Connecting Viewpoints by Shared Phenomena

Michael Jackson
AT&T Research
Murray Hill NJ
and MAJ Consulting Ltd
101 Hamilton Terrace London NW8 9QY England
jacksonma@attmail.att.com
mj@doc.ic.ac.uk

A Contribution to the
Viewpoints '96 Workshop at FSE-4
October 14th & 15th 1996

1 Connecting Viewpoints

The term *viewpoint* is interpreted in many different ways [Kot96]. A viewpoint may be identified with a *stakeholder* — a person or group having a legitimate interest in the system to be built; with any *knowledge source* about the application domain; with a *participant* in the development process; with a *domain* (by which we mean the machine, or a human or other part of the environment that interacts with the machine); with a *projection* of the properties of any part of the system and the associated *descriptions*, notations and *methods*; or with almost any combination of these. Whichever intepretation is adopted, software developers who use more than one viewpoint must certainly consider how different viewpoints are related, and whether they are consistent [Eas96].

The central theme of this paper is that relationships among different viewpoints can be understood in terms of *shared phenomena*. This applies in particular to viewpoints regarded as:

- separate descriptions of distinct *domains*;
- separate descriptions of distinct *properties* of one domain; or
- separate descriptions of distinct system *requirements*.

In the body of the paper these relationships are briefly discussed in terms of shared phenomena, and some general considerations relevant to shared phenomena are explored.

2 Shared Phenomena

The fundamental idea of shared phenomena is clearly seen in CSP [Hoa85]. Each CSP process has an alphabet of classes of event in which it engages: for example, the alphabets of processes P and Q may be {*a,b,c*} and {*c,d*} respectively. Whenever an event of class *c* occurs both P and Q participate: they are participating in the same event. A shared event is not mediated in any way — for example, by a transmission channel. There is no notion that a *c* in P and the corresponding *c* in Q are distinct but simultaneous events. There is only one event.

States, of course, can be shared in the same way as events, although this is not done in CSP.

Shared phenomena form the connections among distinct domains in a very obvious way. The distinct connected domains may be the machine and its environment [Jac95a], or agents or domains recognised within the environment [Joh88, Jac95b]. For example, a passenger in a lift presses a button, and in the same event the controlling computer receives an input signal; or the controlling computer in a railway signalling system sets an output line to high, and in the same event a red signal light is illuminated.

Shared phenomena also form the relationships among descriptions of distinct properties. For example, a text worked on in a simple editor [Jac95c] may be described as a text string (for purposes of inserting, spell-checking and finding words) and also as a sequence of lines (for purposes of cursor movement and pagination). These descriptions are related by shared phenomena. The characters in the words in the text string description are the same characters, in the same order, as those in the line sequence description (excluding hyphens and newline characters); and upwards movement of the cursor by one line is the same event as some backwards movement of the cursor within the string.

Similarly, shared phenomena tie together distinct subproblems —that is, distinct projections of system requirements [Zav93, Eas96, Jac96]. In the requirements for appropriate caller features and appropriate callee features in a simple telephone system, a particular event *caller-hangs-up* in the callee requirement is the same event as a particular *hang-up* event in the caller requirement; and *callee-is-idle* in the caller requirement is the same state as

idle in the callee requirement.

3 A Sound Basis for Shared Phenomena

Because shared phenomena are the foundation of these important relationships in a development, it is essential to place them on a sound basis. This means choosing, and explicitly recording, sound answers to these questions:

- What are the phenomena of interest for the purpose in hand? and
- Which phenomena should be regarded as shared?
- How should phenomena of interest, and their sharing, be declared and described?

These questions are more difficult and subtle than may appear at first sight. The central difficulty arises from the fact that the world in which requirements are located is informal. For almost any proposed characteristic predicate we can easily find hard cases, of which we can scarcely say whether or not the predicate is true. To almost any proposed rule we can easily find an apparent exception, in which the given formulation of the rule breaks down. We can dispute endlessly about the meaning of almost any word in natural language.

This informality must not be carried over into the system. In human affairs it is often right to leave an area of uncertainty. Parliamentary drafting often leaves detailed discriminations to the courts: the judges will decide what is meant by “fair and reasonable”, or by “due care and attention”. But the computer is not human. It is a machine carefully constructed to be equivalent to a formal system. At its interface, of phenomena shared with the environment, the machine will exhibit a formally describable behaviour. We must be as confident as we can that this behaviour will produce the effects we want in the informal environment. An acceptable level of confidence demands that we must aim to reason as reliably about the environment as we can about the machine.

