**Keynote Talk at ITG/SEV Symposium, Zürich, 29 September 1999**

# Problem Analysis and Structure

Michael Jackson
AT&T Research, Florham Park NJ, USA
and Independent Consultant, London, England

jacksonma@acm.org

mj@doc.ic.ac.uk

## 1. Introduction

Traditionally, thinking and research in software development has focused on solutions: on programs and on various abstractions that may be useful in designing and writing program texts. We have paid little or no attention to the problems that those programs are intended to solve. Even methods and approaches that claim the title of 'problem analysis' usually prove, on closer inspection, to deal entirely with putative or outline solutions; the problem to be solved must be inferred from its solution.

This solution-oriented approach may work well in a field where the problems are all well known and have been thoroughly described, classified and investigated — where innovation lies only in devising new solutions to old problems. But software development is not such a field. The versatility of computers and their rapid pace of evolution present us with a constantly changing repertoire of problems to whose solution software may be central. As a result, our field is underdeveloped in crucial respects. In particular, the repeated calls for professionalisation and for the establishment of a corpus of core software engineering knowledge are symptoms of a broad failure to identify what practising software developers should know if they are to be fit to tackle the problems of the many different application areas.

In this talk I want to sketch an approach to problem analysis and structuring that — I believe — avoids the magnetic attraction of solution-orientation. The approach is based on the idea of a problem frame. Problem frames characterise classes of problems that commonly occur as subproblems of larger, realistic, problems. The intention is to analyse realistic problems by decomposing them into constituent subproblems that correspond to known problem frames. This analysis guides the decomposition, gives warning of the concerns and difficulties that are likely to arise, and provides a context in which previously captured experience can be effectively exploited.

## 2. The World, Phenomena and Domains

Some problems are abstract in a mathematical sense, and do not partake of the physical nature of the world. Factorising large integers, finding cut sets of graphs and playing chess are examples of such problems. But most problems are located in the physical world. Such problems include controlling lifts, switching telephone calls, controlling the brakes of a car, bank accounting, managing theatre seat reservations, controlling a VCR and administering a library. In all these cases the effectiveness of a solution is to be evaluated in the physical world outside the computer. The problem is located in the world; the computer, executing our program text, is the solution.

### Phenomena

Because problems are located in the world, problem analysis must be concerned with the world and its phenomena. We need a phenomenology that has nothing to do with programming languages or object interaction, but everything to do with the physical world. An appropriate phenomenology includes:–

- *entities*, which are mutable individuals such as cars and people;
- *events*, recognised as individuals;
- *values*, which are immutable individuals such as integers and strings;
- *states*, which are time-changing relations over non-event individuals;
- *truths*, which are unchanging relations over non-event individuals; and
- *roles*, which are the participation of individuals in events.

Among these it is useful to recognise the class of *controllable phenomena* — events, state changes and roles — that occur on the initiative of one part of the world rather than another. For example, a keystroke is an event in which the user and the keyboard both participate, but it is controlled by the user. It is also useful to treat roles as distinct phenomena. In the keystroke the user controls both the event and the role that is the participation of a particular key; but in a disk read operation the reader controls the event while the disk controls the paricipation of the particular record that is returned.

## Domains

For purposes of problem analysis it is natural to recognise distinct parts of the world; we will call them *domains*. A domain can be thought of as a collection of related phenomena; the kinds of phenomena in a domain and the relationships among them constitute the *domain properties*. Domains may share phenomena: the only way two domains can interact is by an interface of shared phenomena. For example, a lift and its passengers interact because both the entry of a passenger into the lift car and the pressing of a floor request button are events shared by the lift domain and the passenger domain. The control computer interacts with the lift because switching on the lift motor is an event shared by the computer and the lift domain (and controlled by the computer).

The most fundamental distinction for problem analysis is between the machine domain — the computer and its software — and the problem domain — the world where the problem is located and the quality of its solution will be evaluated. These two domains must share phenomena if the problem is to be soluble.

