occur"

in both of which the condition c can be evaluated entirely in terms of the past history of shared phenomena, and e is a shared event or state transition that is controlled by the machine. Whether the real-time constraint t can be satisfied will depend on the properties – in particular, the speed – of the computer.

3. Description and the environment

3.1. Formalisation and informal environments

For the programmer, the computer can be treated as a formal system. Although the underlying physical reality is inevitably informal, the computer has been carefully constructed so that for practical purposes we may rely on the formal description of its behaviour given in the programming manual. When we read: "Execution of the instruction 21,1,5 causes register R1 to be set to the value held in register R5", there is no room for doubt about the meanings of the terms used or about the validity of the statement. The task of establishing a reliable phenomenology for the computer has already been performed.

The environment, by contrast, is usually a part of the physical world that has not been formalised. We normally speak of it in natural language, with all its attendant ambiguities and uncertainties. Terms such as 'sale' or 'payment' have different meanings for different speakers, and even for one speaker at different times. Even where there is no ambiguity about the intended meaning of a term, there will be many cases in which it is hard to decide whether or not the term should be applied. Furthermore, there is an unbounded collection of considerations that may be relevant to any proposed statement; as a result, any general statement about the environment may be subject to an unlimited number of exceptions and special cases.

There is, of course, one part of the environment that has been formalised: the phenomena shared with the machine. We may be unsure whether one car can be said to have hit another in a traffic incident, but there is no uncertainty whether a particular key was hit on a computer keyboard. The keyboard circuitry and the associated software are engineered precisely to avoid doubt in the question by providing an objective criterion. This is why it is reasonable to demand formality in specifications, and to see no fundamental difficulty in meeting that demand.

In requirements, by contrast, the informal nature of the environment does present a fundamental difficulty. The informality must be tamed if we are to describe the requirements intelligibly and to reason reliably about their satisfaction by the interaction of the machine and the environment.

3.2. Ground terms

The first need is to establish an adequate set of ground terms for our descriptions. Ground terms for a description are the terms that fix the relationship between the description and what it describes. For example, if we wish to describe human biological relationships we may use many terms such as *mother*, *father*, *uncle*, *brother*, *aunt*, *niece*, *grand-daughter*, *second cousin*, and so on. But a sufficient set of ground terms is $\{$*male*, *female*, *parent*$\}$. All the other terms can be defined on the basis of these three, and all our descriptions can then be understood if these three ground terms are understood.

A requirements description, whether indicative or optative, expresses a relationship over environment phenomena. It will not be understandable unless it is made unambiguously clear what phenomena are denoted by each term of the description, and how occurrences of those phenomena are to be distinguished from non-occurrences. The uncertainties of natural language are not dispelled merely by resolving apparent conflicts between competing descriptions or viewpoints [Easterbrook 1993; Easterbrook and Nuseibeh 1995]. Even when all conflicts have been resolved, it is still necessary to provide an unambiguous mapping between formal terms and informal phenomena.

The fundamental technique in providing this unambiguous mapping is to choose as ground terms only those phenomena that admit of sufficiently reliable and unambiguous recognition. It is a serious mistake to assume that because a noun or verb or adjective is conveniently used in informal natural language discourse it must necessarily denote some phenomenon or class of phenomena that can be treated as a ground term in discourse about the environment.

Consider, for example, the development of a system for managing airline services. It may seem natural to assume that *flight* may be treated as a ground term, because it is a term commonly used in informal discourse about airline services. But the treatment of flights as recognisable and distinct individuals is fraught with difficulty. Is a flight that is cancelled a flight? Can one flight be split into two legs flown by different planes? Can two flights be combined in one? Can a heavily used shuttle flight be simultaneously flown by two different planes? The difficulty of answering these questions indicates that we should not attempt in this context to treat flights as ground terms.

3.3. Designations

Each choice of a ground term must be explicitly made and explicitly captured. The appropriate tool for this purpose is a *designation* [Jackson and Zave 1993; Jackson 1995]. A designation associates a formal ground term, such as a predicate, with the denoted phenomena, such as an event or entity class or a relationship over events or entities. For example, we might write the designation

$$\text{Mother}(x, y) \cong x \text{ is the genetic mother of } y.$$

The left-hand side is the formal term; in this case the predicate Mother(x, y). The right-hand side is a – necessarily – informal recognition rule by which the designated phenomena may be unambiguously recognised. Mother(x, y) is true if and only if x is the genetic mother of y.

Because the natural world is informal, the recognition rule in a designation is inevitably imperfect. However carefully it may be formulated, there may always be hard cases in which we are uncertain whether or not the rule applies. But we must limit this uncertainty to an acceptable level.