3.1 Designating Appropriate Phenomena

This reliability is achievable in many application areas, by techniques not entirely unrelated to those used to formalise the machine. Hardware engineers simulate precise discrete phenomena by careful control of continuous phenomena. If *ON* is 3v, and *OFF* is 0V, there are inevitably zones of uncertainty in the continuous transition over the full voltage range. But it can be bypassed by careful design. A tolerance band is defined — 3v±0.2v is *ON*, and 0v±0.2v is *OFF* — and the potential is simply not examined while it is rising or falling. So from the point of view of a discriminating circuit the voltage is always recognisable as definitely *ON* or definitely *OFF*. From the informal phenomena of continuous potential differences, a discrete formal *ON/OFF* phenomenon has been engineered.

In describing the informal environment we are not free to engineer it in quite this way. But we can be careful to designate as phenomena of interest [Jac95b] only those that can be recognised with enough reliability for the current context and purpose. For a system controlling the lights in a greenhouse, a light switch is either on or off; but for a system managing an experiment concerned with the degeneration of electrical contacts by arcing, this is not nearly reliable enough. A lift control system recognises the state *doors-obstructed*; a racetrack photo-finish system must make far finer discriminations.

We have considerable freedom to choose the phenomena of interest. Candidate terms for denoting environment phenomena may be:

- *Designatable terms*: terms denoting phenomena that are *reliably recognisable* by informally described criteria.
- *Definable terms*: terms that can be *defined by formal expressions* over designatable terms.
- *Vague terms*: terms that are neither designatable not definable.

Vague terms must be rejected. Their use in early discussion with stakeholders is unavoidable. But in written requirements and specifications they perpetuate uncertainty and confusion. This disadvantage is exacerbated, rather than mitigated, by attempts to negotiate agreements on the meanings of vague terms or to reconcile manifestly conflicting interpretations.

3.2 Using Definition

Developers should aim to use as few designatable terms as possible, building up most of the necessary terminology by formal definition. In this way the relationship between requirements descriptions and their subject matter can be made as simple and perspicuous as possible.

Consider, for example, the concept of a *flight* in a system concerned with airline schedules. Notoriously, this concept is hopelessly vague. But not many designatable phenomena are needed to support the practical purposes for which it is used. From these designatable phenomena — for example, *aeroplane, airport, person, take-off, land, embark, disembark* — enough terminology can be defined — for example, *trip, stage, round-trip* — to allow the concept of *flight* to be discarded and the system requirements put on a firmer foundation.

3.3 When Are Phenomena Shared?

In some cases, the sharing of particular phenomena is obvious and beyond dispute. The event in which car A collides with car B is indisputably the same event as that in which car B collides with car A.

But in most cases, deciding whether particular phenomena are shared between two distinct domains

demands the same kind of judgement as the original designation of phenomena. For example, we may choose to regard the pressing of a lift button and the receipt of the corresponding signal by the lift control computer as the same event. Of course there is some lapse of time while the current flows in the circuit; and there is some possibility, however small, that a broken wire will frustrate the receipt of the signal although the button has been pressed. But for purposes of controlling a lift the button press and the signal receipt can properly be treated as one and the same event.

In a more interesting example, the posting and delivery of a letter may in some circumstances be considered a shared event in which sender and recipient participate. Suppose we are concerned with a traditional game of postal chess. Although the Royal Mail introduces inevitable delay and uncertainty between posting and delivery, yet for the purpose in hand these may be ignored. While the postcard recording the progress of the game is in transit through the mail, no move can occur: effectively the world stops, and the transmission of the card may be regarded as an atomic event.

4 Three Aspects of Shared Phenomena

The sharing of phenomena must be described explicitly, for example by statements such as:

SHARED (writer.*write*, reader.*read*);

Each *write* event in the writer domain is also a *read* event in the reader domain.

In addition to specifying explicitly which phenomena are shared, we must specify what is shared in each phenomenon, and what is the nature of the sharing relationship. Three important considerations in describing sharing are *control*, *event projection*s, and *event classification*.

4.1 Control

Control is important because in general the participants in a shared phenomenon do not participate on an equal footing. In CSP participation is symmetric: for an event to occur, each participant must be ready to participate, but none takes the initiative. The initiative comes from an imaginary source that is constantly trying all classes of event: among those whose participants are ready one is selected at random. This is not intended as a realistic account of event occurrence. More realistically, each event is caused by one agent or domain, or by the machine. The other participants are passive. In some cases they may refuse or frustrate the event, but they never initiate it.

