Chapter III

Operationally Complex Systems

Information Systems and Operational Systems

Since the middle of the twentieth century, electronic systems have been designed to perform increasingly complex combinations of tasks. Many of these systems fall within the domain of information technology such as database management, text editing, and graphics and image processing systems. However, some of the most complex systems are in the domain of direct control of physical equipment. The equipment controlled can be nuclear power stations, chemical manufacturing plants, oil refineries, or telecommunications networks. The electronic systems control these facilities in real time with little or no human intervention. These operationally complex systems generally experience constraints on their architectural form which are significantly more severe than the constraints experienced by information technology systems. A human brain controls very complex physical "equipment" (i.e. a body) in real time with little or no external intervention. Analogous constraints on architectural form therefore exist, although the resultant form has minimal resemblance to any current commercial electronic system.

An operational system (or control system) is one which acts upon its environment and itself in order to achieve objectives. A very simple example of an operational system is a thermostat with the objective of keeping the internal temperature of a refrigerator within a specified range. The thermostat receives an electrical signal which provides information about the temperature within the refrigerator, and turns cooling on if the signal indicates a temperature above the range and off if below the range. In addition, the thermostat must respond to its input signal within a reasonable period of time.

A slightly more complex operational system would be a refrigerator thermostat with the additional objective of minimizing energy consumption as far as possible. Additional behaviours would include turning cooling on at a range of different levels where higher levels cooled faster but were less efficient in using energy. An additional sensor input signal could be an indication of the temperature external to the refrigerator. The decision on current behaviour would depend upon an interaction within the control system between information on past and current temperatures inside and outside the refrigerator and the two objectives.

For example, rates of temperature change inside and outside the refrigerator might be estimated.

Operationally Complex Systems

An operationally complex system is one in which the number of possible behaviors is very large, there are many potentially conflicting objectives, considerable interaction is required within a large body of information derived from sensory inputs to determine appropriate behaviour at each point in time, and the time available after occurrence of an input condition in which to determine appropriate behaviour is short.

To illustrate these factors, consider two types of operationally complex electronic systems, both of which have been in use for many years. One is the flight simulator [Chastek and Brownsword 1996], the other is the even more complex central office telecommunications switch [Nortel Networks 2000]. A current flight simulator may have one or two million lines of software code, and simulates the environment experienced by the pilot of an aircraft such as a military jet, including changes to that environment caused by the actions of the pilot. A central office switch with twenty or thirty million lines of code is physically connected with 100 thousand telephones and computers, and provides their users with a wide range of telecommunications services on demand.

The behaviours controlled by a flight simulator include changing the images on monitors simulating the view through cockpit windows, changing the settings on cockpit instruments, and moving the emulated cockpit platform in three dimensions as the pilot performs tasks including takeoff, landing, maneuvering in flight including combat situations, midair refueling, and deploying weapons systems. The appropriate system behaviour is determined by an interaction between: firstly the past and current settings of the simulated aircraft components such as engine, control surfaces, aircraft component failures, radar and weapons; secondly the simulated environment such as wind, turbulence, the ground or aircraft carrier, and other aircraft; and thirdly the actions of the pilot. The system must generate sufficiently accurate behaviour fast enough that the simulated response is sufficiently close to real response for pilot training to be effective. For example, slow system responses could mean that training was dangerously counterproductive. In addition, timing mismatches between visual and movement changes can result in physiological reactions known as simulator sickness, even when the mismatches are too slight to be consciously perceived.

The basic tasks of a central office switch include establishing and taking down connections between telephones and computers using a limited pool of connectivity resources; collecting billing information; collecting traffic information to determine when and where upgrades are required; making changes to services; and identifying problems. However, these basic tasks are subject to a number of considerations which make the system much more complex than might appear superficially.

The first consideration is that there are thousands of features which must be taken into account in the course of establishing a connection. Features like call forwarding, emergency calls, and toll free calls to institutions rather than to specific telephones all require decisions on destination, routing, and billing based on information in addition to the actual number

dialed. Features like conferencing and call monitoring for law enforcement require connectivity to be established between more than two points. Features like fraud prevention must intervene before suspect calls are established. Features like prioritization of calls in national emergencies must permit some calls to have priority in taking connectivity resources from the common pool. Data and quality of service features must make it possible to select from different types of end-to-end connectivity. Many of these features interact in the sense that the activity of one feature can change the way another feature operates. An example would be when one telephone has the service of only accepting calls from a specific set of other telephones, and one of those other telephones invokes a service of forwarding all its calls to the telephone which restricts its callers.

The second consideration is a requirement that no single switch be totally out of service for more than two hours in a forty year period, including any outage time needed for system upgrades and other maintenance. For instance, in 1999 Nortel Networks switches were out of service for an average of 18 seconds per switch per year including failure and maintenance time. A key reason for this requirement is that one switch provides service to a significant geographical area. Failure of just one telephone leaves the option in an emergency of finding another working telephone nearby, but failure of the switch removes service from all the telephones in the area. One implication of this requirement is that the system must perform constant self diagnostics to test the performance of all hardware and software systems. Duplicates of critical subsystems must exist, with diagnostics which are capable of determining which subsystem is defective and moving all operations on to the other subsystem with no loss of service or interruption to existing connections. Any changes to individual user services such as location changes or additional features, and any upgrades to system hardware or software, must occur without service interruption.

The third consideration is that dial tone must be provided within a second of any request for service, and the connection established within a few seconds of dialing, even if hundreds of other new calls are in progress and thousands of connections already exist. Furthermore, the time available for information to pass between two points is very limited. The delay in transmitting a sound element within a conversation must be less than a few milliseconds, and the delay for real time video less than a few tens of milliseconds, even if many other functions are using the same physical connection. The quality of the received voice or image must be high, with minimal noise or distortion.

At any given instant in time, multiple behaviours could be indicated, including establishing a connection, breaking down a connection and releasing its resources for other calls, collecting billing information, performing a diagnostic task, recording the current volume of user requests for service, or making a change to the system. If processor or other resource limitations make it impossible to perform all the behaviours immediately, an interaction between internally specified objectives will determine the relative priority of different behaviours. For example, regular diagnostics may be delayed when user service demands are high, but if the delay becomes prolonged, the risk of serious undetected failures becomes significant and user services will be delayed if necessary to allow diagnostics to proceed. The combination of over 100 thousand users, thousands of interacting features, and tight reliability and real time constraints make central office switches some of the most complex electronic systems in existence.

The human brain performs an operationally complex combination of tasks. It must learn to select an appropriate response to a vast range of input conditions from within its wide range of available behaviours. These available behaviours include body movements (including changes to facial expression); verbal behaviours; and attention behaviours. Some attention behaviours are body movements (e.g. to look at different objects), others are internal actions to retain information active (e.g. the numbers when dialing a telephone call) or to activate information not currently active (e.g. mental images). There is often limited time within which appropriate behaviour must be selected.

