Extended abstract: Dagstuhl Seminar 09381, 13-18 September 2009

Refinement, Problems and Structures

Michael Jackson

The Open University
jacksonma@acm.org

Introduction

Refinement is a powerful reasoning tool in software development. success in its use for program construction encourages the hope of equal success in the construction of dependable computer-based systems, especially in safety-critical applications. These are systems in which the computer, with the software it executes, forms only one component of the whole system. The other components are other engineered artifacts and systems of many kinds, parts of the natural world, and human beings who are users or operators of the system or in any other way participate in its behaviour.

The criterion of dependability of such a system is not to be judged in the software alone, but rather in the behaviour of the whole system. This behaviour is constrained in two ways. First, it is constrained by the given properties, characteristics and behaviours of the relevant parts of the world. For example: in a lift control system the physical arrangement of the shaft prevents the lift car from travelling from the second floor to the fourth without passing the third; in the same system the electrical and mechanical equipment ensures that if the hoist motor is switched on, with upwards polarity, the lift car will rise in the shaft. Second, system behaviour is constrained by execution of the software, monitoring and controlling the relevant parts of world both through interfaces to parts directly connected to the computer, and through the interactions of those parts with remoter parts. The lift system is dependable if the whole system dependably satisfies its requirements: in response to users' requests it carries them safely and efficiently from floor to floor.

Developing systems of this kind is different from developing programs. In program development the goal is to achieve a certain input-output behaviour of the computer by constructing a text in a programming language. Effectively, the programming language and the semantics of its execution are formally defined. The input-output behaviour to be achieved—the program specification—is also formal. In refinement-based development the program text is developed from the formal specification, each step justified by the known formal properties of the programming language.

In system development, by contrast, the requirement to be satisfied is a certain behaviour of the whole system: the computer's input-output behaviour is merely an instrumental means to this end. Usually the requirement itself is not formally specified, so the root of the refinement tree is not well-defined. Further, the relevant parts of the system outside the computer—the *problem world domains*—are not themselves formal. Their given behaviours and properties can be formally described, but a formal description can never be more than an approximation: appeal to the formal description is therefore never fully reliable. Further yet, systems are increasingly expected to embody a proliferation of functional features whose mutual interactions add another order of complexity.

Contrivances and Operational Principles

We shall regard systems of the kind we are concerned with as *contrivances*, in the sense explained by the physical chemist and philosopher Michael Polanyi. One broad class of contrivances comprises physical inventions such as clocks, telephones, locomotives and cameras. A contrivance is characterised by its *parts*, interacting according to the *operational principle* of the contrivance to achieve its *purpose*.

Figure 1 depicts the parts and purpose of the pendulum clock invented by Christiaan Huygens in 1656 and constructed by Salomon Coster in 1657. The rectangles represent the physical parts of the clock and the progression of time, the solid lines connecting them representing interactions among the parts. The dashed oval represents the purpose of the clock, which is to ensure that the positions of the hands on the clock face correspond to the passage of time (the arrowhead indicating that the purpose is to constrain the hands to match the time, not vice versa).

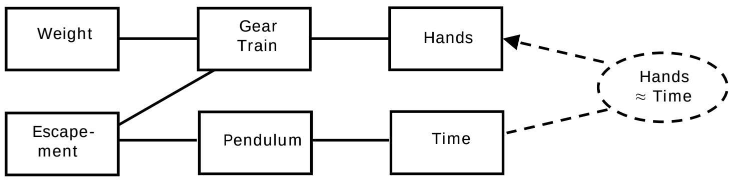

Fig. 1. A pendulum clock

The operational principle of the clock is readily explained. The falling weight drives the gear train, which drives the hands and the escapement. Rotation of the escapement shaft is constrained by the pendulum, each swing releasing one tooth of the escapement and receiving an impulse to maintain the pendulum's momentum. Gear train rotation is therefore proportional to the number of swings. The swing period is roughly constant, so the hand positions correspond to elapsed time.