## Descriptions

In very general terms, the process of problem analysis is concerned with these descriptions over the phenomena of the problem domain:–

- *The requirement. This is a description of properties that the domain does not possess intrinsically but are desired by the sponsor of the development. It will be the machine's task to endow the problem domain with those properties. For example: the property that the lift comes when a button is pressed.*
- The *domain properties*. This is a description of the properties that the domain possesses intrinsically, regardless of the behaviour of the machine. For example: the property that from floor n the lift can go only to floor n+1 or n–1.
- The *machine specification*. This is a description of the desired behaviour of the machine at its interface with the problem domain. For example: when button *n* is pressed [in certain circumstances] the machine must set the lift motor polarity to *up* and set motor power to *on*. Although this is a description of machine behaviour, it is expressed entirely in terms of problem domain phenomena: the shared phenomena at its interface with the machine belong, of course, both to the problem domain and to the machine domain.

The formal criterion for success in a development is an entailment relationship among these descriptions:

> *machine specification, domain properties* ⊢ *requirement*

If the machine behaves as specified and the domain has the described intrinsic properties, then it is impossible for the requirement not to be satisfied. If the machine detects button presses and sensor states and operates the lift and door motors, all in accordance with the specification, and if the lift position and behaviour are related to the sensor states and motor settings as described in the domain properties description, then the lift will come when the button is pressed. Essentially, the intrinsic domain properties bridge the gap between the phenomena mentioned in the requirement and the phenomena directly accessible to the machine at its external interface to the world.

## 3. Elementary Problem Frames

The account just given of problem analysis is too general. Real problems are more specific. A problem frame captures the characteristics of a specific tightly constrained class of idealised problems. These problem classes correspond to intuitive notions of different kinds of problem, but make the intuition more precise. They stipulate the structure and characteristics of the requirement, of the problem domain — possibly structuring it as two or more domains — and of the interfaces among domains.

