Coward, L.A. (2000). A Functional Architecture Approach to Neural Systems. *International Journal of Systems Research and Information Systems*, 9(2-4), 69-120.

A Functional Architecture Approach to Neural Systems

L. Andrew Coward
email: landrewc@aol.com

A Functional Architecture Approach to Neural Systems

L. Andrew Coward

Abstract

The technology for the design of systems to perform extremely complex combinations of real-time functionality has developed over a long period. This technology is based on the use of a hardware architecture with a physical separation into memory and processing, and a software architecture which divides functionality into a disciplined hierarchy of software components which exchange unambiguous information. This technology experiences difficulty in design of systems to perform parallel processing, and extreme difficulty in design of systems which can heuristically change their own functionality. These limitations derive from the approach to information exchange between functional components. A design approach in which functional components can exchange ambiguous information leads to systems with the recommendation architecture which are less subject to these limitations. Biological brains have been constrained by natural pressures to adopt functional architectures with this different information exchange approach. Neural networks have not made a complete shift to use of ambiguous information, and do not address adequate management of context for ambiguous information exchange between modules. As a result such networks cannot be scaled to complex functionality. Simulations of systems with the recommendation architecture demonstrate the capability to heuristically organize to perform complex functionality.

1. Introduction

A sophisticated technology has been developed for the design of systems to perform extremely complex combinations of real-time functionality. This technology is based on the use of a hardware architecture with a physical separation into memory and processing, and a software architecture which divides functionality into a disciplined hierarchy of software components which exchange unambiguous information. This technology experiences difficulty in design of systems to perform parallel processing, and extreme difficulty in design of systems which can heuristically change their own functionality. These limitations derive from the approach to information exchange between functional components. A design approach in which functional components can exchange ambiguous information leads to systems which are less subject to these limitations. Biological brains have been constrained by natural pressures to adopt functional architectures with this different information exchange approach. Neural networks have not made a complete shift to use of ambiguous information, and do not address adequate management of context for ambiguous information exchange between modules. As a result such networks cannot be scaled to complex functionality.

2. The complexity of current electronic systems

Electronic systems at the upper limits of current complexity include systems performing telecommunications services (Soni 1995), controlling manufacturing facilities (Paul 1992), and performing air traffic control (Ng 1994, Kruchen 1995). Such systems may include tens of millions of lines of software code and thousands of millions of transistors that execute millions of millions of device changes of state per second. The functionality of such systems is divided into thousands of software modules that generate the correct combination of system operations. "Correct" in this context means both logical correctness and the generation of results within time constraints. Delay in connecting an emergency telephone call or in detecting that two aircraft are on a collision course can have catastrophic consequences. Furthermore, system functionality must in general include the ability to detect and recover from faults (Ramos-Thuel 1994).

It is not unknown for systems at an advanced stage of development to experience major functional problems, or even to be abandoned as impractical. The technology to design complex systems frequently outruns academic studies, the literature on the architecture of such systems "relies heavily on informal

examples and anecdotes, and most of [the] work has not been proven to be scalable for large systems" (Soni 1995).

3. Issues which develop with system complexity

There are a number of problems encountered in the design of extremely complex systems. The precise definition of the complex combination of functional requirements which the system must meet is difficult (Perry 1992), and these requirements change throughout the life of the system (Strosnider 1997). Changing requirements can lead to unexpected, undesirable interactions between features (Keck 1998). It is difficult to divide the system functionality into modules which can be designed by different people (Parnas 1985). Reuse of modules for different functions is desirable but difficult (Garlan 1995).

It is also difficult to trace functionality errors from the system problem which is experienced to a level of detail at which the problem can be corrected (Yu 1998). Problems are often encountered in synchronizing tasks which must occur at the same time (Fernandez 1993) and in generating systems results rapidly enough to respond to external events in real time (Avrunin 1998).

It can be difficult to select and integrate the appropriate combination of modules for a system in a specific environment, the configuration management problem. Although the most complex functionality in modern systems is instantiated in software, there is nonetheless the requirement to execute the software on a hardware platform which is also complex. This hardware platform must be designed, and integrated with the software. Software is also constrained by the need to interwork with a hardware platform also subject to change (Parnas 1985).

4. The development of the discipline of software architecture

In the late 1960s it became apparent that the complex functionality represented by software needed to be patterned and structured in a disciplined fashion at a higher level than the software code itself (Dijkstra 1968). In the 1970s Parnas and his collaborators described many of the issues around organization into functional modules made up of combinations of programs (see Parnas 1985 for an overview). In order to break up the system into modules which could be designed independently by different engineers, Parnas advocated "information hiding" in which each module captured a design decision in such a way that the decision could be changed without affecting any other module. Modules require standard interfaces by which they exchange information with other modules. It must be possible to change the internal structure of one module without knowing about or affecting any other module, provided that the standard interface is unaffected. If system functionality is changed, it must be possible to identify all the modules that might contribute to the change. If a module interface must be changed, it must be possible to identify all the modules that could be affected by the change. Types of system functionality changes needed frequently must be possible without changing module interfaces. Less frequent changes can affect module interfaces, but only the interfaces of modules that interface with limited numbers of other modules. These module practices are sound in theory but often difficult to achieve in practice.

Parnas in the 1970s was working on systems with hundreds of modules. As system complexity increased to thousands and tens of thousands of modules it became clear that modules needed to be organized into yet larger units, and a software architecture imposing a style of organization on modules was required (Perry 1992, Garlan 1993). A software architecture provides an intellectual context which can be understood by many designers developing interworking modules. There are a number of different architectural styles including pipeline, client/server, and layering (Buschmann 1996). A consistent style makes it possible to integrate modules developed by different designers, and can make it less difficult to reuse modules developed for one system in a different system (Garlan 1995). A good system design adopts an architectural style appropriate to the problem to be solved. Differences in architectural style are in essence differences in the predominant ways in which information is exchanged between modules. In the pipeline architecture the predominant flow is along a chain of modules. In the client/server architecture the predominant information flow is in a star pattern between clients and central servers, with more limited flow between servers. In the layered architecture the predominant flow is between modules within one layer, with more limited flow between adjacent layers.

A critical aspect of information exchange is the requirement for context. In a commercial electronic system, information exchanged between modules is unambiguous to the receiving component. Modules that exchange information must therefore share enough context for the information to be acted on unambiguously. A major problem in integrating modules designed for different systems is the lack of common information contexts (Garlan 1995).

Overlaid on the information exchange issue is the requirement for performance. A real-time system interacts directly with its environment and has time constraints within which it must respond. For example, a telecommunications switch is expected to provide dial tone within a fraction of a second and set up the connection within a few seconds no matter how many other customers are making telephone calls. A real time system may directly control a physical system based on input data with no human intervention. For example, a telephone switch establishes connections between telephones without a telephone operator. The response of a real-time system depends upon what has previously happened. For example, the connection path between two telephones created by a switch will depend on other traffic and on any fault conditions. Timely response depends upon the completion of many tasks which may have to be carried out concurrently. The problem in real-time system design is to partition the allowed end-to-end elapse times from external event to the deadline for system action between the many modules which may require both time to generate their individual outputs and also input information from other modules which themselves require time to generate that information.

There can be conflicts between partitioning to achieve simple and consistent information interfaces and the need for real-time performance. For example, a pipeline architecture may simplify the information interfaces for some types of problem but result in inadequate performance because modules in the pipeline cannot begin executing until all preceding modules have finished. No available hardware may have the speed to complete the sequence within the required times. There is no universal solution to the problem of achieving real-time performance. Current implementation methods rely on keeping utilization of hardware processing resources below empirically determined bounds and using sophisticated scheduling algorithms (Sha 1993).

5. System architecture and the design process

To design a system performing a complex functionality, it is thus necessary to divide functionality into components, further subdivide components into subcomponents and so on until the most detailed level of functional element is reached. This organization of functionality into a hierarchy of functional components is known as a functional architecture and is illustrated schematically in figure 1. Any component on any level receives input information from the external environment and/or other components, and generates output information which is communicated to other components and/or causes action on the system and/or the external environment.

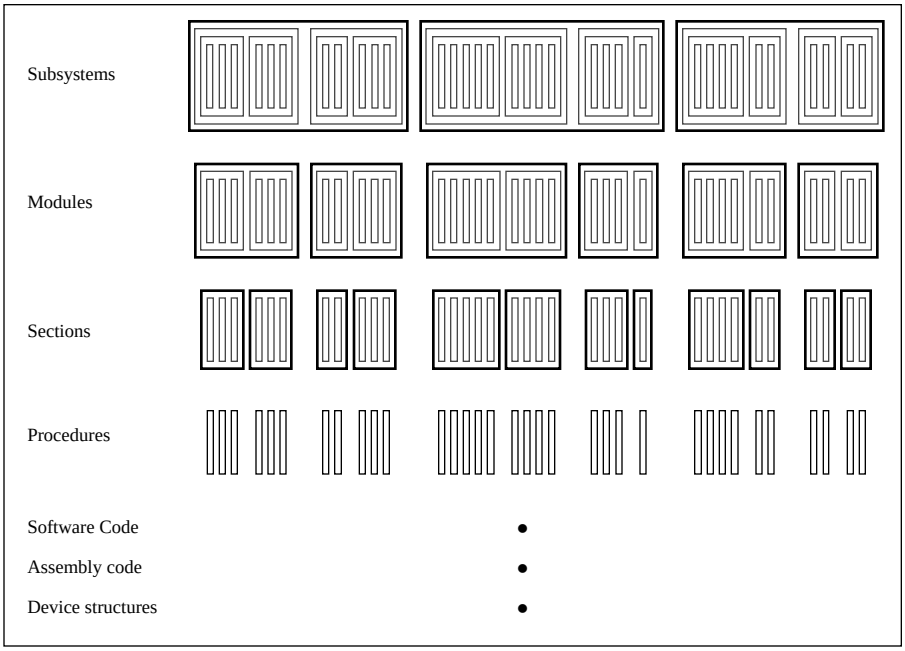

Figure 1. A schematic view of a functional architecture. The functionality of the system is separated into subsystems, subsystems into modules, and so on down to the most detailed functional elements. At every

level the partitioning is a compromize between equal proportions of system operations in each component and minimized information exchange

There are constraints on the form of a functional architecture which result from the general system design requirements described in section 3. These requirements include the need for relatively independent design of components, for diagnosis and repair of problems, and for modification of functionality. These needs all tend to result in a partitioning of components at each level such that all components at one level perform roughly equal proportions of system operations, the volume of information exchanged between the components on one level is minimized, and less information is exchanged by a component with other components than between the subcomponents of that component. A functional architecture which achieves a good compromise between these sometimes conflicting partitioning requirements is known as a simple functional architecture. Such an architecture is also a hierarchy of functional descriptions in which exactly the same functionality is described on every level, but the information content of the description is compressed at the higher levels. Such a description hierarchy is extremely important for intellectual understanding of the system (Coward 1999a).