For example, suppose that two people, Rick and Elaine, encounter each other, and consider the operational role of Rick's brain. A simple categorization problem would be to identify Elaine and speak her name. Even this problem is not as simple as it might appear, because variations in Elaine's appearance and the lighting conditions may make the raw visual input quite different from previous occasions. Furthermore, the raw visual input to Rick's brain is constantly changing as he and Elaine move, as Rick directs his attention to different parts of Elaine's face and body, and to other objects in the immediate environment. The higher level activated information in Rick's brain may change as he considers mental images of various memories. Elaine's appearance (e.g. expression) may be changing in response to Rick. To make things even more complex, the real behavioural problem is not a simple categorization. For example, depending on the location of the encounter, the appropriate response might be "Hi Elaine, what are you doing here?" Depending on her detailed appearance it might be "Hi Elaine, are you feeling okay?". This latter response might not be appropriate depending on the identities of other people present, but if the appearance indicated that Elaine was very unwell would be appropriate independent of who else was there. In addition, the appropriate response might include a hug, a kiss, or a handshake. The appropriate response also depends upon Rick's internal objectives: enhancing his business or personal relationship with Elaine; enhancing his relationships with other people present; and/or completing the task he was currently performing without being drawn into a conversation. The relative weight of these internal objectives might be affected by changes to Elaine's expression and by the responses of co-workers who happened to be present. Internal behaviours might include searching through recent memories for anything relevant to Elaine. Finally, the appropriate response must be delivered within a couple of seconds, and followed up by a sequence of appropriate and consistent other behaviours.

The human brain thus meets the definition of a operationally complex system: the number of possible behaviours is large; there are a number of potentially conflicting objectives; considerable interaction is required within a large body of information derived from sensory inputs in order to determine appropriate behaviour at each point in time; and the time available in which to determine behaviour is short.

Organization of System Resources

An operational system does not calculate mathematical functions in any useful sense. Rather, it detects conditions within the information space available to it and associates different combinations of those conditions with different behaviours. The information space available to the system is made up of inputs which provide information about the state of the equipment being controlled, the state of the environment in which the equipment operates, and the internal state of the system itself.

In principle, an operational system could be implemented as an immense look-up table, with each operationally relevant state of the total information space listed separately with its corresponding behaviour. If sequences of such states sometimes determined behaviour, all such sequences would also have to be listed in the table. Such an implementation would require impractical levels of information recording and processing resources. In practice therefore, an operational system must detect a much smaller population of conditions within different subsets of its information space, and associate many different combinations of this limited set of conditions with different behaviours. Identifying an appropriate but limited set of such conditions is a primary design problem for such systems. However, even such a comparatively limited set will in general be extremely large for an operationally complex system.

In order to conserve resources, it will therefore also be necessary to collect similar conditions into groups. The conditions in one group can be detected by a set of system resources optimized for detecting conditions of the group type. In this context, two conditions are similar if a significant proportion of the information defining them is the same. It will also be necessary to detect complex conditions by first detecting relatively simple conditions which occur in many complex conditions, then to detect the more complex conditions using the detections of the simpler conditions. A set of system resources optimized for detecting a group of similar conditions is called a module. Because similar conditions may be relevant to many different behaviours or features, modules will not correspond with such behaviours or features.

Note also that this definition of module is not the same as the concept of identical or almost identical units convenient for the construction of a system. There may be some general similarities between some modules for construction related reasons as discussed below, provided that such similarities are consistent with the primary role of modules. However, the primary role of modules is making the best use of resources in the performance of system operations, and there will be significant heterogeneity amongst even similar modules in order to achieve this primary purpose.

A system must also have a portfolio of primitive actions which it can perform on itself and its environment. For example, the brain has a portfolio of individual muscle movements. However, resource limitations will also make it impractical to associate conditions with individual actions. It will be necessary to define behaviours as frequently used sequences and combinations of actions, and types of behaviour as groups of similar sequences. A component contains a set of system resources optimized for driving a group of similar behaviours, and will be invoked by the detection of appropriate combinations of conditions.

An interaction between two modules occurs when a condition detected by one module is incorporated into a condition detected by another module. One condition detected by one module may be incorporated into different conditions detected by many other modules, either directly or via condition detections by intermediate modules. An interaction between two components occurs when both are invoked by currently detected conditions, and it is necessary to determine whether both can be invoked at the same time and if not, which of the two is most strongly invoked by currently detected conditions. All these interactions can also be viewed as information exchanges.

The User Manual and the System Architecture

The user manual for a system describes how the features of the system operate and interact from the point of view of an external observer. A feature is a group of similar behaviours invoked by similar circumstances, but in this context "similarity" is defined for the convenience of the external observer.

The system architecture describes how the operations of the system are separated into modules and components, the interactions between these modules and components, and how these modules, components and interactions result in the behaviour of the system. The separation of the system into modules and components is strongly influenced by resource constraints as discussed in the previous section. System modules and components will therefore not correspond with user manual type features. Rather, a module or component will contribute to the performance of many features and a feature will be dependent upon many modules and components. Hence the relationship between system architecture and user manual will be very complex. How the system operates is most easily understood from the user manual, but how features result from system operations can only be understood from the system architecture.

The implications for the brain are that the system architecture at the physiological level may have a very complex relationship with descriptions of cognitive phenomena as measured by psychology.

Practical Considerations which Constrain Architecture

Practical considerations which must be taken into account to some degree for any system become critical influences on the architecture of operationally complex systems. The primary such consideration is limitation to the available information recording and processing resources. As discussed earlier, one effect of this limitation is organization of resources into components and modules. However, there are some additional considerations which also have an effect on architecture.

One such consideration is the need to maintain adequate meaning for all information moving within the system. A second consideration which has important implications for

information meaning is the need to modify some features without excessive undesirable side effects on other features. A third, related consideration is the need to diagnose and repair the system without too much difficulty. A fourth is the need to construct many copies of the system from some kind of design specification by a process which is not too complex and error prone, and which does not require a specification containing an excessive volume of information.

Information moving within the system must have an adequate level of meaning. Firstly, temporal relationships between information being processed by different parts of the system must be maintained: the results obtained from the processing of a group of system inputs which arrived at the same time must not be confused with the results obtained from processing inputs which arrived at a later time. Secondly, because the results of processing by one module may be communicated to many other modules, and each module may use the same result for a different purpose, any changes to modules must be such that an adequate meaning for its results is preserved for all modules receiving those results.

Module changes may be required to add and modify features, or to recover from physical damage. The need to perform such changes efficiently means that it must be possible to identify a set of module changes which can implement such modifications or recoveries with minimal undesirable side effects on other features. The identification process can only use information which is readily available to the agent performing the change. This agent may be an external designer, but when changes are learned the agent is the system itself.

Finally, the need to construct the system by a process which can only use a limited volume of design information means that systems tend to be made up of a small set of standard building block types. The system then contains many building blocks of each type. Each block within a type is fairly similar to the standard for the type. The design of a building block type can then be described just once, and individual building blocks only require description of their differences from the standard. Since the system must consist of modules, a compromise is also required between making modules as similar as possible to minimize design information and customizing each module to make the most effective use of system resources for its assigned system role.

Although the brain is not designed under external intellectual control, analogous constraints to many of the above exist. A brain able to perform a higher number of features with the same physical information handling resources will have significant natural selection advantages. The brain must not confuse information derived from different objects perceived at different times. Information moving within the brain from any one source to different locations must have an adequate level of meaning to all recipients. The brain must be able to recover to some degree from physical damage. It must be possible to add features and modify existing features without excessive side effects on other features. The genetic information space available to specify the construction of the brain is not unlimited, and the construction process must be such that it can be specified with limited genetic information and operate either without errors or with recovery from errors.