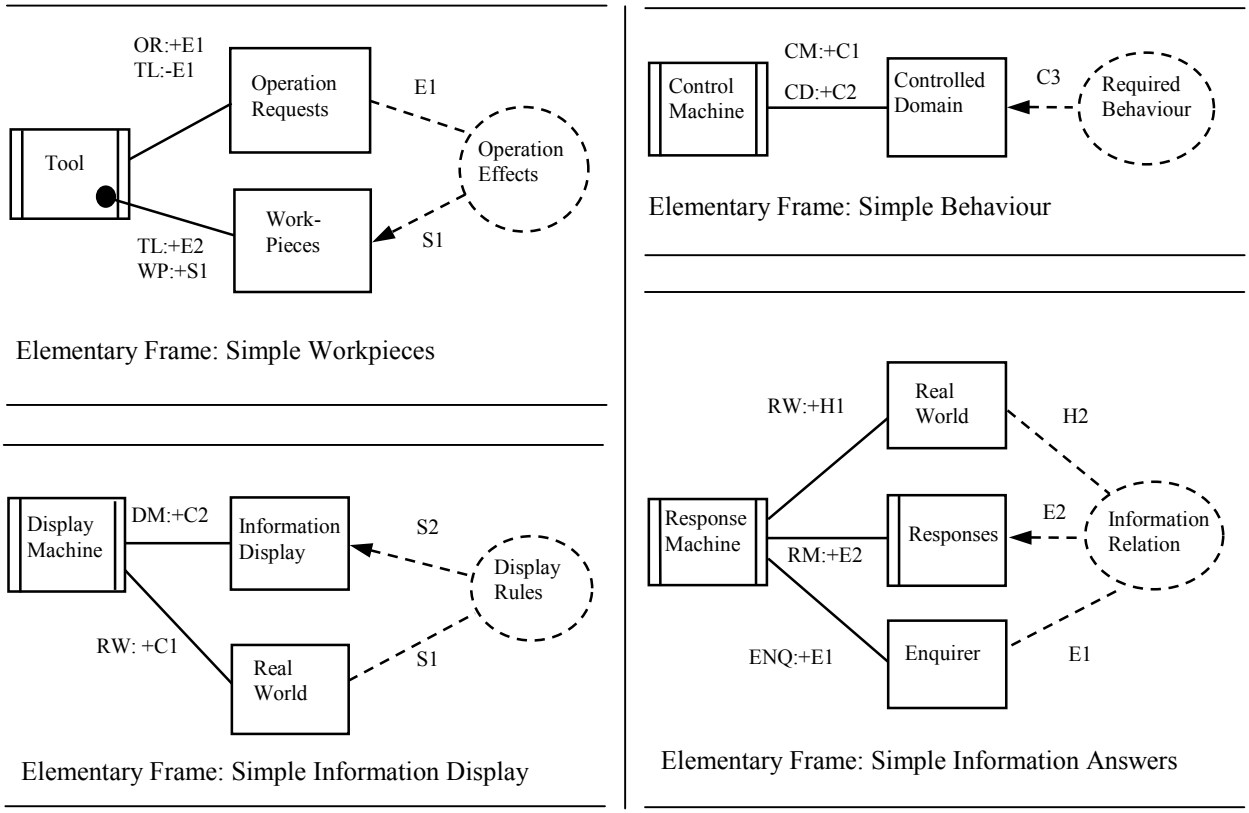


Figure 1: Four Elementary Problem Frames

Figure 1 shows the problem frame diagrams of four elementary problem frames. Reading clockwise from the top right they are:−

- *Simple Behaviour. This is an idealised form of a simple control problem. The requirement (Required Behaviour) is to impose a certain behaviour on the problem domain (Controlled Domain). The requirement is expressed in terms of controllable phenomena C3. The interface between the machine (Control Machine) and the controlled domain consists of shared controllable phenomena — C1 controlled by the machine and C2 controlled by the controlled domain. Typically, the controlled domain is partly autonomous and partly responsive to the phenomena C1.*

  A central concern is the adequacy of the information conveyed to the machine by the phenomena C2 for the machine to implement an effective control rule by controlling the phenomena C2. ABS is an example of a simple behaviour problem.

- Simple Information Answers. This is an idealised form of a simple information system (*Response Machine*) that answers enquiries. Enquiries in the form of an unstructured stream of events E1 come from an autonomous *Enquirer*; the machine creates its answers (*Responses*) by events E2. The subject of the enquiries is the *Real World*, which may be static (having no controllable phenomena); if it is not static it is autonomous, controlling all the phenomena H1 at its interface with the machine. The requirement (*Information Relation*) stipulates a relationship between the answer events E2, the enquiry events E1, and the phenomena H2 of the real world.

  A central concern is the use of the interface phenomena H1 to make inferences about the phenomena H2 that are the subject of the requirement. Provision of stock prices is an example of a simple enquiry problem.

- Simple Information Display. This is an idealised form of a simple information system (*Display Machine*) that maintains a continuous display of information (*Information Display*) about an autonomous dynamic *Real World*. The requirement (*Display Rules*) stipulates the state S2 of the display for each state S1 of the real world. The display is reactive, changing its states S2 in response to the machine-controlled phenomena C2.

A central concern is the use of the interface phenomena C1 to provide inferences about the phenomena S1 that are the subject of the requirement. Controlling the display in a hotel lobby that shows the current positions of the lifts is an example of a simple information display problem.

- Simple Workpieces. This is an idealised form of a problem in which the machine (*Tool*) acts as a simple tool for the creation and manipulation of text or graphic objects (*Workpieces*). The user of the tool autonomously issues an unstructured stream of commands (*Operation Requests*) E1, which the machine may sometimes inhibit. The workpieces are regarded as given — that is, their design is not considered to be a part of the workpieces problem itself. They are state reactive: that is, their behaviour consists only of changing their states S1 in response to events E2. The heavy dot on their interface with the machine indicates that they are contained in the machine: that is, all their phenomena are phenomena of the machine.

  A central concern is that the machine must inhibit invalid operation requests E1 (such as a deletion request for a non-existent workpiece), and must convert valid requests E1 into appropriate combinations of invocations E2 at the interface with the workpieces. The control of setting a VCR memo to record a TV program is an example of a simple workpieces problem.

## 4. Problem Decomposition

The elementary problem frames deal only with simplified idealised problems. Even when extended by a number of common variants and composites, the corpus of frames does not encompass many — perhaps any — problems of realistic size and complexity. Dealing with a realistic problem means decomposition into subproblems. An adequate corpus of frames is one in which we can always find a set of subproblems to give an appropriate decomposition of any realistic problem. There are several possible approaches to the decomposition task. In this section we mention three of them.