The reasons that functional architectures are constrained in this fashion can be understood by considering the problems to be solved in more detail. Consider first the diagnosis and repair problem. A failure is experienced as a deficit in a system level feature. To repair the failure it must be possible to follow a relatively simple logical path which relates the experienced system deficit to a functional component or small set of components on a detailed level which can be repaired. The rough equality of functional components and minimized information exchange makes it possible for such relatively simple logical paths to exist.

Secondly, functional changes are in general defined at the system level. Hence it must be possible to create simple, logical paths which relate the system level change to changes at the detailed level, and also which ensure that the detailed level changes do not result in additional, undesirable changes at the system level. The reason less information must be exchanged by a component externally than internally is also to maximize functional separation between components. The minimization of information exchange can also be understood in terms of the need to minimize the information contexts which components must understand, and thus minimize component complexity.

These constraints also satisfy the requirement that functionality be separated into components which can be independently designed by different engineers, and also allows high level design of all the components at one level followed later by detailed design of their subcomponents.

6. Information context and the instruction architecture

The information provided to a component must be adequate to allow it to perform its function. In commercial systems every component has an unambiguous context for any information received from another component (see also Coward 1999a). As illustrated in figure 2, context may be explicit or implicit, but the effect is that the output of any component can be interpreted as an unambiguous command to the system, or instruction. Hierarchies of information are defined from global information for which a context is available to all system components through different levels of local information for which the context is only available within a component at some level in the functional hierarchy. The local information exchanged within a component is not exchanged outside the component, which is another way of describing the compression of the information content of functional descriptions discussed in section 5. The information hierarchy is an architectural view that can be important for intellectual understanding of a system. Because information exchange between components is both unambiguous and minimized, the functions of individual components can often be described in terms of detection and response to cognitive categories on various levels of detail.

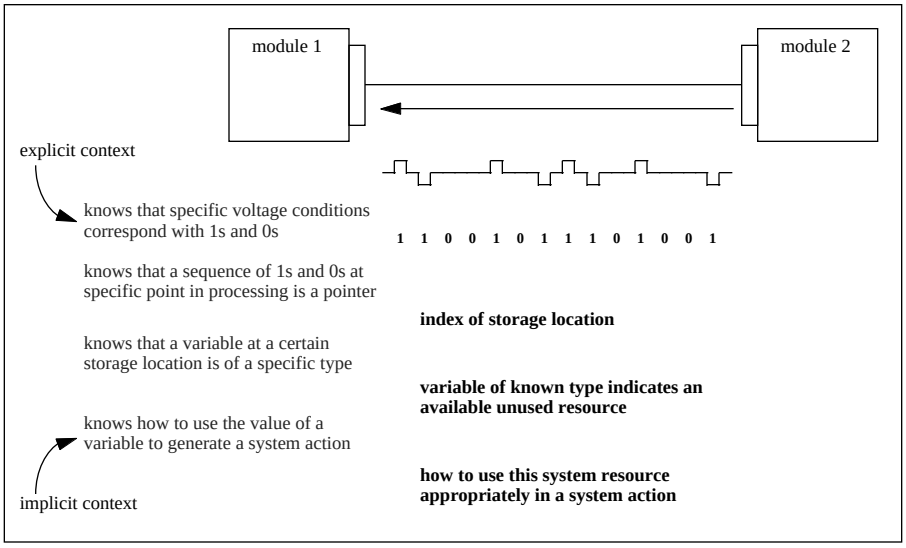

Figure 2. The importance of context for information exchange. In a commercial electronic system, information exchanged between modules is unambiguous to the receiving component

Because multiple components may need to use a given element of unambiguous information, a reference copy of all such information is required. This reference copy is stored in a memory. Because use of information may change that information, execution of components must be ordered sequentially in some appropriate fashion so that two components are not simultaneously changing the same information in inconsistent ways. Information exchange between components must therefore be organized to achieve concurrent tasks and real-time performance. The sequential ordering results in components being executed by a common processing function. The only way to achieve parallel processing is by partitioning information into orthogonal sets acted upon by different components. In a functionally complex system this partitioning must be dynamic and is difficult to achieve. The use of unambiguous information exchange between functional components is thus a root cause of the memory/processing, instruction based, sequential execution paradigm ubiquitous in commercial electronic systems.

In the design process for functionally complex commercial electronic systems the functionality of the system is precisely defined at the user level (the requirements), then partitioned into different architectural views. These views include a view that describes information exchange between components, a view that describes the sequencing of component operations to achieve real time performance, and a view that describes the information context for components. Design proceeds by partitioning functionality experimentally into roughly equal components, then performing both functional and real-time scenarios to test the effectiveness of the component separations from the point of view of minimized information exchange and performance. Partitioning is modified and scenarios repeated until an effective compromise between equal partitioning, information exchange, and real time performance is achieved. The resulting components are then partitioned and the process repeated at the next level of detail. An optimization scenario at a detailed level can sometimes require repartitioning at a higher level. Detailed level components which have worked well in similar systems in the past can sometimes be built up into components at a higher level. Overall this can be an extremely complex design process involving the coordination of the efforts of thousands of engineers.

7. A qualitatively different type of functional architecture

Coward (1990) has argued that a functional architecture in which ambiguous information is exchanged between functional components is possible. The use of ambiguous information means that component outputs cannot be interpreted as instructions, only as recommendations for system action, and in a functionally complex system a range of alternative recommendations must compete for acceptance. A primary separation in such systems is between a functional hierarchy that generates alternative recommendations and a competitive function that selects the most appropriate action. This is the

clustering/competition separation in the recommendation architecture (Coward 1990, 1997a). The functional hierarchy organizes system input information into a hierarchy of repetition similarity. The competition function uses information indicating the consequences of past actions to modulate the probability of functional architecture outputs similar to those which generated the past actions being interpreted and accepted as actions in the future.

However, even if components receive ambiguous information, that information must still be adequate to allow the component to discharge its function. Input information can be ambiguous but not meaningless (Coward 1999a). Components must have a context within which they can interpret ambiguous input information. A central advantage of the imprinting algorithms proposed by Coward (1990) over neural network algorithms is that they can provide a context within which ambiguous information can be interpreted. Provided an adequate information context is provided, the use of ambiguous information relaxes the requirement for sequential execution. Components are tolerant of ambiguity in input information and do not always need to wait on completion of the operations of other components on the same information.

8. Heuristic definition of functionality

A system which heuristically defines its own functionality in the purest sense is given only input information, a portfolio of "atomic" actions which it can combine into behaviors, and some criteria for success and failure conditions. It must define a successful set of associations between combinations of input information (including temporal combinations) and combinations of actions.

Such a system still needs to be able to recover from damage, make multiple use of components, and continue to modify functionality in response to changed conditions. It therefore still experiences pressures to adopt a simple functional architecture. A component in such an architecture receives input information from other components and/or from external to the system and generates output information. If a system heuristically defines its own functionality, component functionality must be heuristically defined. Heuristic definition of component functionality means that a component must change the inputs it receives and/or the algorithms by which it generates its outputs.

If all the information exchanged between functional components is unambiguous, all heuristic component changes must result in components retaining an unambiguous context for all input information. This is an extremely severe constraint, as demonstrated by the impracticality of implementing systems with self modifying code. A system which heuristically modifies its own functionality must therefore be able to exchange ambiguous information between functional components. The system will therefore be forced to adopt the recommendation architecture.

The use of ambiguous information relaxes but does not eliminate the constraints imposed by the requirement for context. A series of ones and zeros, for example, is meaningless without some context for interpretation. In principle there could be many different context frameworks allowing exchange of ambiguous information. One context is the imprinting based approach (Coward 1990) described in more detail in section 10. Alternative contexts would be analogous with different architectural styles in the instruction architecture.

The requirement for context is the reason for the separation between clustering and competition in the recommendation architecture. Consider a system in which many heuristically defined components exchanged information and produced outputs which could be interpreted as different recommendations. Suppose that at a particular point in time a combination of component outputs is interpreted as a particular system action recommendation which is accepted and implemented, and the consequences are unfavorable. Suppose also that information indicating unpleasant consequences is used to change the probability of the currently active components producing similar outputs under similar input conditions in the future. In a functionally complex system the changed components will also provide outputs to other components not currently producing an output. These other components may have heuristically defined their functionality using the original outputs from the changed components. Under somewhat different system input conditions in which the inputs to the changed components are similar but the other components are active, the functionality of these other components has been changed. Direct use of consequence information thus generates unrelated, unpredictable changes in other functionality. To avoid this problem, consequence information cannot be used in the heuristically defined functional architecture, only in the interpretation of the outputs of that architecture.