First, we must write the recognition rules with great care. If we had written

$$\text{Mother}(x, y) \cong x \text{ is the mother of } y,$$

we would have left the reader uncertain whether we meant to include stepmothers and adoptive mothers. If we had written

$$\text{Mother}(x, y) \cong x \text{ is the natural mother of } y,$$

we would have failed to clarify our intention in cases of surrogate motherhood. Of course, even writing

$$\text{Mother}(x, y) \cong x \text{ is the genetic mother of } y$$

may be insufficient in some nightmare future in which genetic engineering makes it possible to combine in one child the genetic inheritance of more than one mother.

Second, we must recognise that while the recognition rule can not be perfect it must be good enough for the world as it is and as it will be during the operational lifetime of the system. Appropriate choice of designated phenomena, therefore, depends heavily on the environment and system for which they are chosen. The designation

$$\text{Bird}(b) \cong b \text{ is a bird}$$

may be appropriate to a system whose environment is a children’s zoo. But it would not do at all for a system concerned with the study of evolution.

3.4. The narrow bridge

Appropriately chosen and carefully written designations provide a strong and narrow bridge between the environment and its description in requirements. They define the scope of a requirement, in the sense of bounding the parts and aspects of the environment with which the requirement is concerned. They clarify the meaning of the descriptions that use them, and allow those descriptions to be subjected to the test of falsifiability. For the requirements engineer who has written explicit designations there can be no refuge in the rejoinder “Well, it all depends on what you mean by X”.

Designations also allow us to reason more reliably about the environment — as we must if we are to convince ourselves that satisfaction of our specification will guarantee satisfaction of the requirements. However good our designations, the informality of the world introduces an inevitable error factor into the mapping between the informal reality and its formal description. The accuracy of our claim that the system will satisfy its requirements can not be better than this error factor. By confining ourselves to ground terms with the most reliable possible recognition rules, we minimise the error factor, and consequently minimise also the error in the results of our formal reasoning based on that mapping.

3.5. The use of definition

Limiting our designations in this way may at first sight appear to be inconveniently restrictive. The convenience of natural language locutions is not accidental, and we need to be able to extend the terminology of our descriptions beyond the narrow confines of reliable designations. The appropriate tool here is *formal definition*. We define new terms on the basis of terms previously designated or previously formally defined.

These formal definitions add nothing to the bridge between the reality and its description; nor do they constitute fresh assertions about the reality. They merely provide more convenient terminology for saying what we could have said less conveniently without them. They may be thought of as abbreviations [Woodcock and Davies 1996]: descriptions using the formally defined terms can always be rewritten to use only the designated ground terms on which they are ultimately based. It follows [Woodcock and Davies 1996] that these formal definitions may not be recursive.

The difference between designation and definition can be clearly seen from a simple example. Suppose that for an inventory system we have designated an event class:

WidgetMvmt$(t, e, m) \cong$ In event e a stock movement of m widgets occurs at time t.

WidgetMvmt(t, e, m) is an observable phenomenon, recognisable if and only if an event e occurs at time t and involves the receipt $(m \geqslant 0)$ or issue $(m < 0)$ of $|m|$ widgets into or out of the warehouse. In an inventory system we will surely be interested in the question: How many widgets *should we* have in stock? We may write a definition.

$$\begin{aligned}&\text{ExpectedWidgetStock}(t, s)\\ &\quad\overset{\text{def}}{=} s = \Big(\sum tm, e, m \mid \text{WidgetMvmt}(tm, e, m) \,\wedge\, tm < t \bullet m\Big).\end{aligned}$$

ExpectedWidgetStock(t, s) is *defined* to mean that s is the cumulative sum of (positive and negative) movement quantities m, the sum being taken over all possible choices of tm, e, and m for which WidgetMvmt(tm, e, m) is true and $tm < t$. That is, s is the sum of the movement quantities in all WidgetMvmt events occuring before t.

ExpectedWidgetStock(t, s) is not designated; it is not a directly observable phenomenon at all. It is simply defined in terms of WidgetMvmt events, which are the only observable phenomena mentioned so far. Any assertion about ExpectedWidgetStock is immediately translatable into an assertion about WidgetMvmt events. The definition adds nothing to our capacity to describe the environment – merely to the convenience of our descriptions.