### Outside-In Decomposition

Sometimes the problem in hand seems to fit no known frame even approximately. It may then be helpful to decompose the problem by working from the outside towards the inside, as it were. The approach here is to try to find recognisable parts or aspects of the problem that correspond to known frames, and analyse them in the context of those frames. Then they may be regarded as solved problems, and the parts and aspects of the original problem that remain to be solved can be considered without the added complication of the already solved subproblems.

This approach is essentially an iterative application of the often-quoted heuristic “find a piece of the problem that you can solve”. If the approach succeeds, the original problem is eventually whittled down to a simple nucleus that fits a known frame.

### Inside-Out Decomposition

Sometimes the problem in hand seems to fit a known frame approximately, but exhibits difficulties that frustrate the pure application of the frame. These difficulties themselves give rise to subproblems that may be recognisable as fitting other frames in their own right. For example, one form of difficulty is a *connection* difficulty: it may be that some information needed by the machine is not available directly when it is needed. It may then be possible to cast the difficulty as an information answers subproblem in which the original machine plays the part of the enquirer. Another kind of difficulty is an *identities* difficulty, in which the machine shares a set of event or state phenomena with the problem domain but does not share the associated roles that identify the participating domain entities.

This approach can be thought of as working from the inside towards the outside, where the inside is the frame that seems to fit approximately and the outside is the surrounding set of difficulties. The core problem can be analysed on the assumption that the difficulties will be overcome in the solutions to the subproblems that capture them. This approach, too, is an application of a well-known heuristic: “solve a simpler problem”.

### Recognising a Standard Composite Frame

Although the elementary frames form the basis of the technique of problem analysis and structuring advocated here, a more fully developed stage of the technique will have a rich set of composite frames. It may be expected that a substantial part of a realistic problem, or even, occasionally, the whole of it, will fit a known composite frame. To recognise and

exploit this fit is to apply the heuristic "the best method is to have solved the same problem before".

One example of a composite frame is the Interactive Workpieces frame which, unlike the Simple Workpieces frame, includes an Interactive Screen domain at which the user can interact with the Tool by viewing the workpiece states and entering operation requests by a mouse or similar device. This composite frame, of course, has a solution in the form of the MVC (Model-View-Controller) framework, well-known in object-oriented design. Another example of a composite frame is an information system with a model. Where the shared phenomena are inadequate or untimely in an information problem, the difficulty can often be overcome by introducing a model domain and decomposing into two subproblems. In one subproblem the machine builds the model from the real world; in the other it uses the model to maintain the display or to answer the enquiries.

This kind of composition is closely analogous to the situation in established branches of engineering, where standardised products — such as cars and television sets and bridges — are elaborate composites of standardised components. The value of a repertoire of well-understood composite frames is, of course, that understanding a composite frame means a lot more than understanding its component subproblems: it means also understanding how the subproblems fit together, being aware of the concerns and difficulties that arise from the composition itself, and knowing how to fit the subproblem solutions together into a satisfactory solution to the original composite problem.

## 5. A Realistic Problem

To illustrate the problem frame technique we take the problem of controlling a package router. Here is the problem statement, adapted from [Swartout & Balzer 82]:–

> "Packages with bar-coded destination labels move along a conveyor to a reading station where their package-ids and destinations are read. They then slide by gravity down pipes fitted with sensors at top and bottom. A delay is introduced between successive packages; the delay is smaller between two packages to be routed to the same destination.
>
> "The pipes are connected by two-position switches that the computer can flip (when no package is present between the incoming and outgoing pipes). The configuration of pipes therefore forms a tree.
>
> "At the leaves of the tree are destination bins corresponding to the bar-coded destinations. A package can not overtake another either in a pipe or in a switch. However, because the packages are of varying shapes and sizes, they slide at unpredictable speeds and may therefore get too close together to allow a switch to be set correctly. A misrouted package may be routed to any bin, an appropriate message being displayed.
>
> "The system must route packages to their destination bins by setting the switches appropriately for each package as it slides down the pipes of the tree."

### Inside-Out Approach

At first sight, this is a simple behaviour problem. Figure 2 shows the problem diagram with the part names of the simple behaviour frame superimposed.