It could be argued that over a long period the use of consequence information from a wide range of behaviors might result in convergence. Such convergence, if it occurred at all, would be very slow for a

system with high functional complexity. Furthermore, developing a response to a sufficiently novel condition occurring after such a long convergence would result in significant, unpredictable functionality changes and require another long convergence to restore functionality. Another way of looking at this problem is that the use of consequence information introduces the use of information with an unambiguous context. It is then a major problem to ensure that all heuristically defined components share that context.

To avoid this problem it is necessary to exclude direct use of consequence information from the functional architecture. Outputs from the functional architecture are strictly recommendations, which must be competitively interpreted into a system behavior using information on the consequences of similar actions under similar conditions in the past in a completely separate competition function. As pointed out in section 7, this separation is the primary separation of the recommendation architecture.

Because of this separation, the only information which the functional architecture can use to heuristically establish its own structure is system input information, and the only way to organize that information is to search for repetition. Suppose that the inputs to the system are derived from a finite set of detectors, each of which can be in two or more states. The total input to the system at a moment in time is then the state of all the detectors. This input can be modeled using a large information set called the detector state set. Each element in the set corresponds with one system detector, and an element can adopt a range of different values, one for each distinguishably different state of the corresponding detector. The experience of the system can then be modeled as a sequence of detector state sets. The value of an element in one set in the sequence indicates the state of the corresponding detector. An even simpler representation of input would be a set in which an element corresponded with a specific detector being in a particular state. There would then be one element for each possible state of each detector, all elements would be zero except the elements corresponding with the current states of the detectors which would be one. By shortening the interval between detector state sets the sequence can approximate the system input to any required degree of accuracy.

If a sequence of detector state sets is the only information available to generate the functional hierarchy, the only way to organize the information is to look for repetitions of information combinations. Such combinations could include relationships between different elements at the same time (e.g. one element is larger than another) and relationships between elements at different times (e.g. one element in increasing and another is decreasing). For a pure heuristic system the only way to find useful combinations would be to select combinations at random and discard combinations which very rarely or never repeat. A less pure heuristic system could have a priori programming of combinations likely to occur frequently, and even of combinations likely to prove useful in discriminating between conditions in which different behaviors would be appropriate, but could not directly use real-time consequence information in heuristically defining its structure.

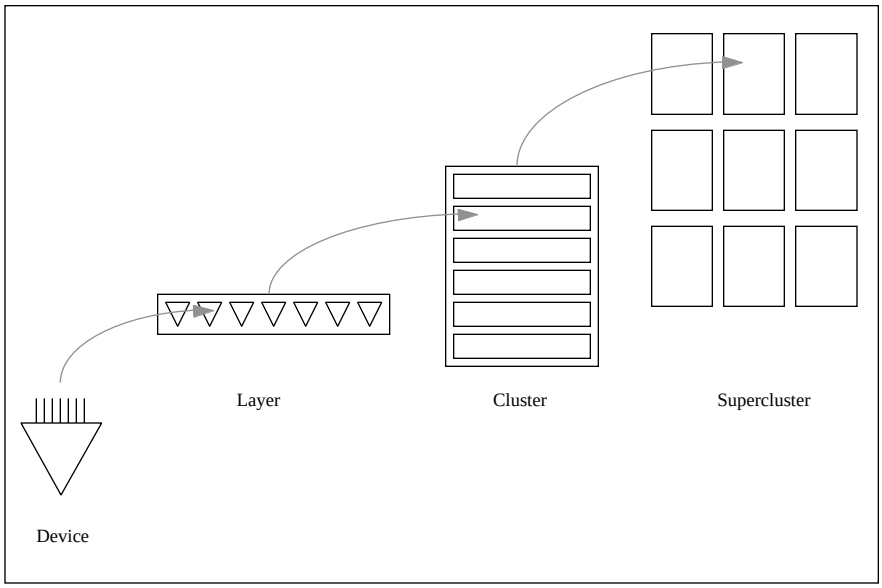

Figure 3. A simple hierarchy of repetition in which the most simple repetitions of information are programmed on devices, devices are combined in layers and so on.

A functional hierarchy of this type is a hierarchy of repetition similarity. Simple repetitions are combined into more complex repetitions, these complex repetitions into yet more complex repetitions and so on. If the most simple repetitions were programmed on devices, devices could be combined in layers, layers in clusters, clusters in superclusters etc. as illustrated in figure 3. The a priori programming discussed in the previous paragraph could include partial preprogramming of information distribution routes, rules for changing the distribution, similarity criteria and rules for changing the criteria for the type of structure illustrated in figure 3, but the rules could not include any use of consequence information.

A programmed combination at one level is made up of a set of information combinations programmed at a more detailed level. Repetition of a combination occurs when a significant subset of its constituent repetitions repeats. This hierarchy of repetition is the clustering function of the recommendation architecture, and the devices, layers, clusters etc. are the functional components on different levels of detail.

At each level, an output from a component indicates detection of the programmed repetition which corresponds with an action recommendation at the component level of detail. Outputs at the highest level correspond with full system action recommendations which compete for control of system behavior in the separate competitive function. The clustering function is evolved to generate output at the highest level in response to any detector state set (or perhaps any small sequence of sets). This requirement ensures that a behavioral recommendation is generated under all conditions. It is possible for the competitive function to reject all recommendations.

The heuristic evolution of a clustering structure would assign any input condition to the clusters to which it was most similar, as indicated by the proportion of their constituent repetitions which are activated. New clusters could be added if an input condition produced minimal response in all existing clusters. Detailed algorithms to achieve this type of heuristic clustering are described in section 10.

If information exchange were unambiguous, the hierarchy illustrated in figure 3 would reduce to a pattern, category, supercategory structure. A cluster does not correlate unambiguously with a cognitive category. A particular cluster might often be present when an identifiable cognitive category was present, but it will also sometimes be absent when the category is present and present when the category is absent. Categories can only be unambiguously identified through a competitive function.

There are a huge number of different cluster hierarchies which would produce an output in response to every one of a given long sequence of detector state sets. If the purpose of the cluster hierarchy were to identify separate well defined categories with the maximum efficiency, then orthogonality of the individual cluster repetition definitions might be valuable. Such a cluster set could even be set up by design. However, if the purpose of the cluster hierarchy is to control a complex functionality, such orthogonality would result in sequential execution and a memory/processing separation. Problems of category identification can be algorithmically complex but are functionally simple because system outputs do not change system inputs (Coward 1999a).

A cluster hierarchy in which all inputs produced outputs in response to all detector state sets would probably not provide enough discrimination for any practical purpose. A compromise is required in which at minimum clusters repeat frequently, but not always at the same time. Reaching such a compromise does not require the use of consequence information. A functionally effective compromise will require rather more than this minimum. There is a mechanism by which consequence information can be used indirectly to optimize the functional hierarchy without affecting the context for information exchange.

If a cluster hierarchy produces similar outputs at different times, and these outputs are accepted as behaviors, the behaviors will be similar. If the consequences of behaviors generated by similar outputs at different times are contradictory, this condition indicates that the cluster hierarchy is not sufficiently discriminating in its definition of repetition. The indication of contradiction could be used to generate additional outputs from the active clusters, the outputs being based on larger combinations of the constituent repetitions of the cluster, and additional inputs which the cluster as a whole could add. Provided that no existing outputs are changed, clusters using those outputs will not be affected, but the additional outputs would allow the competitive function to discriminate between the contradictory consequence conditions.

A further issue is how to determine the appropriate number of clusters for a given functional problem. If a maximum number is defined in advance, then the system can be evolved until every experience produces an output from one or more clusters. For example, new clusters could initially be established only when an

experience is novel by some definition, then the similarity criteria of the existing clusters could be gradually expanded until all experiences produce a response. This algorithm for similarity and establishing new clusters also needs to be specified in advance, although it uses heuristically determined parameter values. If the maximum number of clusters were not specified, an algorithm for similarity and establishing new clusters is still required.

The algorithms for similarity and establishing new clusters could be determined arbitrarily by the system. Such an approach might result in viable behavior but would probably also result in excessive use of resources. However, some algorithms will result in more effective functionality because they result in a good compromise between maximum discrimination between experienced conditions and minimized use of resources. If system resources are limited, discovery of such algorithms heuristically would require the ability to discard the clusters resulting from early, inefficient algorithms, resulting in loss of the value derived from the experience. There are considerable advantages in effective algorithms being defined in advance by design (or programmed genetically by natural selection in a biological system). Such algorithms could allow some flexibility in the total number of clusters within resource constraints, and as discussed in sections 11 and 13 would include initial information distribution conditions and how the distribution is modified heuristically. Although the algorithms include heuristically determined parameter values, these values cannot make direct use of consequence information for the architectural reasons discussed above.

The separate competitive function must interpret different repetitions as recommendations, and select or reject recommendations on the basis of the success or failure of similar recommendations under similar conditions in the past. The general mechanism suggested in Coward 1990 to achieve this function is a parallel pipe structure (illustrated in figure 9). Each pipe corresponds with a different behavioral type. Alternative recommendations of different types enter their corresponding pipes. Activity in one layer of a pipe stimulates activity in the next layer of the same pipe but inhibits activity in later layers of the other pipes. This competition between stimulation and inhibition is adjusted so that an output emerges from at most one pipe. Such an output allows the corresponding actual cluster outputs to be applied to control of behavior. Functionally simpler systems could use simpler algorithms such as the algorithm used in the simulations described in section 14.