Polanyi points out that the operational principle is expressed in terms of what he calls the *logic of contrivance*, and that this is distinct from scientific knowledge. It explains "how its characteristic parts—its organs—fulfil their special function in combining to an overall operation which achieves the purpose of the machine. It describes how each organ acts on another organ within this context." Scientific knowledge comes into play only within the given structure of the design of the contrivance and its known operational principle. Science can then explain the success or failure of a particular clock or a particular design, and calculate physical values that will allow the contrivance to fulfil its purpose. What is the swing period of the pendulum? Why do we expect it to be nearly constant? Are the gear ratios between the escapement and the hands correctly calculated? Does the escapement deliver an impulse large enough to counteract the slowing of the pendulum by friction and air resistance? Is the weight heavy enough to overcome the gear train friction? All these are questions of science, but they are posed only in reference to the contrivance and its operational principle. The operational principle tells us how the system is intended to work, and science tells us whether it will actually do so, and how well.

A contrivance is designed to achieve its purpose in a specific context, and is not expected to operate successfully outside that context. The pendulum clock must be stably positioned, in calm air and in a vertical orientation, on the earth's surface; it cannot operate successfully in a moving carriage, or in a ship at sea. The constancy of the pendulum's period depends also on a stable ambient temperature: in summer the clock ran slower as the pendulum period increased as its length expanded in the heat.

Systems As Contrivances and Problems

Systems of the kind we are considering can be similarly regarded as contrivances. Figure 2 shows the configuration of a system whose purpose is to ensure orderly and safe traffic of vehicles and pedestrians at a very complex road crossing.

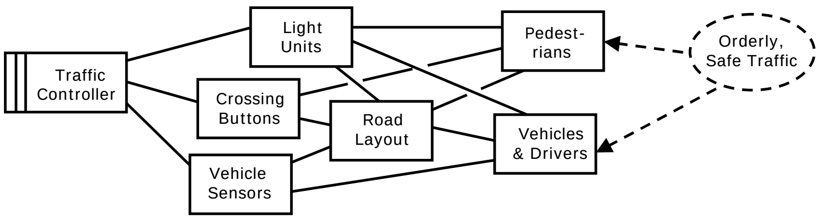

Fig. 2. A traffic control system

Again the rectangles represent parts of the system and the oval represents the system's purpose. The striped rectangle represents the computer and its software, directly connected to the light units, the pedestrian crossing buttons and the vehicle sensors embedded in the road.

Unlike the depiction of the pendulum clock in Figure 1, this representation depicts not only the system and its parts and purpose, but also a development *problem*. The problem is to discover, invent, or specify a Traffic Controller part whose external behaviour—by its interactions with the light units, crossing buttons and sensors—will ensure satisfaction of the system's purpose. To solve this problem the developer must investigate and understand—and then respect and exploit—the given properties of the other parts of the system. How fast do the pedestrians make their way across the various crossings? How fast do the cars go? How do the various streams of traffic intersect? What is the protocol for operating the light units? How reliably do drivers stop at red lights? These given properties, like the properties of the clock parts, depend on the specific context assumed for the system. If the junction is near an old-age home some pedestrians may walk very slowly; if one of the feeder roads is a motorway many vehicles can be expected to be driving above the legal speed limit; if an industrial plant is nearby there will be an unusually high number of very large vehicles; and so on.

Simplistically, we can imagine that the Traffic Controller specification and code can be developed by a refinement progression that moves across the diagram from right to left, from the requirement to the external behaviour of the machine. First the required (orderly and safe) behaviour of the pedestrians and the vehicles and drivers is refined. Relying on the given properties of pedestrians and vehicles, this behaviour is then refined into a required behaviour of the light units, crossing buttons and sensors. Relying on the properties of these devices, a final refinement produces a required external behaviour of the Traffic Controller machine. At each refinement step the problem is *progressed* [2,3] towards a software specification.