However, although these constraints are analogous with some of those on electronic systems, a critical difference is that the behaviour of an electronic system is specified under external intellectual control, but the brain must learn a significant proportion of its behaviour

heuristically. The result of this difference is that the architectural form into which the brain is constrained is qualitatively different from that into which electronic systems are constrained.

Impact of the Practical Considerations on System Architecture

The various considerations place constraints on the possible form of a system architecture which become more and more severe as operational complexity increases relative to the resources available. Some of the constraints are the same for any operationally complex system. Others differ depending on whether system features are defined under external intellectual control or heuristically through experience (i.e. learned).

Hierarchy of Modules

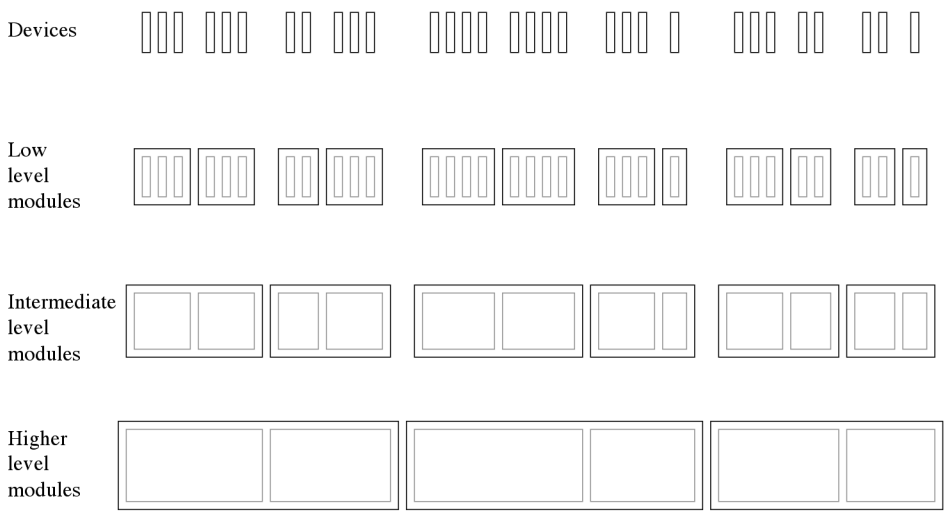

Figure 3.1 Organization of system operations into a modular hierarchy. The organization of the hierarchy is on the basis of condition information similarity with the objective of achieving information recording and processing resource economies. Devices are the most detailed “modules” and detect groups of very similar conditions. Low level modules are made up of groups of devices and detect all the conditions detected by their constituent devices. There is a strong but somewhat lower degree of similarity within this larger group of conditions and a significant degree of resource sharing in the detection of different conditions is possible. An intermediate level module is made up of a group of low level modules. The degree of condition similarity across the set detected by all the constituent low level modules is lower, and a lower but still important degree of resource sharing is possible. Higher level modules are made up of groups of intermediate level modules with a yet lower but still significant degree of condition similarity and consequent resource sharing.

As discussed earlier, limitations to resources result in similar conditions being organized into groups which can be detected by a module containing resources optimized for the group type. To achieve further resource economies, intermediate level modules made up of groups of more detailed modules are defined. The set of conditions detected across such a group are more different than the set detected by one detailed module, but are sufficiently similar that some of the detection process can be performed by optimized resources shared across the group. This resource sharing defines the intermediate module. Yet higher level modules can be defined by resource sharing across a group of intermediate modules and so on. The resultant modular hierarchy is illustrated in figure 3.1.

Hierarchy of Components

The level of connectivity required to connect relevant conditions to every primitive action driver would be excessive. Frequently used combinations and sequences of actions are therefore defined as behaviours with one corresponding control component. This component receives inputs from various modules indicating the detection of conditions, and generates outputs which drive the appropriate sequence of actions. Groups of similar behaviours invoked under similar conditions are collected into higher level components where most of the condition detecting inputs can be directed to the higher level component.

Information Exchange within a Modular Hierarchy

Condition similarity in an information sense does not guarantee similarity of behavioural meaning. The conditions detected by one module will therefore be relevant to many behavioural features, and the operation of any one feature will depend upon conditions detected by many modules. Using modular hierarchies to reduce resource requirements therefore introduces a problem. If the operation of the system must be changed in some desirable manner without excessive proliferation of the conditions which must be detected, in general some previously defined groups of conditions must be modified. Such modifications will tend to introduce undesirable side effects into other operations which employ the original unmodified groups. It will therefore be necessary to find a compromise between use of resources and ease of modification. To make operational change practicable, modules must be defined in such a way that overall information exchange between modules is minimized as far as possible. In other words, conditions detected by one module must be used as little as possible by other modules, as far as is consistent with the need to economize on resources. The modular hierarchy is therefore a compromise between resource economy and ease of modification. Experience with the design of operationally complex systems indicates that such modular hierarchies are indeed essential to make operational changes possible [Kamel 1987].

Information Exchange within a Component Hierarchy

The primary role of information exchange within a component hierarchy is to coordinate different behaviours and resolve conflicts between them. Because components correspond with behaviours, the management of information meanings is simpler than within a modular hierarchy. An input to a component can only mean that the corresponding behaviour is encouraged or discouraged, and an output that the corresponding behaviour is encouraged and/or any other behaviour is discouraged.

Examples from Design of Electronic Systems

As an illustration of the compromises needed in defining the various hierarchies, consider the flight simulator example. Different modules could be organized to correspond approximately with different tasks performed by the pilot, such as takeoff, level flight, turning etc. With this approach, most of the information required at any point in time is within one module and the processing time required is minimized. It is therefore easier to achieve the necessary real time response. However, information on different parts of the aircraft must be maintained in each module and any changes to aircraft design will require changes to all modules. If modules are organized to correspond more closely with different parts of the aircraft, functional changes to the simulator to reflect aircraft technological advances are easier. Simulator architectures have therefore shifted from task modules to aircraft component modules as available processing speeds have increased and aircraft technological change has accelerated [Bass et al 1998].

In the case of the much more complex central office switch, call processing, diagnostics, billing and traffic measurement are different major customer features. However, the operational complexity of these tasks and the interactions between them is so great that if high level modules corresponded with these features, the interactions between modules and the demands for computing resources would be excessive. For example, billing requires information on the features invoked in the course of call processing, and call processing requires immediate information on problems identified by diagnostics.

As a result, there is very little correspondence on any level between modules and identifiable customer features. At the highest level, there are no modules corresponding with the major feature domains like call processing, diagnostics or billing. There are no modules on any level which correspond exactly with features like conferencing or call processing. Rather, at a detailed level the management of a connection between two users, from originating a connection through answering the call to disconnection of both users, is managed by repeated invocation of software modules performing functions with names like queue handler, task driver, event router, function processor, or arbitrator, and differences in the features invoked are only reflected in the information provided to these modules.