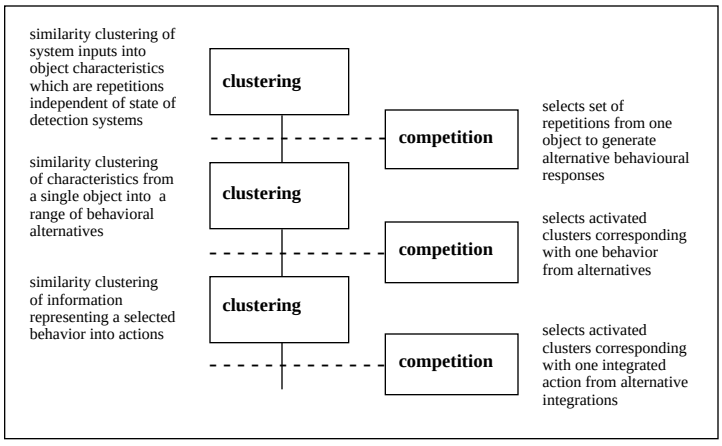

Figure 4. The major functional separations of the recommendation (or pattern extraction) architecture are repetition similarity clustering of ambiguous information and management of the competition between activated clusters for access to the next stage towards control of system action. The repetitions and clusters in the recommendation architecture are similarity clusters. If unambiguous information were used, these repetitions and clusters would be cognitive patterns and categories.

An actual system could have a series of clustering and competition stages as illustrated in figure 4. Each competitive stage reduces the information provided to the subsequent clustering. An interpretation of such a series of stages in terms of the mammal brain is shown in figure 4, following Coward 1990.

Components using unambiguous information cannot interwork with components using ambiguous information because of the lack of context for exchanged information. The only way to mix instruction and recommendation architectures is to use the (unambiguous) output of an instruction architecture as (ambiguous) input to a recommendation architecture, and/or to use outputs of a competitive function as inputs to an instruction architecture. In other words, instruction and recommendation components must form completely separate systems.

9. The device basis for heuristic definition of functionality

In a functional system, functionality at detailed levels is driven by functional separations at higher levels, within constraints set by the availability of suitable hardware devices. In the instruction architecture, high level functionality is coded down to assembly code. Assembly code is designed to be executed by the available hardware elements which are generally at a somewhat high level than transistors (e.g. AND, OR, NAND etc.) and may be much higher level (e.g. processor devices).

There are two aspects to functionality at any level. One is the output generated immediately in response to an input, the other is any changes to the portfolio of outputs produced in response to future inputs (Mira 1997). The latter aspect in the instruction architecture would correspond with self-modifying code, which as discussed earlier is precluded by the difficulty of maintaining an unambiguous information context. In the recommendation architecture a natural device functionality is detection of the presence of combinations of input information. These combinations are repetitions that may be either predefined or heuristically defined. Change to a programmed repetition or the response to such a repetition will in general require the presence of information corresponding with a higher level functional signal recommending the change. Repetitions programmed at the device level are not patterns in any cognitive sense because the information defining the repetition is ambiguous. The presence of a programmed repetition is only an indication that a condition may exist, and the output is a recommendation to the system to pay attention. The strength of the output (e.g. the rate of firing) could be used to indicate the strength of the recommendation. Correlations in the time variation in the strength of output (e.g. a common firing modulation rate) could be used to indicate a population of devices which have detected repetitions within a common set of information. Such a use of second order modulation resembles the 40 Hz frequency in biological systems as interpreted by Llinas et alii 1994. Different populations could be distinguished by different modulation rates.

The requirement for simple functional separation means that in general the repetitions programmed on a single device will have similar functional interpretations, tending to recommend similar system responses. Simple functional separation of this type could require considerable algorithmic complexity, for example if the time sequence in which information elements appear is part of the definition of the programmed repetition.

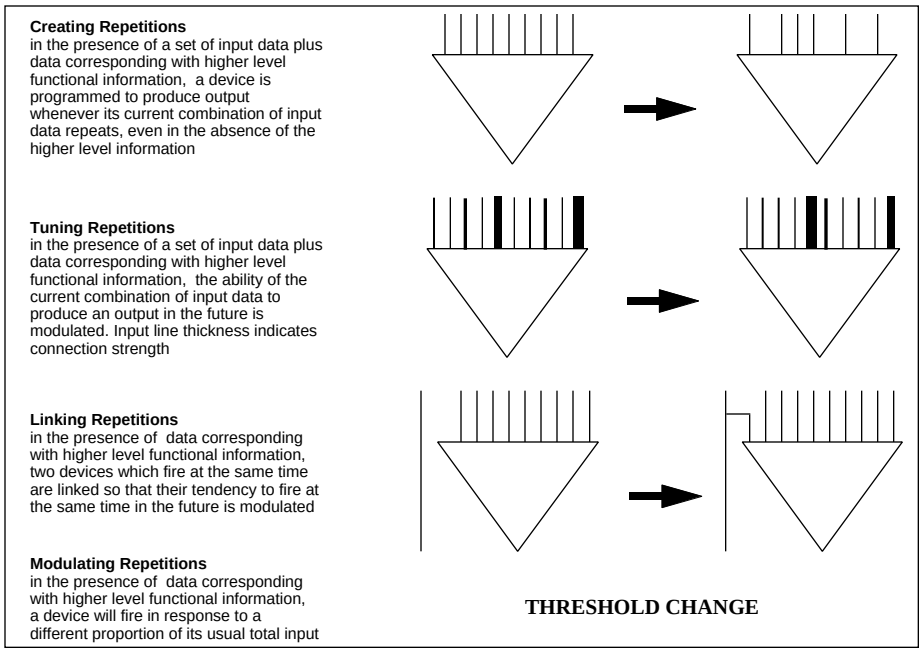

Figure 5 Alternative ways in which the repetitions of ambiguous information programmed at the device level can change. Such change requires in addition the presence of information corresponding with the output of higher functional components recommending change.

Changes to programmed repetitions can be of four general types. These types are repetition creation, repetition tuning, repetition linking, and repetition modulation as in figure 5. The types are important for different types of recommendation architecture functional separations as discussed in Coward 1997b. Repetition creation occurs in the presence of a set of activated regular inputs plus a set of active inputs representing the presence of a higher level functional output recommending repetition creation. Under these conditions the device is imprinted to produce an output immediately, and at any time in the future if the same set of regular inputs repeats, even in the absence of the higher level signal. This concept was introduced in Coward 1990, and in its simplest form a device with a large number of inputs is programmed at a single point in time to a single repetition by deletion of its inactive inputs and setting its threshold at or below its current total input strength. More realistic models with multiple repetitions per device are discussed below. This type of change can support instantaneously created permanent declarative memory like traces. Tuning of repetitions is the standard perceptron model (Rosenblatt 1961) in which the relative weights of different inputs are adjusted. Connecting repetitions is a variation of the standard Hebbian mechanism (Hebb 1949) in which the connection between two devices that frequently fire at the same time is strengthened. The variation is that a connection may be established if two devices frequently fire at the same time. A fourth type of change is modulating repetitions, in which the total active information required to generate output (i.e. the threshold) is modulated by the presence of higher functional information.

The imprinting mechanism is the basis for heuristic clustering. As discussed in section 10, imprinting mechanisms can proceed without the use of consequence information. In addition, because repetitions do not change after the first imprinting, there are no problems with communicating changes of information contexts to any devices which may have heuristically established an input from the device prior to the change. Tuning of repetitions is important in the competitive function, where consequence information is used as the basis for adjusting relative weights. Connecting repetitions is important in the management of information distribution. Modulating repetitions can be based on only repetition information (i.e. independent of consequence information) and in this form can play a role in modulating the relative probability of different behavioral recommendation types being generated.

10. Detailed functional partitioning within heuristically defined functionality

The device imprinting mechanism makes it possible to sort a sequence of experiences into a set of repetition clusters as described in general in section 8. If all information exchange were unambiguous, these clusters would be cognitive categories. Because information exchange is ambiguous a separate competitive function must use cluster outputs to generate system behaviors, which could include developing an appropriate response to members of different categories under different conditions. Note that the appropriate response to a member of one category could vary depending on other conditions.

To give a very simple illustration, consider a system that experiences sequentially a large population of animals of five different types A, B, C, D, and E. These experiences create a sequence of detector state sets as described in section 8.

The similarity clustering process described in detail below could create a set of clusters, for example eight numbered one through eight. Each cluster would indicate the repetition of an information condition, the specific output from a cluster indicating variations within that condition. A competitive function using correct/incorrect feedback could associate, for example, strong output from cluster three, weak outputs from clusters five and six with a behavior appropriate to animal type B. The detailed combination of outputs could be used to vary behavior with individual examples of B. The correct/incorrect feedback does not affect the similarity clustering, the separation between clustering and competition is the key functional separation which makes a complex system functionality possible.

Figure 6. Connectivity of a cluster module using the repetition imprinting mechanism at the device level. The layers perform similarity subfunctions. Five layers are illustrated, six are discussed in the text. Layer to device connectivity corresponding with functional output from the layer requires connectivity from multiple layer output devices. Separating functionality between layers allows independent optimization of the separated functions.

In figure 6 a repetition similarity cluster module made up of a series of layers of devices which can be imprinted with repetitions is illustrated. The regular information flow is forward through the layers, and the repetitions programmed in one layer are therefore combinations of the repetitions programmed in the preceding layer. In addition the layers are functional separations, and five layers are illustrated corresponding with five separate similarity functions. Layer α receives inputs from sources external to the cluster which indicate the presence of relatively simple information combinations from system input or the outputs of lower complexity clusters. The function of layer α is to select the sources from which information will be allowed to influence the cluster. This selection is part of the minimization of information distribution discussed in section 13, and could be influenced by the detection of contradictory consequences. Such detection could recommend expansion of the sources of inputs to the cluster if the expansion of γ layer inputs from within the cluster described below does not solve the problem alone. Layer β recommends the imprinting of additional repetitions in all cluster layers in the presence of enough of its programmed information repetitions. Layer γ in the illustration has two functions. One is to inhibit imprinting in all layers, the other is to generate combinations of outputs from the cluster which are in general unique to individual conditions. The latter function is particularly important for implementing the ability to respond to detection of contradictory consequences, the γ layer must draw on a wider range of information, perhaps from multiple layers in the cluster, to generate additional outputs. Layer $\varepsilon 1$ receives input from higher complexity clusters which recommends that any currently active target devices continue to generate output. The $\varepsilon 1$ function relates to maintaining repetition orthogonality and is discussed later. Layer $\varepsilon 2$ outputs to inhibit the creation of unused clusters. The separation of functionality into layers means that each functionality can be optimized independently without affecting the others. The combination of two functions in the γ layer means that within a system performing very complex functionality six layers would be needed, but have not been drawn because of diagram complexity.