Now suppose that we want to be able to answer the question: How many widgets *do we actually* have in stock? We need to designate a fresh phenomenon:

$$\text{WidgetStock}(t, s) \cong \text{ At time } t \text{ the number of widgets in the warehouse is } s.$$

WidgetStock(t, s) is an independently observable phenomenon, recognisable as the presence of s widgets in the warehouse at time t. The designation adds significantly to our capacity to describe the environment: using WidgetStock(t, s), we can make assertions that can not be made without it. For example, we can assert that the actual stock of widgets changes only by stock movement events – there is no theft or evaporation, and no spontaneous creation of widgets:

$$\forall t, s \bullet \text{WidgetStock}(t, s) \Leftrightarrow \text{ExpectedWidgetStock}(t, s).$$

For any choice of t and s, the actual stock at time t is s if and only if the expected stock at time t is s. This assertion is equivalent to the less convenient

$$\begin{aligned}&\forall t, s \bullet \text{WidgetStock}(t, s)\\&\quad \Leftrightarrow s = \Big(\sum tm, e, m \mid \text{WidgetMvmt}(tm, e, m) \wedge tm < t \bullet m\Big).\end{aligned}$$

3.6. *Defining individuals*

The definition of ExpectedWidgetStock(t, s) increases the convenience of our descriptions by adding a new predicate symbol. Often, we want to add terminology that seems to involve new individuals.

Suppose that in describing airline operations we have designated:

$$\begin{aligned}\text{Plane}(p) &\cong p \text{ is a plane},\\ \text{Land}(e, p, t) &\cong \text{In event } e \text{ the plane } p \text{ lands at time } t,\\ \text{TakeOff}(e, p, t) &\cong \text{In event } e \text{ the plane } p \text{ takes off at time } t.\end{aligned}$$

These designations involve individuals that are planes, individuals that are points in time, and individuals that are events. These are distinct individuals, in the sense that we can reliably and unambiguously distinguish one plane from another, one event from another, and one point in time from another. Now we wish to deal with some of the considerations that make *flight* a useful term in talking about airline operations. We rejected *flight* earlier, on the grounds that there are no clear criteria for distinguishing one flight from another, and that it is therefore impossible to designate the term. That rejection still holds. But we can use definition to build up a set of identifiers and defined predicates that will serve at least some of our purposes.

For example, we can define the notion of a *trip* as it applies to air travel. An appropriate definition is

$$\begin{aligned}&\big(\mathrm{i} \bullet \mathrm{Trip}(i) \ \wedge\ \mathrm{TripPlane}(p,i) \ \wedge\ \mathrm{StartsTrip}(e,i) \ \wedge\ \mathrm{FinishesTrip}(f,i)\big)\\ &\overset{\text{def}}{=} \big(p,e,f,t1,t2 \mid \mathrm{Plane}(p) \ \wedge\ \mathrm{TakeOff}(e,p,t1) \ \wedge\ \mathrm{Land}(f,p,t2) \ \wedge\ t1 < t2\\ &\qquad\qquad \wedge \neg \big(\exists g,t3 \bullet \big(\mathrm{Land}(g,p,t3) \ \wedge\ t1 < t3 < t2\big)\big)\big).\end{aligned}$$

The definition defines both the individual i and the four predicates – Trip, TripPlane, StartsTrip and FinishesTrip – in which i may appear as an argument. There is a defined individual i for each distinct choice of p, e, f, $t1$ and $t2$ such that Plane(p) and TakeOff$(e,p,t1)$ and Land$(f,p,t2)$ are all true, and $t1$ is earlier than $t2$, and no Land event of the same plane intervenes between e and f.

The four predicates Trip, TripPlane, StartsTrip and FinishesTrip are defined to have their obvious meanings; i is a trip; p is the plane involved in the trip; e and f are the events that start and end the trip. Intuitively, any matching pair of take-off and land events of the same plane defines a unique trip started by the take-off event and finished by the land event.

The values of the identifier i for which Trip(i) is true by this definition do not denote independently observable individuals in the environment. They can not, therefore, appear as arguments in designated terms. But they can be used both in descriptions and in further definitions. For example,

$$\mathrm{TripStartTime}(i,t) \overset{\text{def}}{=} \mathrm{Trip}(i) \ \wedge\ \big(\exists e,p \bullet \mathrm{StartsTrip}(e,i) \ \wedge\ \mathrm{TakeOff}(e,p,t)\big).$$

The trip start time is defined to be the time of its starting take-off event.

3.7. *Using definition to classify phenomena*

Introduction of a general term, whether by designation or by definition, implies a classification of phenomena. Some events are take-off events of planes, and the others are not. It is only because we can classify that we can describe anything at all.