Modules in the Brain

Limitations to information handling resources are certainly present in biology, and will tend to result in a modular hierarchy in the brain. However, modules are not simply arbitrary parts from which the system is constructed, but must have specific properties tailored so that the brain achieves resource economies. The modular hierarchy must divide up the population of operations performed by the brain into major modules in such a way that similar operations are grouped together but the information exchange between them is minimized as far as possible. Major modules must be divided into submodules on the same basis, and so on down to the most detailed system operations. One general result of this process is that a module on any level will tend to have much more information exchange internally (i.e. between its submodules) than with other modules. Because, in the brain, information flows along physical connection pathways, this leads to the expectation of a hierarchy of anatomical modules distinguished by much more internal connectivity than external.

To illustrate the resource/modifiability problem for the brain, consider how visual input could be used to control a number of types of behaviour with respect to a river. These types of behaviour could include white water canoeing in, swimming in, fording, fishing in, drinking from, and verbally describing the river. For each of these types there are different visual characteristics which identify the optimum locations and ways to perform different behaviours within the type. These visual characteristics are all observations of the river which correlate partially with operationally significant conditions like depth of water, rate of flow, presence of visible or hidden rocks, evenness of the river bed, condition of the river banks etc. Such characteristics might include areas of white water, areas of smooth water, visible rocks, or whirlpools. The shapes and internal variations of different instances of these characteristics could be relevant to determining behaviour and therefore be part of the characteristic.

However, the shape and appearance of an area of relatively smooth water most appropriate for indicating a place to stop the canoe and plan the next part of the route through the rapids may not be exactly the same as the shape and appearance most appropriate for indicating the probable location of a fish, and the shape and appearance of a smooth area in which drinking water would contain the least sediment may be different again. Furthermore, even within one behaviour type, the shape and appearance of the relatively smooth area most appropriate for stopping the canoe may be different from the relatively smooth area indicating the best point of entry into a rapid. These high level characteristics might be constructed from more detailed characteristics like ripples, waves, relative colorations and shades, boundaries of different types etc. However, again the more detailed characteristics best for constructing the most appropriate higher level characteristics may not be the same for all behaviours.

One solution would be to define characteristics independently on all levels for each behaviour for which optimum characteristics were different in any way. This solution will in general be very resource intensive for an operationally complex system, and in most cases a compromise must be found. For example, a characteristic "patch of smooth water" might be used for multiple behaviours but for each behaviour there could be additional characteristics which could distinguish between operationally different patches.

Consider now the problem of learning an additional behaviour, such as learning to find a place to ford a river after white water canoeing, fishing, and drinking have been learned. Suppose that this learning must occur with minimum consumption of resources but also with minimum interference with already learned behaviours. Minimizing the use of resources would be achieved by minimizing the number of additional characteristics, in other words by modifying existing characteristics for the new behaviour. Minimized interference would be achieved by creating a new characteristic whenever there was a difference between the optimum characteristic for the new behaviour and any existing characteristic. Some compromise must be found between these two extremes, in which some already defined characteristics are slightly modified in a way which does not severely affect existing behaviours, and only a small number of additional characteristics are defined when the side effects of modification would be extensive and/or hard to correct. The effect of this process is that individual characteristics will not correspond with conditions which are behaviourally useful for just one behaviour. Rather, they are conditions which in different combinations are effective for managing multiple behaviours.

Cognitive science has made many attempts to define modules at relatively high psychological levels, with the hope of being able to identify such modules with physiological structures. "Module" in this context has generally been understood (following Fodor) as a unit which can only access a limited subset of the information available to the system, which performs a set of tasks following algorithms not visible to other modules, and which has a dedicated neural structure. Such a definition has some similarities with the minimized information exchange requirement. Cognitive modules of many different types have been proposed, including peripheral modules, domain specific modules, and conceptual modules. Proposed peripheral modules include early vision, face recognition, and language. Such modules take information from a particular domain and only perform a specific range of functions. Proposed domain specific modules include driving a car or flying an airplane [Hirschfeld and Gelman 1994]. In such modules highly specific knowledge and skill is well developed in a particular domain but does not translate easily into other domains. Conceptual modules are innate modules containing intuitive knowledge of broad domains, such as folk-psychology and folk-physics, with limited information flow to other parts of the brain [Leslie 1994]. However, various reasons including the difficulty of associating any such modules with plausible physiological structures has led to growing skepticism of the existence of such modules in the brain.

Because of the high operational complexity of the functions performed by the brain, the modules which would be expected to develop on the basis of practical considerations would not correspond with cognitive functions such as those described in the previous paragraph. However, the modules would be expected to correspond with physiological structures.

Module and Component Physical Similarities

Because systems must be constructed using a volume of design information which is as limited as possible, modules tend to resemble each other. Such a resemblance means that the same design information can be used with minor overlays to construct many different components and modules. For example, in electronic systems transistors are formed into integrated circuits and integrated circuits assembled on printed circuit boards, but the numbers of different types of transistors, integrated circuits, and printed circuit assemblies are kept as small as possible. Software is generally written in a high level language which can be translated (compiled and assembled) to assembly and machine code, but the portfolios of machine code, assembly code and high level language instructions are limited as much as possible.

A brain is constructed by a process which uses genetic information, and limiting the volume and complexity of this information could be expected to reduce the risk of copying errors and errors in the construction process. Brains can therefore be expected to be constructed from basic sets of modules and components on many levels of detail, which can be combined in various ways to achieve all required system functions. Note, however, that although the physical structure may be generally similar, every module and component will have a different operational role and will be physically different at a detailed level.

Condition Synchronicity

An operational system receives a constant sequence of input information. Frequently, a behaviourally relevant condition occurs within information that becomes available to the system at the same point in time. Other behaviourally relevant conditions may be made up of groups of simpler information conditions which are detected and became available to the system at different times separated by specific time intervals. Whether a condition is made up of direct system inputs or of subconditions made up directly or indirectly of system inputs, all of the information making up the condition must be derived from system inputs at the same point in time or at times with the appropriate temporal relationships.

For example, in the aircraft simulator example, a change to the orientation of the aircraft initiated by the pilot at one point in time could result in changes to many different aircraft components and to the cockpit displays and platform orientation. The module corresponding with each component could take a different time to determine its new status, and it is critical that all status changes occur synchronously.

A module will take a certain amount of time after the arrival of its inputs to detect the presence of any of its conditions and generate outputs indicating those detections. Outputs will take a certain amount of time to travel from the module generating them to the modules utilizing them. In the absence of correction, these delay times will tend to result in condition detections within information which does not have the appropriate temporal relationships.

The system architecture must therefore have the capability to ensure that conditions are detected within information with the appropriate temporal relationships. There are two

possible approaches to this synchronicity management: global and local. The two approaches are illustrated in figure 3.2.

In the global approach, all the relevant inputs at the same point in time are recorded in a reference memory. Modules detect their conditions using the input information in this reference memory, and record the presence or absence of conditions they detect in this input set in the memory. Such recorded conditions can then be used by other modules. Modules must detect conditions in a sequence which ensures that the presence of a condition is not tested until all its component conditions have been tested.

In the local approach, modules are arranged in layers, and one layer can only draw condition defining information from the immediately preceding layer, or from system inputs in the case of the first layer. If all the modules in one layer take the same amount of time to detect their conditions and transmission time to any module in the next layer is the same, then all conditions will be detected within a set of system inputs at the same point in time.