A layer could be made up of multiple levels of devices. In such a case, the repetition complexity in the regular information flow increases from level to level, but any outputs from the layer at the higher functional level have a common function. For example, if $\varepsilon 2$ were made up of several levels, any $\varepsilon 2$ outputs from any level directed to an unimprinted module would function to inhibit imprinting. There could even be subfunctions within such a layer, if for example such outputs only originated from the final level of the layer.

To simplify the discussion, the functional logic with one repetition per device will be described first. Regular devices have already been imprinted with their repetition, virgin devices have not yet been imprinted. An unused cluster is made up entirely of virgin devices. For a cluster that has been operating for some time, the inputs to virgin devices are randomly assigned with a statistical bias in favor of inputs that have frequently fired regular devices in the same layer in the past. Inputs to the first layer come from external system input sensors or from clusters programmed with simpler information combinations. The function of the first layer is to limit the number of different source clusters to a set that frequently produce outputs at the same time. In an unused cluster the inputs to the all-virgin first layer have a statistical bias towards combinations which have frequently occurred together when no cluster at a peer repetition complexity level has produced an output. The statistical biases reflect the importance of minimizing information distribution in a simple functional architecture. Mechanisms by which the biases can be achieved are discussed below.

In the presence of inhibitive signals from layer γ no virgin devices are imprinted. In the absence of such inhibitive signals and in the presence of stimulative signals for layer β, the threshold of a virgin devices gradually decreases until either it fires, its threshold reaches a minimum proportion of inputs, or imprinting is cut off by inhibitive γ output.

Consider now a system made up initially of a large number of unused similarity clusters. The system goes through a series of experiences. During each experience the system is exposed to a sequence of objects of type A, B, C, D, and E as described in the animal scenario. Between each experience phase, the system is taken off line and input simulating a fast rerun of recent experience is provided to the system. All the regular devices fire in proportion to their firing frequency in the past, with a bias towards the most recent past. Virgin devices accept random selections of inputs from the normal sources for the layer with a statistical bias towards inputs which frequently contribute to firing regular devices in their peer layer. This process is one aspect of the minimization of information distribution in an heuristically defined functional architecture discussed in section 13 and was proposed in Coward 1990 as the functional role of sleep with dreaming in mammal brains.

The system can only activate at most one unused cluster per experience period, and only if that cluster has been preconfigured in the previous functional rerun or sleep phase. The sleep phase configures one unused cluster with inputs to the α layer biased towards inputs that frequently occurred together with no $\varepsilon 2$ outputs from any existing cluster. In the subsequent experience phase, if an object is experienced which generates a high proportion of the favored inputs to the unused cluster and no outputs from the $\varepsilon 2$ layer in any existing cluster, imprinting at all levels occurs until an output results. From that moment onwards, another object will generate output or imprinting until an output results if the object generates enough similar information combinations to create a significant level of output from the β level. A qualification on this statement is that enough virgin devices exist with appropriate input supersets to support a path to output. If these resources are inadequate there is a high probability that appropriate resources would be configured in the next sleep period.

Consider now how such a system would evolve through a series of experiences. New clusters would continue to be configured until enough existed that the system produces an output in response to a high proportion of objects. The output from $\varepsilon 2$ inhibits proliferation of new modules, and has the additional function during sleep of increasing the stimulative strength of functional β layer outputs if the cluster frequently produces $\varepsilon 2$ functional inhibition and no output. The balance between these requires a design choice of the number of clusters that will be established on average at any peer level. These mechanisms ensure that eventually a high proportion of objects will generate an output from one or more cluster.

As the simulation results discussed below demonstrate, such a system creates a set of clusters that may number more or less than the number of cognitive categories of objects. Because the particular combination of γ outputs from a cluster is in general unique to the individual object, the cluster outputs can be used by a competitive function to generate high integrity category specific behavior. This competitive function requires correct/incorrect feedback which can itself be ambiguous. Unlike neural network approaches the correct/incorrect feedback does not affect similarity clustering. One result of the consistent use of ambiguous information is that there are no separate learning and operational phases, for example, a new cognitive category can be added after responses to the other categories have been established, and the new category accommodated with minimal effect on existing functionality.

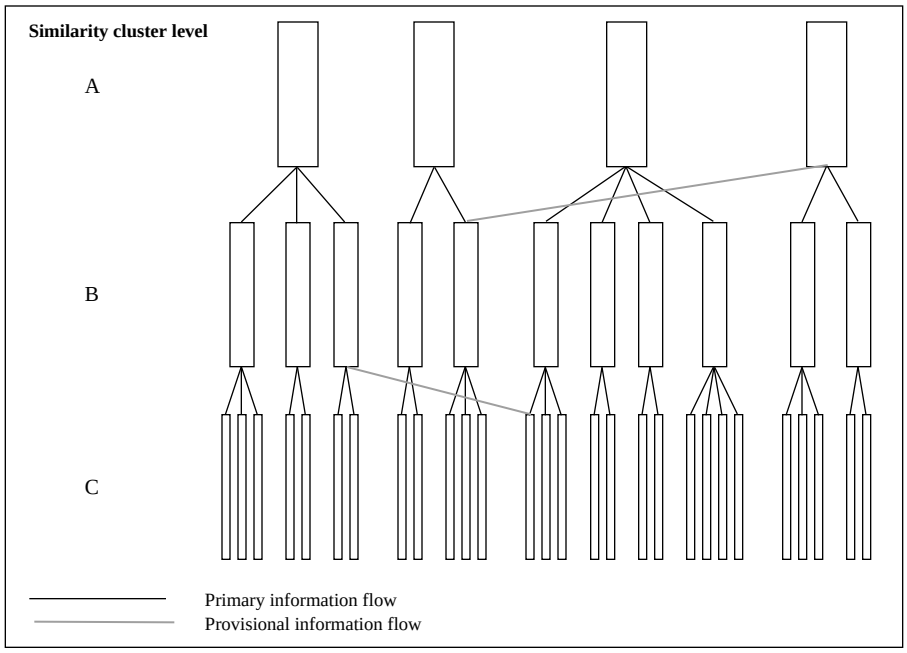

Figure 7 An heuristically defined hierarchy of repetition similarity can be created which can make use of repetition clusters on one level to define more complex repetition clusters on later levels. Such a hierarchy could also integrate cluster sets developed from different input information domains.

The creation of a set of clusters can be generalized into creation of a hierarchy of repetition similarity as shown in figure 7. The outputs of each cluster at the first level A are themselves repetition similarity clustered into clusters at level B. Outputs from clusters at level B are repetition similarity clustered into clusters at level C. Information cross linkages can be established with outputs from a cluster at one level influencing the subclusters of a different peer cluster. In principle every cluster on every level could output to every other cluster on every level, but this high level of information distribution would conflict with the need for a simple functional architecture. The requirement is that only the connections most likely to be functionally useful are established. Unless there is a separate functional role for a connection, the most useful connections will tend to be from one level to the next. A connection between two modules is most likely to be functionally useful if the two modules tend to be active under similar system input conditions. This simultaneous activity requirement can be met in a similar manner to the way in which device inputs are managed. The management process of rapid experience rerun can be used to establish provisional connections from the outputs of a cluster to the inputs of a cluster at the next level which tends to be active at the same time as the source cluster. The result of this process is a hierarchy of repetition similarity which is heuristically defined.

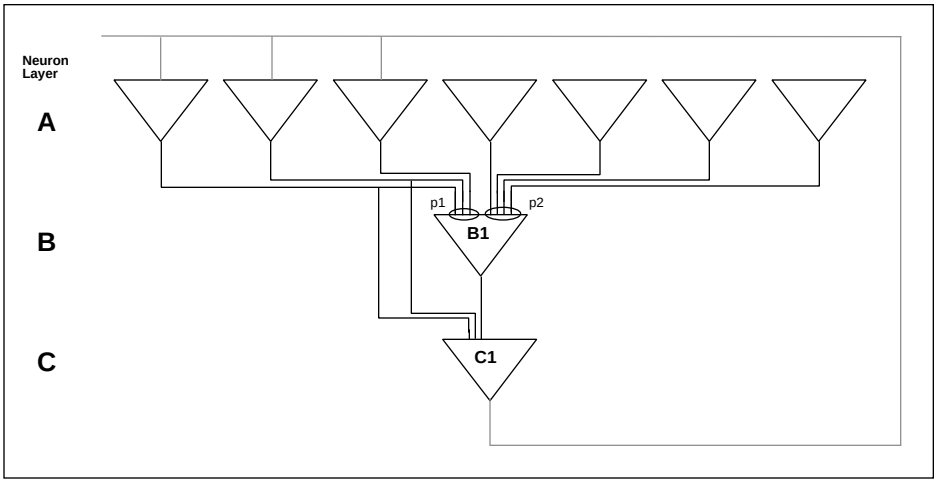

Figure 8. Repetition orthogonality problem and solutions. If a device is programmed with multiple repetitions, target devices must distinguish between outputs due to appropriate repetitions and inappropriate repetitions. For example, repetition p1 on B1 is appropriate for C1, but not p2 on B1. One mechanism is to forbid sharing of inputs between different repetitions on the same device and allow some inputs to reach the target from earlier levels. A less restrictive mechanism is to combine partial input orthogonality with feedback to lower complexity devices, which results in continuing output from the lower level only if the higher level is present. The system will then converge on a consistent set of device activations