But we may reasonably expect that different purposes will demand different classifications. For example, in describing the environment of a PABX (Private Automatic Branch Exchange) for a telephone system, we may designate these phenomena (among others) as

$$\begin{aligned}\mathrm{Ringing}(p,t) &\cong \text{Telephone } p \text{ is ringing at time } t,\\ \mathrm{OffHook}(e,p,t) &\cong \text{ In event } e \text{ telephone } p \text{ goes offhook at time } t,\end{aligned}$$

$$\begin{aligned}&\text{SpeakerOn}(e,p,t)\\&\quad\cong\ \text{In event } e \text{ the speaker button of telephone } p \text{ is pressed at time } t.\end{aligned}$$

But in describing the requirements for a telephone system we may well want to form other, different, classifications of the same environment phenomena. We can do so by making suitable definitions, such as

$$\text{Answer}(e,p,t) \overset{\text{def}}{=} \big(\text{Ringing}(p,t)\ \wedge\ \big(\text{OffHook}(e,p,t)\ \vee\ \text{SpeakerOn}(e,p,t)\big)\big).$$

An answer event is defined to be a telephone event that occurs at a time when the phone is ringing and is either an offhook event or an event in which the speaker button is pressed.

In principle there is no limit to the number of classes to which an individual may belong, and no reason to try to force the classes into a hierarchical structure. Not all answer events are offhook events; and not all offhook events are answer events.

3.8. The discipline of designation and definition

The two tools, designation and definition, underpin an essential discipline in description. Every term used in every description must be either designated or defined, and its meaning must therefore be directly or indirectly grounded in reliable observation of the environment.

This discipline is a safeguard against some of the more insidious difficulties of requirements description. It inhibits the unthinking introduction of undefined terms that can be used with different intended meanings in different descriptions, or with meanings that subsequently prove hard to define unambiguously. In this way it contributes to the prevention of ontological drift [Robinson and Bannon 1991], in which as abstractions pass through the different subgroups of an organisation they are interpreted in terms of that particular community's set of meanings [Easterbrook 1993].

More generally, the benefit of the discipline is simply that with it we can know what our descriptions mean, and without it we can not. Readers of descriptions that lack explicit designations are compelled to treat the descriptions themselves as if they were designations. A rough initial idea of the meaning of a term, suggested by its natural language interpretation, is fleshed out by testing it against the assertions contained in the descriptions that have been read so far. If the description fits the putative meaning, that is a partial confirmation; if not, it is the meaning that must be adjusted. As the reading of the descriptions proceeds, this process of testing and adjusting interpretations must be carried out, more or less simultaneously, for every term that should have been designated.

Evidently, such a process precludes any critical check of the truth of the assertions encountered. An assertion that seems false leads merely to adjusting the reader's interpretation of the terms used, rather than to challenging the assertion itself. Only when the possibilities of interpretation are exhausted can the assertion be challenged.

This will not occur until either a formal contradiction is discovered or the interpretation of a term is strained to a point that is no longer credible.

The misunderstanding of requirements that can arise from a lack of explicit designations can have serious consequences. For example, a contributory cause to the 1979 accident at the Three Mile Island nuclear power plant was a control panel indication that a certain valve was shut when in fact it was open [Ferguson 1992]. The requirement formally expressible as

$$\text{IndicateValveShut}(v,t) \Leftrightarrow \text{ValveShut}(v,t)$$

– at time t the indicator for valve v must show that it is shut if and only if the valve is shut at that time – was wrongly implemented. The indicator in fact showed whether the current was on or off in the electromagnet that actuated the valve, not whether the valve itself was open or shut. The absence of a clear and explicit designation for the term ValveShut(v,t) allowed the mistake to be easily made.

3.9. Support for the discipline

To support this discipline, the notations and formalisms used must distinguish clearly between designated and defined terms, and between definitions and assertions. The meaning of the definition

$$\begin{aligned}&\text{ExpectedWidgetStock}(t,s)\\ &\overset{\text{def}}{=} s = \left(\sum tm, e, m \mid \text{WidgetMvmt}(tm,e,m) \ \wedge\ tm < t \ \bullet\ m\right)\end{aligned}$$

is quite different from that of the assertion

$$\text{WidgetStock}(t,s) \Leftrightarrow s = \left(\sum tm, e, m \mid \text{WidgetMvmt}(tm,e,m) \ \wedge\ tm < t \bullet m\right).$$

The difference is clearly indicated by the different symbols ($\overset{\text{def}}{=}$ and $\Leftrightarrow$), and by the fact that the predicate WidgetStock is designated, while ExpectedWidgetStock is not.