The global approach requires additional resources and complexity to support explicit management of synchronicity. The local approach avoids the need for this explicit synchronicity management process, but in general will be less accurate.

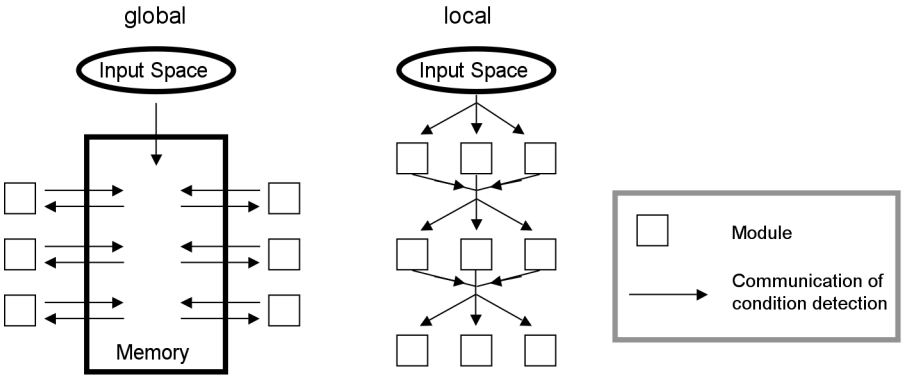

Figure 3.2. The global and local approaches to ensuring appropriate temporal relationships within the information making up conditions. In the global approach, an input state containing the input information at one point in time is recorded in a reference memory. Conditions are detected using information from that memory. The presence or absence of conditions in the input state is also recorded in the memory for use in the detection of more complex conditions. All conditions are therefore detected in a synchronous input state. In the local approach, simple conditions are detected within the information from the input state in a first layer of modules. Outputs from these modules indicating detection of their conditions are provided to a second layer which detects more complex conditions and so on. The layering and predominant connectivity only from one layer to the next layer means that all the conditions detected in one layer are within a synchronous input state provided module processing times and communication times are consistent across the layer. The local approach is therefore less complex but more prone to errors.

In commercial operational systems, global management of synchronicity is invariably used to ensure that the order in which module operations occur minimizes the probability of errors. For example, in actual flight the orientation and motion of an aircraft affects the

operation of a weapon system and the operation of a weapon system affects the orientation and motion of the aircraft. The timing of the arrival of updated information between modules influencing these different components therefore affects the overall accuracy of the simulation. In a central office switch, call processing uses connectivity resources from a common pool, and both call processing and diagnostics can add and remove resources from the pool. If the knowledge of what remains in the pool is not updated in an appropriate sequence, a call in progress may plan to include a resource which is no longer in the pool, and have to recalculate its end-to-end connectivity path when it discovers that the resource is not actually available.

Modification, Diagnosis and Repair Processes

Although modules do not correspond with identifiable features, the modular hierarchy nevertheless makes it possible to add or modify features while limiting side effects and to diagnose and fix problems. To understand how the module hierarchy provides these capabilities in an electronic system, the first step is to recognize that a side effect of the way modules are defined is that the module hierarchy can operate as a hierarchy of descriptions for operational processes. If a module participates several times in the course of some process, a description which names the module rather than providing a full description of every internal operation will be simpler. Because modules are defined so that most interactions (i.e. information exchanges) are within modules and only a small proportion are with other modules, such a description can just include reference to the small external proportion. Such descriptions are more likely to be within the information handling capacity of the human brain. Although a particular feature will not in general be totally within one module, the criterion for module definition will tend to limit any one feature primarily to a relatively small set of modules.

An operational process can therefore be described at high level by making explicit reference to the names of each module which participates in the process and to the relatively small level of interaction between the modules. A new feature or a problem can be understood at that level, and the necessary changes or corrections to each module at high level understood. Consideration can then be limited to each participating high level module in turn by considering only the parts of the operational process in which each such module is active. These parts of the process are considered at the level of the submodules of each such module and the changes or corrections to each participating submodule which will create the desired module changes determined. Each submodule can then be considered separately and so on down to the level of software instructions and transistors at which changes can be implemented. In practice this change process also involves shifts back to higher levels, to determine whether a proposed change at detailed levels actually achieves the desired change at higher level, and whether it has undesirable side effects when different operational processes involving the same modules are considered at higher level. The modular hierarchy thus makes it possible to determine an appropriate set of detailed implementable changes to achieve a required high level feature or corrective change by a process which is always within the information capacity limitations of a human being.

If features must be modified heuristically, the existence of the modular hierarchy as defined is still an advantage. If circumstances are experienced which indicate the need for a change in behaviour, the set of modules most often active in those circumstances can be identified as the most appropriate targets for change. The overall minimization of information exchange will tend to limit the size of this set.

The existence of a modular hierarchy can also make recovery from damage more effective. For example, suppose that one module has been damaged. An external designer could of course identify the damaged module. A system which must use only its own resources to recover from damage is less likely to have such a direct capability. Brains need (and exhibit) the capability to recover from some of the deficits created by a stroke. It is essential that the actions selected heuristically to achieve recovery from damage have a high probability of achieving the desired change and low probability of undesirable side effects. This high probability must be achieved using only whatever information is actually available to the system. Such information could include identification that some modules were no longer receiving inputs (because these inputs came from the now damaged module). Resources for a new module to replace those inputs could be assigned, with outputs to the modules no longer receiving inputs. If the system has the capability to identify which modules have often been active in the past at the same time (see below for reasons why such a capability will in general be required), then the new module could be given provisional inputs from modules which were frequently active in the past at the same time as the modules no longer receiving inputs. These input biases will improve the probability that as the new module learns, it will replace the operations of the damaged module. Modularization and minimization of information exchange thus improves the effectiveness of damage recovery.

Context for Information Exchange within a Modular Hierarchy

As discussed earlier, the detection of a condition by one module is sometimes communicated to other modules. Such information exchanges also have an operational meaning to modules receiving them. To a recipient module, a specific condition detection means that the currently appropriate system behaviour is within a specific subset of the set of all behaviours influenced by the module.

A specific condition detection may have a different context (i.e. a different operational meaning) in each of the modules which receive it. For example, the detection of a specific condition by one module in a flight simulator might be used to indicate the need for changes to cockpit monitor images, cockpit displays, and platform motion.

Changes to a module can include both changes to its input population and changes to the conditions it detects within that input population. Such changes will affect outputs, and changes for one operational purpose may affect any module which makes use of outputs from the changed modules, directly or via intermediate modules. There is no guarantee that these secondary changes are desirable. This issue is conceptually illustrated in figure 3.3.

For example, a change to the definition of a condition detected by one module in a flight simulator made to reflect a change to the instrument displaying such conditions might have undesirable side effects on the cockpit platform motion which depended upon the original condition definition. The operational change problem can therefore be viewed as a problem of maintaining meaningful contexts for all information exchange. The seriousness of the context maintenance problem is illustrated by the experience that achieving adequate contexts for shared information is a major (and sometimes insurmountable) problem in integrating independently developed software systems, even for information technology applications [Garlan et al].

There are two different confidence levels which could be supported for operational meanings. One is unambiguous, in which a condition detection limits appropriate behaviour to the specific subset with 100% confidence. In this case a condition detection can be interpreted as a system command or instruction.