The simple imprinting algorithm used in the above discussion can be modified to allow devices to be programmed to extract multiple repetitions imprinted at different times. Such a modification must solve the orthogonality problem illustrated in figure 8. How can a target device C1 in layer C which has been imprinted with a set of repetitions including B1(p1) in layer B be prevented from responding to the activation of B1(p2)? The simplest solution is to require that multiple repetitions on a device like B1 do not have any inputs in common, and that their targets like C1 also receive some inputs from layer A which is the primary source for inputs to B1. This structure is still feedforward, but net repetition complexity increases somewhat more slowly from layer to layer than in the simple case. In practice this approach would probably sustain a practical degree of orthogonality even if there were a small degree of input sharing between different repetitions on the same device. If the sharing becomes significant, an additional mechanism is feedback connections from C1 back to earlier layers such as A. Such feedback connections could be established provisionally and made regular if both devices were active at the time of imprinting of a repetition on C, and have the effect that an output from the recipient of the feedback would only continue if significant feedback were received. In a real system this feedback will be more effective if directed back from more complex clusters which receive output from the cluster containing C, and the function of receiving such feedback separated in a specific functional layer as in figure 6.

Such a mechanism would drive convergence on a consistent set of repetitions through many layers, the consistency being that the set as a whole have tended to be activated at the same time in the past. Cauller (1997a) has proposed that the extensive feedback connectivity observed in biological brains has a predictive role and drives convergence of a chaotic system towards an attractor. The mechanism described here is functionally equivalent but expressed in recommendation architecture terms. Understanding of a system activation will be most accessible in the applicable functional paradigm.

The feedback mechanism allows simple separation of the functionality to maintain orthogonality between the multiple repetitions imprinted on a single device. Higher level components can therefore use either the simple or more complex device functionality to achieve the same higher level functionality. The simple mechanism can therefore be used to understand the higher level functionality in device terms, the separate orthogonality mechanism can be added if required for system operating efficiency reasons.

A higher level in the functional architecture is the supercluster made up of a hierarchy of clusters as illustrated in figure 7. In a system with a number of superclusters, each supercluster would generate a different general type of behavior. For example, in a biological system such general types could be aggressive, fearful, food seeking etc. Such superclusters can be modulated by system recommendations

derived from information indicating system needs to influence the relative probability of different behavioral types, corresponding with anger, fear, and hunger etc. in a biological system (Coward 1990). The same cluster hierarchy could in principal be used for different superclusters, with the same outputs being interpreted differently by different components of the competitive function. Such multiple use would preclude modulation of relative behavior probability within the functional architecture, although it could still act upon the competitive function. Multiple use would also preclude the use of initial configuration and contradictory consequences information to tune the hierarchy for the behavioral type as described above and in section 8. Although for these reasons the superclusters will be different, heuristic definition of different cluster hierarchies for each major behavioral type will result in duplication of experience records with some consequences discussed in section 15.

The ambiguous component outputs generated using the imprinting algorithm are adequate for communication between heuristically defined components for several reasons. The first reason is that outputs can communicate both detection of an information condition meeting an internally defined similarity criterion and variations within the criterion. The combination of γ outputs from a cluster generated in response to a sufficiently similar condition is a compressed indication of all the repetitions activated within the cluster. The second reason is a hierarchy of functional signals based on similarity at different levels is supported. Thirdly, the algorithms allow modulation of the similarity clustering based on its discrimination effectiveness for functional purposes without using direct feedback of consequence information that would introduce a requirement for unambiguous information into the functional hierarchy.

As a result the functionality of the system can be dynamically defined. An experimental but plausible response can be generated immediately in response to a novel experience provided that there are some elements of similarity to past experiences. Experience will improve the response, but there is no need for a long training period before any response is possible.

11. The competitive function

In the recommendation architecture the second major functional separation is competition. The competition function can be viewed in two ways. From an information point of view it operates to reduce the volume of ambiguous information which proceeds to the next repetition clustering function. To say the same thing in functional terms, it operates to select one of the available action recommendations to proceed towards implementation. In the model proposed for the mammal brain in Coward 1990, the first stage of repetition clustering extracts repetitions corresponding with sensory system independent repetitions (for example, repetitions correlating ambiguously with object color independent of illumination). There is competition between "primal sketches" (Marr 1982) correlating with different cohesive sensory domains, and the repetitions from the bounded sensory domain corresponding with the winning primal sketch proceeds to the next repetition clustering stage.

This first competition corresponds with the high level function of attention, and could for example operate by tagging all the appropriate repetitions with a firing modulation frequency, with such tagged repetitions being the ones allowed to proceed. The second repetition clustering stage generates outputs that can be interpreted as alternative action recommendations towards the object or condition that is the focus of attention. The second competitive stage selects one recommendation for implementation, with the result that the information corresponding with that recommendation is allowed to proceed to a third repetition clustering stage, in which the output clusters correspond with a portfolio of direct physical actions. The structure proposed in Coward 1990 to implement this second competitive function is illustrated in figure 9. The structure consists of a set of parallel pipes, and the information corresponding with different types of recommendation enters different pipes. From layer to layer within a pipe the connections are stimulative, between pipes the connections are inhibitive.

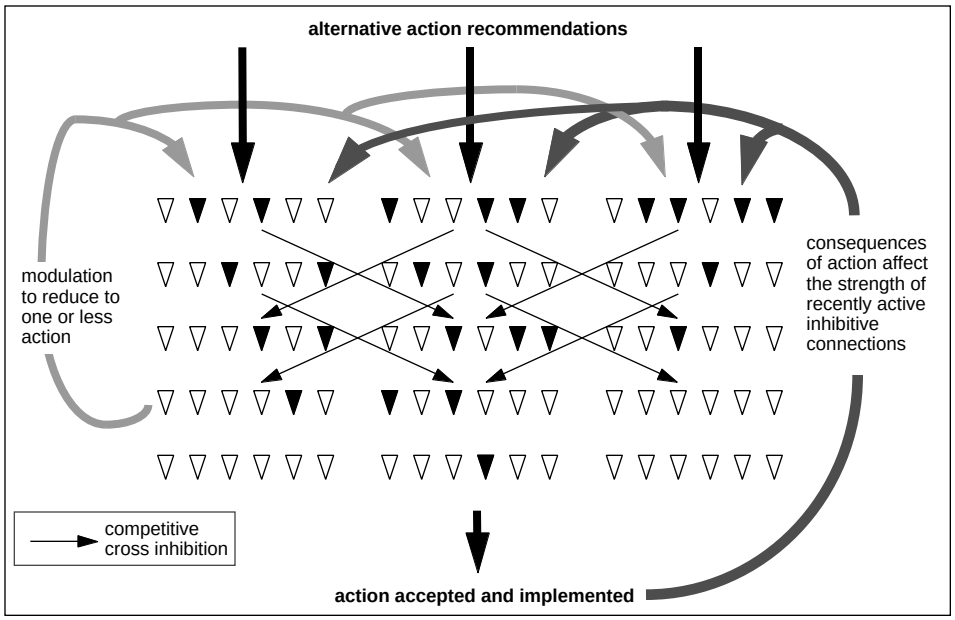

Figure 9. A competition structure. Active information representing alternative action recommendations of different types enters separate pipe like paths. The recommendation in one path inhibits the transmission in all other paths. If more than one recommendation approaches output, a feedback loop damps down all paths to generate not more than one output. Feedback on the consequences of an accepted recommendation changes the strength of recently active inhibitive connections in the successful path and thus modulates the probability of similar actions in the future.

As recommendations pass down the pipes they therefore weaken each other. Thresholds across the entire structure are modulated to ensure that activated information emerges from at most one pipe. This single pipe output triggers the release of the full set of corresponding recommendation information to the next clustering stage. Once the action has been carried out, sensory input is clustered to detect (potentially ambiguous) favorable or unfavorable information activation sets. A favorable activation set is functionally a recommendation to weaken all recently active inhibitive connections and strengthen all recently active stimulative connections in the successful pipe. An unfavorable activation set is functionally a recommendation to strengthen all recently active inhibitive connections and weaken all recently active stimulative connections in the successful pipe. This input strength modulation process modulates the probability of similar inputs to a pipe gaining control of system action in the future.

The clustering process generates very high volumes of information, and the intermediate competitive stages can be viewed as mechanisms to reduce the volume of information which needs to be clustered in the later stages, while retaining the information most likely to be relevant in defining a consistent, fully appropriate behavior.

There are two conflicting requirements which must be resolved in defining the relative arrangement of clustering and competitive functions. One is the need to have any competition operate on as wide a range of information as possible, to minimize the chance of a false local priority gaining control of action. The local priority effect can be illustrated by considering a scenario in which I arrive home and there is competition for control of my behavior between a barking dog inside, a door, a cat brushing my leg, and a key in my trouser pocket. If information from the dog competed with information from the door and the door won, and separately information from cat and key competed and cat won, and finally information from door and cat competed, then cat might win. If the competition took place in one stage, the presence of door information might result in key information being the overall winner and I would unlock the door. Local competition will therefore rarely be appropriate for complex functional combinations.

The other requirement is the need to minimize the total volume of information which a clustering function must handle, to allow the search for repetition similarity to focus on the most relevant information

and avoid confusion from excessive activation. The appropriate compromise will differ for different system functions. Some alternatives are illustrated in figure 10.