It is therefore necessary to indicate, for each class of phenomenon, where control resides. If events and states are declared within domains, a suitable notation may be:

DOMAIN writer; EVENT ?*write*; ...
DOMAIN reader; EVENT !*read*; ...
SHARED (writer.*write*, reader.*read*);

The marking ? or ! indicates that the phenomenon — in this case, an event — is respectively externally or internally controlled. We take the position that control — that is, the initiative in an event — must belong to one domain, never to two or more.

The same control consideration applies to shared states. For example, if that state of a sensor is shared between the machine and the domain in which the sensor is set and reset, we may declare:

DOMAIN floor; STATE !*f_sensor*; ...
DOMAIN machine; STATE ?*m_sensor*; ...
SHARED (floor.*f_sensor*, machine.*m_sensor*);

Naturally, the control declarations for a shared class of phenomenon must be consistent. In the domain declarations of each shared class, there must be one and only one declaration specifying internal control.

4.2 Event Projections

The true control situation is still more complex than this. The initiative in an event may be distributed, different participating domains being responsible for different aspects. For example, the computer and a disk drive both participate in a *read/write* operation; the computer initiates the occurrence of the event; but the drive determines the data to be transferred.

Further, individuals may play a role in an event in one domain but not in another. For example, in the computer domain a role in the *read/write* event may be played by the individual buffer into which the data will be read. The computer domain determines the buffer, just as the disk drive domain determines the data to be transferred. But the buffer is not an individual in the disk drive domain, and can not appear in the event declaration in that domain.

Essentially, then, we must decompose the declaration of each event class by separating the declaration of the event occurrence from the declaration of the roles played by individuals in each participating domain, and the specification of control over those roles. So we need a notation such as:

DOMAIN writer; EVENT ?*write*(!*data*); ...
DOMAIN reader; EVENT !*read*(?*value*,!*buffer*); ...
SHARED (writer.*write*(*data*), reader.*read*(*value*));

4.3 Event Classification

Event classification is important [Zav96] because there is no *a priori* reason to believe that the classification of events appropriate to one domain, to one projection of a domain's properties, or to one subproblem or requirement will be appropriate to another. Descriptive techniques are therefore needed that allow arbitrary event classification within and between domains.

This kind of event classification must take account of states. For example, taking a telephone receiver offhook when the phone is ringing is classified as an *answer* event, while taking it offhook while the phone is idle is classified as an *initiate-call* event.

5 Summary

Different viewpoints are related and tied together by shared designated phenomena. Choosing which phenomena are to be designated and which are shared is a fundamental decision about how we think the world fits together, how our descriptions fit together, and how requirements fit together. An effective viewpoints approach requires a sound basis in phenomenology in general, and shared phenomena in particular.

6 Acknowledgements

Much of the work on which this short paper is based has been done in conjunction with Pamela Zave or Daniel Jackson.

7 References

[Eas96] Easterbrook, Steve and Bashar Nuseibeh (1996), "Using ViewPoints for inconsistency management", *Software Engineering Journal 11,1*.

[Hoa85] Hoare, C.A.R. (1985), *Communicating Sequential Processes*, Prentice-Hall International.

[Jac95a] Jackson, Michael and Pamela Zave (1995), "Deriving Specifications from Requirements: An Example", In *Proceedings of the 17th International Conference on Software Engineering*, IEEE CS Press.

[Jac95b] Jackson, Michael (1995), *Software Requirements & Specifications*, Addison-Wesley, Reading, MA.

[Jac95c] Jackson, Daniel (1995), "Structuring Z Specifications with Views", *ACM Transactions on Software Engineering and Methodology, 4, 4*.

[Jac96] Jackson, Daniel and Michael Jackson (1996), "Problem Decomposition for Reuse", *Software Engineering Journal 11,1*.

[Joh88] Johnson, W.L. (1988), "Deriving Specifications from Requirements", In *Proceedings of the 10th International Conference on Software Engineering*, IEEE CS Press.

[Kot96] Kotonya, G., and I. Sommerville (1996), "Requirements Engineering with viewpoints", *Software Engineering Journal 11,1*.

[Zav93] Zave, Pamela, and Michael Jackson (1993), "Conjunction as Composition", *ACM Transactions on Software Methodology 2, 4*.

[Zav96] Zave, Pamela, and Michael Jackson (1996), "Where Do Operations Come From?", *ACM Transactions on Software Methodology, to appear*.