The alternative is meaningful but partially ambiguous meanings. In this case a condition detection indicates that the appropriate behaviour is probably within a specific subset. Such information exchanges can only be interpreted as operational recommendations, with the implication that a higher level of condition detection resources will be required to generate high integrity behaviours.

Although a system supporting only ambiguous contexts will require more condition detecting resources for a given set of system features, it has the advantage that heuristic changes to features are feasible in such a system and generally impractical in a system using unambiguous exchanges. To understand the practical issues around heuristic change, consider again the set of intercommunicating modules in figure 3.3.

In that figure, on average each of the eleven modules receives information from 2 modules and provides information to two modules.

Suppose that information exchanges are unambiguous. In this case the detection of a condition can be interpreted as an operational command. If a change is made to the conditions detected by the black module M in order to implement a feature change, that change will potentially affect the conditions detected by the three gray modules in figure 3.3 which receive information directly from M. Because these modules also influence features not requiring change, and the result of a change to a condition will be operational commands under the changed condition and not the original, changes may be required to the gray modules to correct for undesirable side effects of the change to M. However, any changes to the gray modules may affect the four striped modules which receive information from the gray modules and so on.

In a system in which features are defined under external intellectual control, the effects of changes to one module must be traced through all affected modules and side effects corrected explicitly. This is a difficult problem to solve and accounts for the difficulty of modification of such systems. However, if features are defined heuristically, the system must experiment with behaviours, receive consequence feedback, and adjust accordingly.

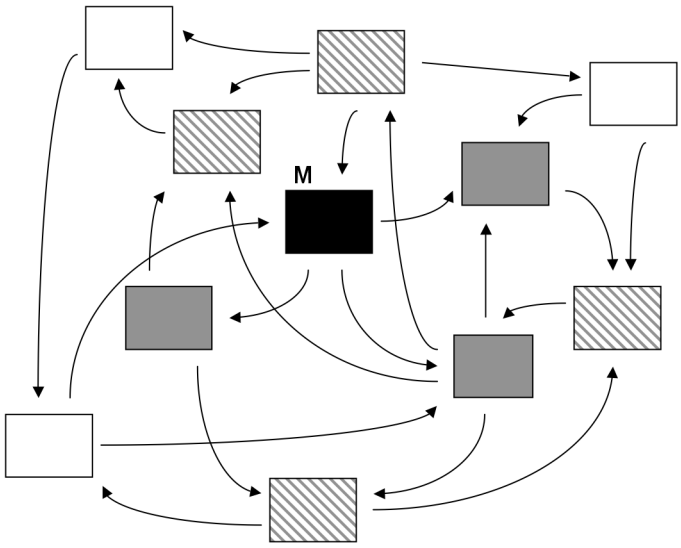

Figure 3.3 Spread of impact of information changes from one changed module. The arrows indicate information communications. In the illustration the black shaded module M is changed in some way to achieve some feature change. Module M provides condition detections to the three gray shaded modules. The features influenced by these modules may therefore be affected by the change to M. The gray shades modules provide condition detections to the striped modules which may therefore be affected by the change to M. The white modules receive inputs from the striped modules and may also be affected. There could be an exponentially increasing wave of side effects developing from the change to M.

An experimental change to a feature will in general require an experimental change to conditions. Any experimental change to a condition will result in a command changes not only affecting the targeted feature but also any other features also influenced by the condition. The change will therefore introduce uncontrolled side effects on these features. However, consequence feedback is received only in response to the targeted feature and will not provide useful information on the other features until such features are invoked.

Thus any experimental change to one feature will result in multiple undesirable side effects on other features. Each of these side effects will eventually require an experimental change followed by consequence feedback to correct. Because all conditions are unambiguous commands, this process is very unlikely to converge on a high integrity set of features. In practice, for a large number of features dependent on the same group of shared resource modules, it is more likely that system behaviour will steadily diverge from desirable behaviour. This is similar to the catastrophic forgetting problem encountered in artificial neural networks, in which later learning completely overwrites and destroys earlier learning [French].

However, if information exchange contexts are partially ambiguous, information exchanges are operationally only recommendations. Any accepted behaviour for one feature will in practice be supported by a large number of recommending condition detections. "Small" changes to a few of these conditions to support changes to other features will not necessarily affect the performance of the feature, and any effects on performance could

potentially be corrected the next time the feature is invoked. Thus provided condition changes are "small", the system could converge on a set of high integrity features. Clearly the definition of "small" is critical.

Local synchronicity management is unlikely to be adequate if an unambiguous level of information meaning must be sustained. However, if information exchanges are partially ambiguous, some level of synchronicity errors resulting from local synchronicity management can be tolerated, and the resource cost and complexity of global management can be avoided.

The von Neuman Architecture

Some of the reasons for the ubiquitous appearance of the memory, processing von Neumann architecture in commercial electronic systems are now apparent. Unambiguous information exchanges are used because much less information handling resources are used. The command structure of software (e.g. if: [x = a] do: […]) reflects use of unambiguous contexts. The use of unambiguous information mandates the memory, processing separation in order to maintain synchronicity. Finally, learning in a von Neumann system performing an operationally complex task has never been demonstrated in practice.

The Recommendation Architecture

If information which is operationally ambiguous is exchanged within a module hierarchy, module outputs indicating the presence of information conditions can only be interpreted as behavioural recommendations. Multiple recommendations will be generated in response to an overall input condition in most cases. A subsystem separate from the modular hierarchy is therefore required which can select one behaviour. To make such selections this subsystem must have access to information in addition to the overall input state, such as the consequences of behaviours under similar conditions in the past.

The separation between a modular hierarchy and a component hierarchy which interprets module outputs as alternative behavioural recommendations and selects one behaviour is one of the primary architectural bounds of the recommendation architecture. This separation is illustrated in figure 3.4. The hierarchy is called *clustering* because it clusters conditions into modules detecting portfolios of similar conditions, and the selection subsystem is called *competition* because it manages the competition between alternative behavioural options.

Information on the most appropriate associations between information conditions detected within clustering and behaviours must either be derived from the experience of other systems (i.e. via design or genetic experience, or the experience of an external instructor) or be derived from the experience of the system itself (i.e. from feedback of the consequences of behaviours under various conditions in the past).

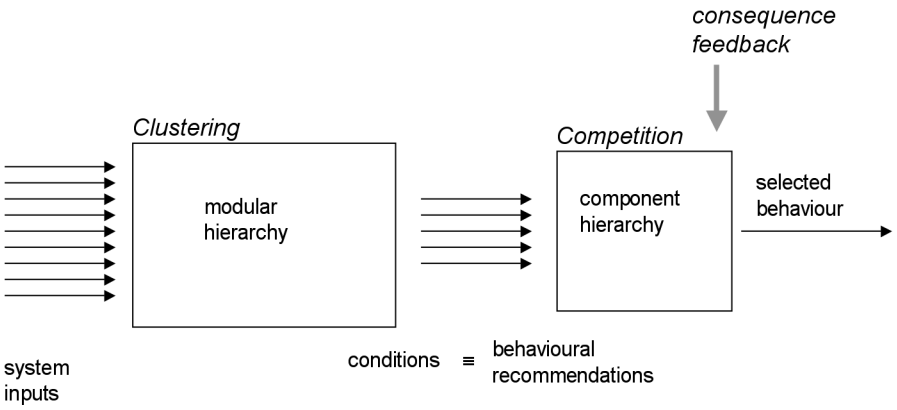

Figure 3.4 Recommendation architecture primary separation between clustering and competition. The clustering subsystem defines and detects behaviourally ambiguous conditions in system inputs. Some conditions are used to determine when and where other conditions will be added or modified within clustering. Other conditions are communicated to the competition subsystem. The competition subsystem uses consequence feedback (sometimes associated with imitation of an external teacher) to interpret conditions detected by clustering as behavioural recommendations and to select the most appropriate behaviour under current circumstances. Conditions detected by clustering cannot be directly changed by competition or by the consequence information available to competition.