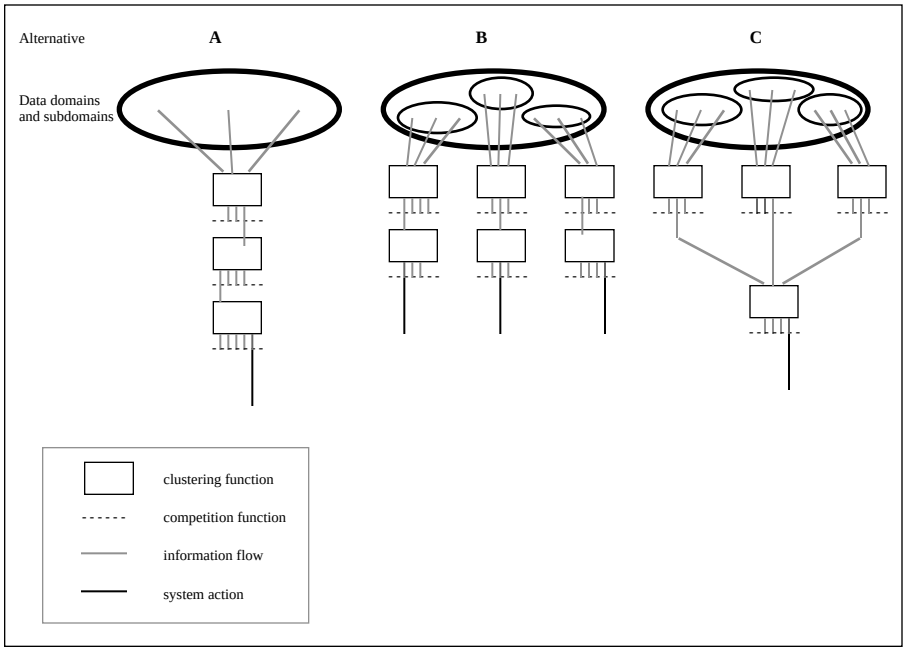

Figure 10 Alternative arrangements of clustering and competition to meet the requirement for availability of appropriate information but information reduction to minimize distribution and load on clustering functions.

The simplest alternative A in figure 10 is if the total input information domain is processed in parallel through a single series of competitive stages to system action, which assumes that the most appropriate subset can be selected at each stage. The other extreme is alternative B, in which information subdomains are defined which contain all the information required for different types of system actions which can be implemented independently. An intermediate case is alternative C in which selection of appropriate subsets can be made separately in separate domains, but the subsets must be clustered together for the highest integrity behavior. Other cases include the output of a clustering/competition combination which acts on either a competitive function or on a clustering function for system management purposes.

Real systems will require combinations of these alternatives to achieve a good compromise between minimizing and maintaining integrity of information flows. Any combination of alternatives must also maintain the clean separation between clustering and competition to achieve a simple functional architecture.

12. Differences between neural networks and the recommendation architecture

There are a number of major differences between neural networks and the recommendation architecture. In general in neural networks no approach to partitioning of a complex functionality is discussed. Investigation of concepts for more complex networks describe the organization of the network in terms of feature combinations (e.g. Andersen 1997). Even when modules are organized into hierarchies, different levels in the hierarchy are interpreted as detection of features of different complexity (e.g. Sabisch 1998). Any identification of function with feature implicitly carries over paradigms from the instruction architecture in which use of unambiguous information makes such identifications possible.

A second difference is that although the distinction between unsupervised and supervised learning (see e.g. Card 1998) is analogous with the separation between clustering and competition, the use of consequence information (e.g. category identification) in both types of neural networks means that the essential architectural separation is not maintained. In neural networks the term "competition" is used to indicate any process of resolution between alternative activations (e.g. Taylor and Alavi 1993).

Competition in this sense occurs in both separations of the recommendation architecture, but the competition function of that architecture, competition refers strictly to the use of consequence information to resolve between alternative functional recommendations.

A third difference is that the context for information exchange between modules is not addressed in neural networks. The typical approach of targeting module outputs on specific vectors provides inadequate information richness for ambiguous communication and in fact drives such outputs towards unambiguous category identifications. For example, a typical problem attempted by Kohonen Nets is self organization of natural language information (Honkela 1995) in which map outputs indicate individual words and relative position on the map indicates the syntax of the word. The use of device algorithms in which relative input weights are changed makes it difficult to maintain information context as discussed earlier.

A fourth difference is that there is little concept of a hierarchy of functional signals. The only exception is in the case of Adaptive Resonance (Carpenter and Grossberg 1988) which makes use of an indication of input similarity as a precondition for learning. This is a single higher level functional signal, but multifunctional, multilevel functional signals as described in section 10 have not been developed.

As a result, neural networks can only be applied to stable, functionally simple problems. An example is recognition of features in MRI brain scans (Sabisch 1997). Such problems are algorithmically complex but functionally simple in that system outputs do not dynamically change system inputs (coward 1999a). They are stable in that any required change to the problem such as recognition of new types of feature would require an extensive relearning process including previously learned features.

The recommendation architecture offers an approach to the design of functionally complex systems in which functionality can be defined and evolved heuristically.

13. Design of a System with the Recommendation Architecture

A simple functional architecture is a good compromise between partitioning of functionality into equal components and minimum exchange of information between components. For a commercial system with the instruction architecture, functionality is experimentally partitioned into equal components, and functional scenarios carried out as thought experiments to determine the degree of information exchange required. The partitioning is adjusted to reduce information exchange, with some loss of component equality. The components are then partitioned into subcomponents and the process repeated at the more detailed level. Some information exchange issues identified at a detailed level require repartitioning at high level, and of course high level partitioning is sometimes guided by knowledge of what has been successfully implemented at a detailed level in similar systems.

The process for design of a system with the repetition architecture can be illustrated by considering how to design a system to resemble the behavior of a biological brain. The first step is to define the functionality of the system at the highest level, which could be to maintain homeostasis and create copies. The next step is to define the domains from which information will be derived (the senses, including senses directed at internal body state and position), and the range of behavioral options available (the body and muscles).

The next step is to establish the major functional partitioning in recommendation architecture terms. There are two equivalent ways of looking at the major separations. In information terms, the use of ambiguous information tends to result in proliferation of information, and clustering separated by competition reduces the volume of information which must be handled by the later clustering stages. In functional terms, different sets of information represent different action recommendations and the stages of competition reduce the alternatives in stages down to the specific action taken. The major separations proposed in Coward 1990 for the mammal brain are illustrated in figure 3. In the first clustering stage sensory input information is clustered to extract repetitions which are independent of the condition and orientation of the sensory systems. Examples could include object size and shape independent of distance and viewing angle. Combinations of activated clusters would from a viewpoint external to the system suggest the existence of such cognitive categories. A competitive function then performs an attention like function, followed by clustering to generate alternative behavioral alternatives with respect to the object at the focus of attention. The second competitive function selects one (or no) behavior, and a third categorization stage selects a portfolio combination of body movements. As discussed in section 16, a further competitive stage then integrates the combination of movements into a smooth behavior.

A major design decision at this point is whether to use heuristic or predefined functionality in the clustering separations, or some combination in which the major information routes between major clusters

are defined in advance and some experience based definition occurs within these constraints. The repetition hierarchy to detect sensory independent repetitions could be more predefined than the later recommendation stages.

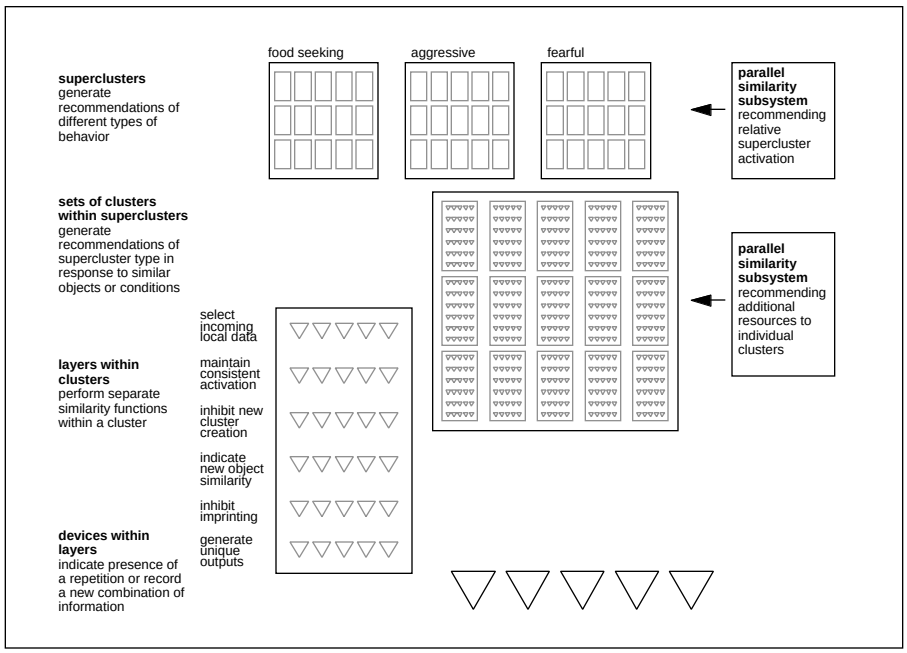

Figure 11. A repetition similarity clustering hierarchy. Separate subsystems can exist with their own clustering hierarchy generating recommendations for management actions on the primary hierarchy. Separate competition functions will manage the access of these recommendations to the primary hierarchy.