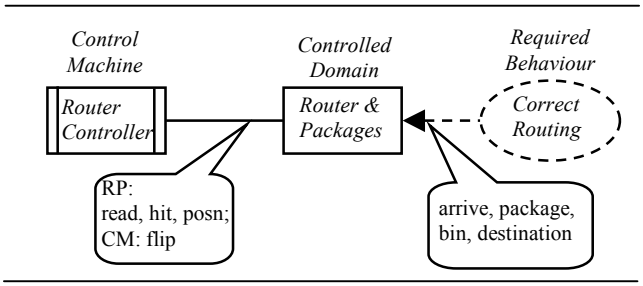


Figure 2: Package Router as a Simple Behaviour Problem

The interface between the Router Controller and the Router & Packages domain, shown in the callout, is as follows:–

- The Router and Packages domain controls *read* events in which the barcode of a package entity is read, and the associated role participation of the resulting barcoded string.
- The Router and Packages domain also controls *hit* events in which a package hits a sensor, and the associated role participation of the sensor.
- The Router and Packages domain controls the *posn(sw,pos)* states, which are the physical positions — left or right — of the switches.
- The Router Controller machine controls the *flip* events, in which a switch is flipped, and the associated role participation of the flipped switch.

The requirement Correct Routing is concerned with the following phenomena:–

- *arrive* events in which a *package* arrives at a *bin* at a leaf of the tree, and the associated roles which are the participation of the package and the bin.
- destination truths relating a package to a *destination* barcoded string, and corr truths relating the barcoded strings to the corresponding destination bins.

## A Connection Difficulty

The briefest attempt at describing domain properties that can close the gap between the requirement and the machine — if carried out with a properly meticulous attitude to the phenomena — will show at once that there is a connection difficulty. The essence of the difficulty is that the requirement is concerned with roles and truths involving *package* entities, but packages appear in none of the roles shared by the Router Controller. For example, the Router Controller can detect that a sensor has been hit, but can not detect which package is responsible.

The difficulty can be dealt with by a composite information frame with a model domain. Figure 3 shows the subproblem in which the model is built:–

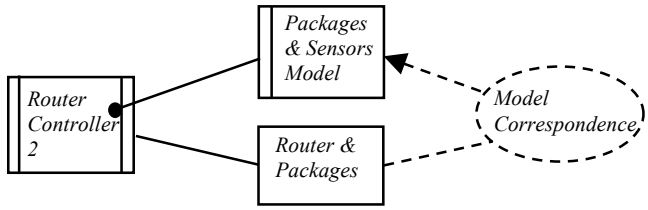


Figure 3: Making a Model Domain for the Packages and Sensors

Because packages do not overtake each other in the pipes and switches, it is possible to regard the packages in the router as forming a set of queues: on each read event at the reading station a new package enters the tail of the queue in the topmost pipe, and on hitting a sensor at the bottom of a pipe it moves from the head of one queue to the tail of another. The model domain, *Packages & Sensors Model*, contains a representation of these queues and a *destination* attribute for each queue element. The destination attribute is assigned when the barcoded destination of the package is read at the reading station.

The model domain can then be interrogated by the Router Controller in the behaviour problem to answer the question: "what is the *barcoded destination* of the package that participated in the most recent *hit* event in which this particular *sensor* participated?" The answer to this question solves this connection difficulty: when a package arrives at the sensor guarding a switch its destination is known.

## Two Identities Difficulties

The problem offers two clear examples of the identities difficulty. The first concerns the sensors and switches. Each sensor and switch is connected to a particular port — a register or sense line — of the machine. When the machine detects a hit it detects it at a particular port, but the identity of the sensor is not made explicit. A similar difficulty arises for the switches.

The second identities difficulty concerns the barcoded destinations and bins. Each bin can be associated one-to-one with the sensor guarding its entrance, but this does not help: the machine has no access to the mapping between the bin sensors and the strings. Effectively, therefore, it has no way of determining which is the destination bin for a particular barcoded string.

Both of these difficulties are solved in the standard way for identities difficulties: the mapping must be made explicit, and put in a form accessible to the machine. Creating the mapping is a simple problem, perhaps fitting the Workpieces frame. The mapping is then used by the router controller to identify the destination of each package with the bin sensor that is its eventual goal.

## Another Connection Difficulty

Another connection difficulty still lurks in the problem. When a package arrives at the sensor guarding a switch the machine must flip the switch or not according to the required routing of the package. But the machine has no access to the necessary routing information: that is, it has no way of determining whether a particular bin can be reached from a particular switch, and if so, whether by its left or its right exit.