In a very simple version of the recommendation architecture, all information conditions and behaviour selection algorithms could be specified a priori. In a more sophisticated version, conditions and algorithms could be almost entirely specified a priori but slight changes could be made to conditions by adding inputs which frequently occurred at the same time as most of the condition and deleting inputs which rarely occurred. Consequence feedback could tune the behaviour selection algorithms. Learning from experience would be present but very limited and the architectural constraints deriving from the need to learn without side effects would therefore be less severe. Coward [1990] has argued that this is the case for the reptile brain. A yet more sophisticated version would allow conditions to be defined heuristically and algorithms to be evolved by consequence feedback and external instruction, which as discussed below appears to correspond with the mammal brain. Different degrees of a priori guidance in the selection of starting points for conditions and algorithms could occur.

If the conditions detected by clustering modules are defined heuristically, the operational ambiguity of information makes it possible to limit the undesirable side effects of heuristic changes. In general terms this can be understood as an effect of multiple recommendation generation. A wide range of conditions will be detected in the typical overall input state. Overall input states corresponding with an already learned behaviour will in general contain a high proportion of conditions which have been associated with the behaviour by the competition subsystem. Slight changes to some conditions will therefore generally not change the behaviour selected. However, a change to a condition will also change any higher complexity condition which has incorporated that condition. The critical issue is therefore the definition of "slight" and "some", as discussed in the next section.

If consequence feedback were applied directly to clustering it would need to be applied to the currently active modules. Unfavourable consequence feedback could be interpreted as indicating that the wrong conditions were being detected by those modules, but would give no indication of how the conditions should be changed. Thus any change could not be guaranteed to improve the selection of the current behaviour in the future, and would have uncontrolled side effects on any other behaviours influenced by the changed modules. In an operationally complex system such changes would be very unlikely to converge on a high integrity set of behaviours. Consequence feedback therefore cannot be used directly to guide the definition of conditions. The primary information available to guide heuristic definition of conditions is therefore limited to frequent occurrence of a condition. This temporal information must be utilized to define clusters of input conditions on many levels of complexity. The clustering process is conceptually similar to the statistical clustering algorithms used in ANN unsupervised learning, as discussed below. External instruction could of course influence the definition of conditions by influencing the environment from which the system derived its inputs.

The use of consequence feedback constrains the form of the competition subsystem. Competition cannot depend on internal exchange of information with complex operational meanings. Components in competition must each be associated with just one behaviour, specific sequence of behaviours invoked as a whole, or general type of behaviour. Competition therefore corresponds with the component hierarchy discussed earlier. The output from such a component to other competition components can be interpreted as a recommendation in favour of the behaviour and against all other behaviours, and an output which goes outside competition can be interpreted as a command to perform the behaviour. Such components can interpret inputs from other clustering modules or competition components as either encouraging or discouraging their one behaviour. The relative weights of such inputs can be adjusted by consequence feedback in response to that one behaviour. As discussed below, this approach is similar to reinforcement learning in ANNs.

Heuristic Changes to Conditions

The output of a module within clustering indicates the detection of an information condition within the portfolio of such conditions programmed on the module. In an operationally complex system such an output may be an input to many other clustering modules. The operational meaning of such an input is the recommendation of a set of behaviours which will in general be different for each recipient. The major issue is therefore how to preserve the multiple meanings which the output of a module has acquired when the conditions programmed on the module which generated the output are changing. This is the issue of context management.

To illustrate the issue, consider a simple human example. Suppose that I have an understanding with my wife that when I leave a telephone message asking her to call me at the office, an emergency has occurred. Suppose also that this understanding is completely unambiguous, and I will never leave such a message in the absence of an emergency. In this situation, if I leave a message she will stop whatever she is doing and act as if there is an

emergency. If one day I just wanted to chat, leaving the message will result in an inappropriate response. In other words there is no scope for learning. However, if leaving a message was partially ambiguous, then the tone of my voice, the exact wording, my mood when I left that morning, and her knowledge of the probability of an emergency at that particular time could all contribute to selection of a different response. In other words, learning is possible. However, there are limits to the degree of change. I cannot leave a message in a foreign language and expect a call, and I cannot leave the usual message and expect to be called at the new number I have just been assigned. So the critical issues for context maintenance are how much the form of the message generated under the same conditions can be changed, and how much the conditions under which the same message is generated can be changed without destroying the usefulness of the message to recipients.

As operational complexity increases, the average degree of change which can be tolerated without unacceptable loss of meaning will decrease. For example, at the most detailed level of the modular hierarchy there must be devices which can select, record, and detect conditions which are combinations of their inputs. Suppose that one such device recorded its first combination (or condition) and indicated its presence by producing an output. Devices in many other modules might then select that output to be part of their own conditions. In general the source device has no "knowledge" of the operational meanings derived from its outputs by its targets. If that source device subsequently changed its programmed condition, the meaning assigned to the output by its recipients would be less valid. Degree of change to a device must therefore be "slight" and located on as "few" devices as possible. Determination of when and where change should occur is itself an operationally complex problem which must be managed by the system. In other words, some of the conditions detected in some modules must determine when change can occur to the conditions programmed on other modules.

Degree of Change

The minimum change situation would be if a device could only be programmed with one condition, and only produce an output in response to an exact repetition of that condition. Such a change algorithm would be extremely expensive in terms of device resources needed. A slightly greater degree of change would be if such a device could subsequently be adjusted to produce an output in response to any large subset of its original condition. A greater degree of change would be if a small proportion of inputs could be added to an existing condition, for example if the additional inputs occurred at the same time as a high proportion of the condition. A yet greater degree of change would be if the device could be programmed with multiple semi-independent conditions with inputs drawn from a population of possible inputs which tended to occur frequently at the same time. Such a device would not produce an output in response to the presence of a subset of one of its independent conditions, but might respond to the simultaneous presence of many such subsets. In all these change algorithms, the generation of an output in response to a specific condition results in a permanently increased tendency to produce an output in response to an exact repetion of the condition. Repeated exposure could be required to guarantee that an output will always occur, or a single exposure could be adequate. The key characteristic of the algorithms is that change only occurs in one direction, always resulting in expansion of the portfolio. This characteristic will

be referred to as “permanent condition recording” although multiple exposures may in some cases be needed as also discussed when physiological mechanisms are reviewed in chapter 8. To avoid portfolios becoming too large and therefore indiscriminate, as a portfolio expands it will become more difficult to add new conditions. A portfolio which became too broad would be supplemented by new portfolios in a similar input space as described in the section on Resource Management Processes in the next chapter.