Problem Characteristics for Simplicity

For any particular system, feasibility of the simplistic view of a refinement process mentioned in the preceding section depends on several factors. The most important factor is the simplicity of the system, evidenced in a simple operational principle: the system must exhibit certain *unities* of purpose and structure. Some of these unities, with illustrative counterexamples of complexity, are:

- Unity of purpose: fails for an air traffic control system in which a certain horizontal and vertical separation is to be maintained, but if that proves impossible some other rule is to be applied.
- Unity of problem domain role: fails for a customer support system in which staff perform operational tasks and are also assigned as 'personal assistants' to changing groups of customers.
- Unity of problem domain properties: fails for a lift-control system in which the lift service function depends on faultless equipment behaviour and the safety function depends on diagnosis of equipment faults.
- Unity of system context: fails for a railway operations system in which train scheduling must assume fixed track configuration and availability and the track maintenance function must manage changes to track configuration.

A system exhibiting these and other unities has a simple operational principle. The intended working of the system can be explained in a simple traversal of the system configuration, saying at each step of the traversal how one part behaves and how it interacts with its neighbours. Scientific—or mathematical—knowledge and reasoning are invoked within this structure to validate the explanation in detail, both locally and end-to-end, and to calculate the behaviour that the machine part of the system must have if it is to ensure satisfaction of the requirement.

Decomposition and Recombination

We address complexity by decomposing a complex development problem into *subproblems*. A subproblem has the form of a problem, with its own machine, problem world, purpose and operational principle; being a problem in this sense, it can also be viewed as a system. A successful decomposition produces simple subproblems exhibiting the unities of the kind mentioned in the preceding section.

Subproblem decomposition produces *projections* of the problem. Projections may be chosen in many different dimensions. A projection in space may separate consideration of distinct problem domains: in the traffic control system, determining the locations of vehicles from the sensors does not involve the pedestrian crossing buttons. A projection in time may separate distinct phases and modes of system operation: in an avionics system, control of an aircraft while it is taxiing is separated from control of take-off. A projection in context may separate control of the lift equipment into assumed-healthy and potentially-faulty.

In any decomposition the complexity of the resulting parts has two sources: the inherent complexity of the part itself, taken in isolation; and the complexity due to its interactions with other parts. Traditional approaches to decomposition conflate these two sources, often frustrating the goal of simplicity. For example, program decomposition into a procedure call hierarchy is *embedded decomposition*, in which the parts are embedded in the whole by precisely matched call interfaces. The structure of a relational database schema results from *jigsaw decomposition*, in which the whole consists only of its parts, which must fit together by matching key values.

For systems of the kind we are considering, a *loose decomposition* is preferable, in which the task of identifying the parts is clearly separated from the task of recombining them into a whole. This separation permits a productive *oversimplification* of subproblems, in which each can be analysed in a restricted context in which the unities are preserved. Only in the later stage of recombination are the subproblem interactions addressed, and solutions devised for any difficulties they may pose. For example, in the railway operations system the train scheduling and track maintenance subproblems are first considered separately, oversimplified by their respective contexts of constant track configuration and complete absence of trains. Only when these subproblems are well understood is their recombination addressed. In designing the recombination it may, of course, be necessary to modify either or both of the oversimplified subproblems; but the need for this modification and its design can be more reliably and confidently carried out at this later stage.

The development process can be seen to have a top-down decomposition facet, and a bottom-up recombination facet. Locally, decomposition must logically precede recombination, but globally the two facets may be interleaved. In this process, structure, and human comprehension of operational principles, provides the essential framework. Formal techniques can be effectively deployed within this framework, in analysing and designing individual subproblem structures and their recombination.

References

[1] Michael Polanyi; Personal Knowledge: Towards a Post-Critical Philosophy; Routledge and Kegan Paul, 1958 and U Chicago Press, 1974.

[2] Robert Seater, Daniel Jackson and Rohit Gheyi; Requirement Progression in Problem Frames: deriving specifications from requirements; Requirements Engineering 12, 2 pages 77-102, April 2007.

[3] Zhi Li, Jon G Hall and Lucia Rapanotti; From requirements to specifications: a formal approach; Proceedings of the 2006 International Workshop on Advances and Applications of Problem Frames, pages 65-70, Shanghai 2006.