The solution to this difficulty is another model, but this time of a static domain. As often happens with static models, it will probably be necessary to create the model with the help of a human informant, as shown in Figure 4:–

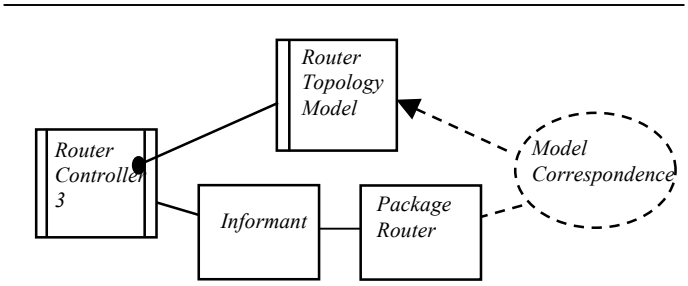


Figure 4: Making a Model Domain for the Router Topology

This static model provides all necessary information about the router topology: Which sensors are on which ends of which pipes? Which pipes feed which switches? Which pipes leave which switches? Which pipe leaves the reading station? Which bins are guarded by which sensors? From this information the router controller can determine the package routes to their destination bins.

## An Information Display Problem

The display of an appropriate message when a misrouted package arrives at a wrong bin is, of course, a simple information display problem. When each package arrives at the sensor guarding a bin, the bin identity can be compared with the package's destination and a message produced in the case of a mismatch.

## A Final Word on Composing the Solution

To compose the solutions of subproblems it is necessary to consider the scheduling of their machines.

In the present case the composition is fairly easy. Evidently the machines implementing the solutions to the identities difficulties, along with the machine that builds the static model of the router topology, must be run to completion first. Subsequently all the machines can run in parallel, essentially synchronised by the *read* and *hit* events controlled by the *Router & Packages* domain. That is, when one of those events occurs, each machine reacts to the event and returns to a quiescent state. Careful consideration may be needed to order the machines' reactions appropriately: the information on which the answer to a question is based must be established before the question is answered.

## 6. Summary

This talk has necessarily been brief and somewhat superficial, but I hope it has given a reasonably clear sketch of the problem frame approach. The chief points of the approach are these:–

- Problems are located in the world, not in the computer. The computer and its software are the solution. It follows that problem analysis must pay meticulous attention to the phenomena of the world and to the characteristics of its constituent domains and of the interfaces between them.
- Large and realistic problems can be seen as compositions of small problems. The small problems must be of recognised classes, both to guide the decomposition and to provide an intellectual structure within which we can capture, develop and disseminate a growing body of knowledge.
- Problem structure is often a parallel composition of subproblems. Hierarchical and embedded structures are also found, but parallel structuring is the commonest. The right metaphor for problem structure is not the bill-of-materials assembly structure but the superimposition of CYMK separations in the printing of a four-colour graphic.
- Each subproblem is concerned with some parts of the world, and the subproblems of one problem are connected by the world phenomena they have in common.
- Problem frames provide a basis for the approach by characterising a repertoire of problem classes in terms of their phenomena and domains.

There are further discussions of the Package Router problem in [Balzer et al 82] and in [Jackson 96]. There is further discussion of problem frames in [Jackson 95] and [Jackson 99].

## References

[Balzer et al 82] Robert M Balzer, Neil M Goldman and David S Wile. Operational Specification as the Basis for Prototyping; ACM Sigsoft SE Notes Volume 7 Number 5 pages 3-16, December 1982.

[Jackson 95] Michael Jackson; Software Requirements & Specifications: A Lexicon of Practice, Principles and Prejudices; Addison-Wesley, 1995.

[Jackson 96] Daniel Jackson and Michael Jackson; Problem Decomposition for Reuse; Software Engineering Journal Volume 11 Number 1 pages 19-30, January 1996.

[Jackson 99] Michael Jackson; Problem Analysis Using Small Problem Frames; Proceedings of WOFACS '98, South African Computer Journal 22, pages 47-60, March 1999.

[Swartout & Balzer 82] William Swartout and Robert Balzer; On the Inevitable Intertwining of Specification and Implementation; Comm ACM 25,7 pages 438-440, July 1982.