By contrast, the distinction between definition and assertion is less clear in a language such as Z. In a commonly adopted style of using the language [Wordsworth 1992] there is no easy and convenient way to distinguish these two schemas:

$$\begin{array}{|l}
\hline \multicolumn{1}{|c}{\mathit{Books\_Lent}}\\
\mathit{lent}\colon \mathbb{P}\ \mathit{Book}\\
\mathit{lent\_to}\colon \mathit{Book} \nrightarrow \mathit{Person}\\
\hline
\mathit{lent} = \operatorname{dom} \mathit{lent\_to}\\
\hline
\end{array}
\qquad
\begin{array}{|l}
\hline \multicolumn{1}{|c}{\mathit{Desks\_Assigned}}\\
\mathit{enrolled}\colon \mathbb{P}\ \mathit{Child}\\
\mathit{assigned}\colon \mathit{Desk} \nrightarrow \mathit{Child}\\
\hline
\mathit{enrolled} = \operatorname{rng} \mathit{assigned}\\
\hline
\end{array}$$

The schema on the left, we may reasonably suppose, is intended as a definition. It defines the term *lent* on the basis of the (implicitly designated) terms *Book*, *Person*, and *lent_to*. A book is *lent* is defined to mean that there is some person it has been lent to. But the description on the right is intended as an environment assertion. The terms *Child*, *Desk*, *enrolled* and *assigned* are all implicitly designated. The assertion is that no enrolled child is without an assigned desk.

3.10. Conclusion

Requirements engineering is not a branch of pure mathematics or logic: the meaning and applicability of an environment description depends crucially on its reliable interpretation in the environment. In requirements engineering we may not postpone interpretation until description is complete: without its interpretation a description at any level is literally meaningless.

The techniques discussed in this paper have been presented in terms of descriptions, but their underlying ideas are concerned with the phenomena of the system environment and how causal and constraint relationships among those phenomena may be reliably approximated and formalised. These ideas appear capable of application in many domains, especially those in which safety-critical systems are deployed.

Acknowledgements

The ideas expressed in this paper have been discussed and refined in joint work with Pamela Zave and Daniel Jackson. Thanks are also due to the anonymous reviewers, two of whom made constructive and helpful comments on an earlier draft.

References

Dardenne, A., A. van Lamsweerde, and S. Fickas (1993), "Goal-Directed Requirements Acquisition," *Science of Computer Programming 20,* 1, 3–50.

Easterbrook, S. (1993), "Domain Modelling with Hierarchies of Alternative Viewpoints," In *Proceedings of the IEEE International Symposium on Requirements Engineering*, IEEE Computer Society Press, Los Alamitos, CA, pp. 65–72.

Easterbrook, S. and B. Nuseibeh (1995), "Managing Inconsistencies in an Evolving Specification," In *Proceedings of the Second IEEE International Symposium on Requirements Engineering,* IEEE Computer Society Press, Los Alamitos, CA, pp. 48–55.

Feather, M. S. (1987), "Language Support for the Specification and Development of Composite Systems," *ACM Transactions on Programming Languages and Systems 9,* 2, 198–234.

Ferguson, E. S. (1992), *Engineering and the Mind's Eye*, MIT Press, Cambridge, MA and London, UK.

Jackson, M. (1995), *Software Requirements and Specifications,* Addison-Wesley, Reading, MA.

Jackson, M. and P. Zave (1993), "Domain Descriptions," In *Proceedings of the Second IEEE International Symposium on Requirements Engineering,* IEEE Computer Society Press, Los Alamitos, CA, pp. 56–64.

Jackson, M. and P. Zave (1995), "Deriving Specifications from Requirements: An Example," In *Proceedings of the 17th International Conference on Software Engineering,* ACM and IEEE Computer Society Press, Los Alamitos, CA, pp. 15–24.

Leveson, N. G. and C. S. Turner (1993), "An Investigation of the Therac-25 Accidents," *IEEE Computer 26*, 7, 18–41.

Neumann, P. (1995), *Computer-Related Risks*, Addison-Wesley, Reading, MA.

Robinson, M. and L. Bannon (1991), "Questioning Representations," In *Proceedings of the Second European Conference on Computer-Supported Cooperative Work ECSCW-91*, L. Bannon, M. Robinson, and K. Schmidt, Eds., Kluwer, Dordrecht.

Van Lamsweerde, A., R. Darimont, and P. Massonet (1995), "Goal-Directed Elaboration of Requirements for a Meeting Scheduler: Problems and Lessons Learnt," In *Proceedings of the Second IEEE International Symposium on Requirements Engineering*, IEEE Computer Society Press, Los Alamitos, CA, pp. 194–203.

Woodcock, J. and J. Davies (1996), *Using Z: Specification, Refinement and Proof*, Prentice-Hall.

Wordsworth, J. B. (1992), *Software Development with Z: A Practical Approach to Formal Methods in Software Engineering*, Addison-Wesley.