There is nevertheless an information context issue even with permanent condition recording algorithms: two identical device outputs at different times may indicate slightly different conditions. Change algorithms allowing higher degrees of change may require additional mechanisms to maintain context. For example, there could be structure in a device output which provided some information about where in the device portfolio the current condition was located. In addition there could be management processes to determine the consistency by some additional criterion between conditions detected by different devices. A source of information for such a criterion could be the timing of past changes to devices. Suppose that whenever a relatively low complexity condition was recorded on a device, feedback connections were established to that device from devices which were currently detecting higher complexity conditions. Then suppose that subsequently if a low complexity device produced an output indicating detection of a condition, it would cease producing the output unless some of these feedback connections were active. If some low complexity devices ceased activity, some higher complexity devices would in general cease activity because of loss of their inputs. The population of active devices would therefore be reduced towards a set with higher consistency in past recording activity. Release of information outside the overall device population could be delayed until the period required for feedback had elapsed. Conditions recorded or changed on devices at all levels in that overall population would only become permanent if device activity continued beyond the feedback period [Coward 2000].

Artificial neural networks use change algorithms based on adjustment to relative input weights. Such algorithms mean that the device does not necessarily produce an output in response to conditions which previously produced an output. As operational complexity increases, the problem of maintaining adequate context within the clustering will become more and more severe. These types of algorithm are therefore unlikely to be adequate under such conditions. However, under the operationally simpler information contexts within the competition subsystem, such relative weight change algorithms will be ubiquitous.

The selection of the change algorithm is a compromise between resource economy and context management adequacy. If there were an input domain within which relatively simple conditions could be defined in early system experience, and the definition completed before these conditions were incorporated extensively in higher complexity conditions, then the change algorithms for the simple conditions could be more flexible, even to the point of using relative weight adjustments. However, no such changes could occur after the early learning period. For ongoing learning of a very complex set of operations subject to resource constraints the most plausible device algorithm is the multiple semi-independent condition recording model with mechanisms to enhance context maintenance. This device is described in detail in the next chapter. The same compromise between resources and context maintenance could also mean that if a condition was recorded but never repeated for a long

period of time, elimination of that condition and reuse of the resources would be relatively low risk, perhaps provided that the condition was not recorded at the time of a behaviour which could be viewed as highly important.

Location of Change

A further consideration for context management within clustering is the determination of when and where changes should occur. To give competition something to work with, even to make the decision to do nothing, clustering must generate some output in response to every overall input state. If output is inadequate, clustering must add conditions or modify conditions until output is adequate. The selection of where such changes should occur is itself a operationally complex problem which must therefore be managed by clustering. In other words, as described in the next chapter, some detected conditions must determine whether or not changes are appropriate in specific modules under current conditions. At the device level this implies that there will be change management inputs in addition to the inputs defining conditions. These change management inputs will excite or inhibit changes to conditions, but will not form part of any condition.

Organization of Change

As described in more detail in chapters 4 and 6, clustering organizes system input states into groups of similar conditions detected within those states. These groups are called portfolios, and conditions can be added to a portfolio but not deleted or changed. Portfolios are added and expanded to meet a number of objectives. Firstly, every input state must contain conditions within at least a minimum number of different portfolios. Secondly, the overlap between portfolios must be as small as possible.

In general terms, the process for definition of a set of portfolios is as follows. A series of input states to the portfolio set are monitored to identify groups of individual inputs which tend to be active at the same time but at different times for each group. Different groups of this type are defined as the initial inputs to different portfolios. Conditions which are subsets of a group are then defined in the corresponding portfolio whenever the input state contains a high level of activity in the input group. Additional inputs can be added to the group if required. However, as the number of different input states containing conditions within one portfolio increases, the ability of that portfolio to add conditions decreases. An a priori bias placed on the initial input group definitions can improve portfolio definition speed and effectiveness.

The set of portfolios defined in this way will divide up the input space into similarity groups which will tend to be different from each other but to occur with similar frequeny in input states. Individual portfolios will not correspond exactly with, for example, cognitive categories, but will permit discrimination between such categories. Thus different categories will tend to activate different subsets of the set of portfolios, and although the subsets may partially overlap, the subset for one category will be unique to that category. There are mechanisms for detecting if such subsets are not unique and adding portfolios in such circumstances.

As described in more detail in chapter 4, an example would be if a set of portfolios was exposed to a sequence of input states derived from different pieces of fruit. Portfolios might

be defined that happened to correspond roughly with similarity groups like "red and stalk", "green and round", or "smooth surface and round" etc. Although no one such portfolio would always be detected when one type of fruit was present and never any other fruit, the specific combination of portfolios detected could provide a high integrity identification of the type. Each portfolio can be given a set of weights corresponding with how strongly it indicates different fruit types, and the most strongly indicated fruit across the currently detected portfolio population determined.

Portfolios defined on different levels of condition complexity could provide discrimination between different cognitive feature, objects, groups of objects etc.

Similarities between Modular Hierarchies in Operational Systems and Description Hierarchies in Scientific Theories

There are some strong similarities between modular hierarchies in operationally complex systems and the hierarchies of description which form theories in the physical sciences as discussed in chapter 2. In both cases the units of description on one level are defined in such a way that their internal construction can largely be neglected in the description of phenomena or processes at that level, and human understanding is made possible by the capability to shift freely between levels.

The modules into which the brain will tend to be constrained by practical resource and other considerations can therefore be the basis for a scientific understanding of cognition in terms of processes between identifiable physiological structures.

References

Bass, L., Clements, P. and Kazman, R. (1998). Software Architecture in Practice. Addison-Wesley.

Chastek, G. and Brownsword, L. (1996). A Case Study in Structural Modeling. Carnegie Mellon University Technical Report CMU/SEI-96-TR-035. www.sei.cmu.edu/pub/documents/96.reports/pdf/tr035.96.pdf

Coward, L. A. (1990). Pattern Thinking, New York: Praeger.

Fodor, J. A. (1983). *Modularity of Mind*. Cambridge, MA: MIT Press.

French, R.M. (1999). Catastrophic Forgetting in Connectionist Networks. *Trends in Cognitive Science* 3(4), 128-135.

Garlan, D., Allen, R. and Ockerbloom, J. (1995). Architectural Mismatch or Why it's hard to build systems out of existing parts. *IEEE Computer Society 17th International Conference on Software Engineering*. New York: ACM.

Hirschfeld, L. and Gelman, S. (1994). *Mapping the Mind: Domain Specificity in Cognition and Culture*. Cambridge University Press; Karmiloff-Smith, A. (1992). Beyond Modularity. MIT Press.

Kamel, R. (1987). Effect of Modularity on System Evolution. *IEEE Software*, January 1987, 48 – 54.

Leslie, A. (1994). ToMM, ToBY and Agency: Core architecture and domain specificity. In *Mapping the Mind*, L. Hirschfeld and S. Gelman, eds. Cambridge University Press.

Nortel Networks (2000). DMS-100/DMS-500 Systems Feature Planning Guide. http://a2032.g.akamai.net/7/2032/5107/20030925230957/www.nortelnetworks.com/products/library/collateral/50004.11-02-00.pdf