Consider now the functional partitioning within the alternative recommendation clustering function. As illustrated in figure 11, such a partitioning at the highest level could be into superclusters corresponding with different major types of behavior: aggressive, fearful, cooperative, food-seeking, sexual. The heuristic approach would be to sort the sequence of object and condition experiences into a hierarchy of clusters. Clusters at the simpler level could then provide input to multiple clusters at the more complex level. The separation into predefined superclusters is an example of the design decision referred to in the previous paragraph. It could be argued that only one supercluster was needed, the output from the same behavioral supercluster could be interpreted in different ways by the competitive phase. This is not a robust architectural choice because it does not allow the functions to be optimized independently. For example, suppose that the same combination of output information corresponding with the same action recommendation was sometimes followed by a favorable result, sometimes unfavorable. This condition could be specific to one type of action, and activation of the set of information (same recommendation, different result) could be an action recommendation to add additional information to the inputs of the cluster set generating the condition and recluster. Such a reclustering would only have value in the behavior where the contradiction was experienced. An example could be that attempting a friendly approach with the same individual produced conflicting results, recommending reclustering with additional information. If the additional information happened to include time of day a successful friendly approach in the morning might be separated from the same approach being unsuccessful in the evening.

A second value of the supercluster separation is that the relative strength of different recommendations with respect to the same objects or conditions can be modulated. Different information combinations derived largely from internal information could lower all the device thresholds in a target supercluster and increase the strength of the corresponding recommendations. For instance, extraction of information from indicators of blood sugar levels could modulate the relative arousal of the food-seeking supercluster. An implementation of such a subsystem is illustrated in figure 11. Another major subsystem illustrated in figure 11 recommends additional device resources for clusters. Because of the heuristic, experience

dependent nature of the functional clusters, the system cannot define in advance the numbers of device level repetitions which will be required to maintain the cluster. A separate subsystem is therefore required to detect information conditions indicating that particular clusters are not imprinting enough repetitions to generate output under conditions of indicated similarity, and to recommend assignment of additional devices. Because this subsystem assigns total system resources as required, in general an implicit time sequence of experience will be established in such a subsystem. This implicit time sequence is information which could itself be used for functional purposes.

When functionality is heuristically defined, the requirement to minimize the distribution of information must be addressed heuristically. The issue appears on various levels. At the device level, the requirement is that a device accepts a relatively limited number of inputs from other devices. Limited is relative, in a system with 10^9 plus components the absolute number of a limited proportion could be very large. At the cluster level, a simple architecture means that one cluster accepts inputs from a relatively limited number of other clusters. The requirement is that the limited proportion includes the most important functional information. The identity of this information cannot be fully known in advance given the heuristic definition of functionality, although predefinition of starting information is an important way in which the effectiveness of heuristic definition can be enhanced by external design knowledge. In a recommendation architecture, the strongest indication of functional importance is simultaneous occurrence. In other words, at the device level the information which is often present at the same time as the device itself or its peer devices in the same layer are active is the most likely to be valuable. Similar arguments apply at the cluster level and above. Hence to achieve a simple functional architecture under conditions of heuristic definition of functionality, a recommendation architecture requires an off line process involving fast rerun of past experience. This rerun establishes experimental connectivity between functional components which are frequently active at the same time, and disconnects connectivity if the functional components are rarely active at the same time.

The programming of repetitions close to system input has a particular set of requirements. An environmental sensor providing system input will typically measure the instantaneous value of a parameter at various points within the sensor. These values will correlate with the values of a parameter in one domain in the external environment. For example, an optical sensor such as a camera detects incoming light intensity at different points on a sensitive surface and outputs a signal which correlates with the intensity of light incoming from a particular solid angle relative to the camera. Very large numbers of measurements may be acquired at each instant in time. However, repetitions which correlate with conditions repeating in the external environment will be complex combinations of the relative value of many such measurements, often across a range of time. For example, suppose the camera outputs were to be used on a production line to locate, grasp and position a certain component. To be useful, repetitions must correlate with the position, distance, and direction of motion of objects in the external environment independent of the orientation of the camera. Such repetitions must be generated by complex algorithms acting on combinations of large numbers of raw inputs, including inputs from the camera over a range of time and inputs indicating the position of the camera relative to the grasping tool. The population of possible complex repetitions includes all possible subsets of all possible values of the raw inputs with all possible combination algorithms. Combination algorithms could be of the type strong input A and weak input B followed after time t by strong input C and weak input D. The population of possible combinations is extremely large. Because of this extreme size, a fully heuristic imprinting process would in general be impractical for finding combinations which correlate with sensor system independent repetitions in the external environment. Extraction of these repetitions must therefore be strongly directed by design, or in the case of biological systems discussed later, by natural selection leading to genetic programming. A heuristic process could tune this extraction process, for example to allow for the individual characteristics of the sensor in different systems. The resulting population of sensor independent repetitions could then be handled by a more fully heuristic process as described above.

14. Electronic Simulations of a Recommendation Architecture

Simulations have been performed to explore the heuristic definition of functionality as described in the previous sections. The simulation scenario assumed an environment containing five types of conditions A, B, C, D, and E. Conditions were experienced through a sensory and sensory preprocessing system that was assumed able to extract condition characteristics that were independent of the sensory system. This part of the system was not simulated, but conditions were created by random selection of a combination of twenty sensory characteristics from a set of one hundred. Different groups of conditions were defined by the

different statistical bias place on the selection process for each group. These characteristic probability distributions for each group are illustrated in figure 12.

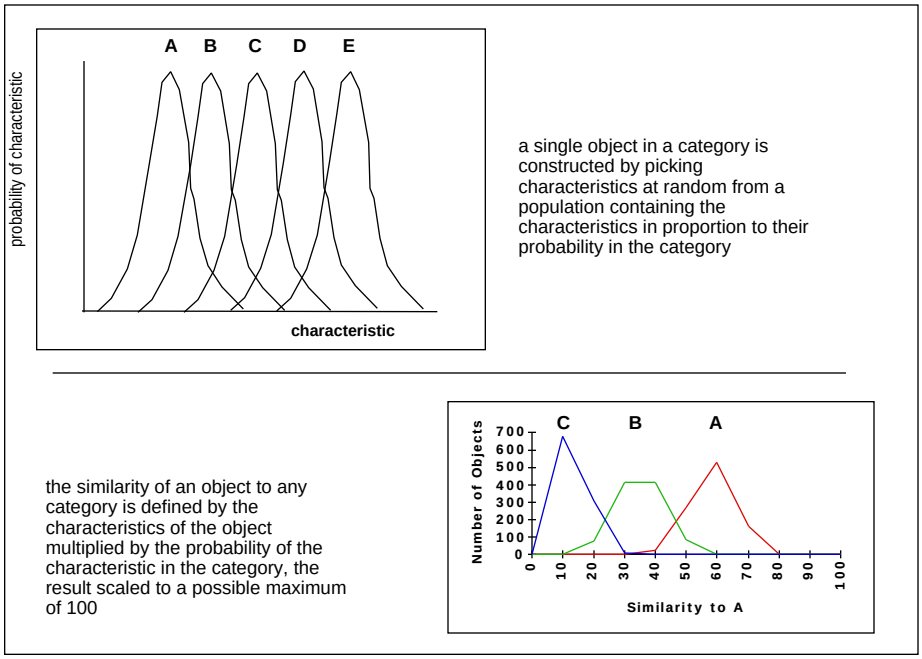

Figure 12. Definition of objects or conditions used in experiences of simulated system with recommendation architecture. An object is made up of a randomly selected combination of input characteristics. Different types of condition are defined by different statistical biases.

About one thousand conditions in each group were created. Duplicate characteristics were eliminated, resulting in some variation in number of characteristics around an average of fourteen. Duplicate conditions were discarded so that all condition experiences were different, but within a group there was a statistically defined similarity, and a statistically defined difference between conditions in different groups. A rough numerical measure of the similarity of a condition to the definition of one group could be defined as the sum of the probabilities of occurrence in the group of all the characteristics in the condition, normalized to some convenient scale. The similarities of one thousand conditions in groups A, B, and C to the definition of group A are also given in figure 12. About 1% of the conditions in a group had similarities on this measure equal to or greater than the similarities of the least similar members of that adjacent group. The ambiguous nature of the information as received by the simulated part of the system derived partly from the overlap of condition similarities between conditions in different groups and partly from the fact that no two condition experiences were the same in terms of the actual combination of sensory characteristics presented.

The simulated repetition similarity clustering function contained an unlimited number of potential similarity clusters. Each cluster contained three levels α, β, and γ as per figure 6. The $\varepsilon 2$ function was combined with α for the relatively simple system, and the $\varepsilon 1$ function was not needed because there was only one programmed repetition per device. All devices started as virgin and were configured by random selection of inputs, from the previous layer and in the case of the α layer from characteristics. The functional connections were simulated by assuming all devices in the layer were connected to the functional output and generated an averaged input to all virgin devices. For an unused module the α inputs were biased in favor of inputs which had frequently been present at the same time in the preceding experience period when no cluster produced significant α output, and the β and γ inputs were random without bias. In the simulations an experience phase in which a series of objects were presented to the system was followed by a sleep phase in which virgin devices were configured and reconfigured. In the experience phase, twenty-five conditions from each of several groups were presented, alternating one condition from each group. The condition order was randomly selected.

In the first experience phase no imprinting occurred, but in the subsequent sleep phase one module was configured with a bias towards α inputs which frequently occurred at the same time in the previous experience phase. In the next experience phase the first condition which contained a high proportion of the frequently simultaneous characteristics imprinted virgin devices until an output resulted. This imprinting process gradually lowered the thresholds of all virgin devices until the device fired, a minimum was reached, or a cluster output resulted. After the initial imprinting condition presentations did not trigger imprinting in the cluster unless the number of firing β devices exceeded a threshold proportion and the number of output γ devices was below a minimum level.

There are a number of variables which determine the performance of the system, including the number of virgin devices in each layer in a new cluster, the number of inputs to these devices, the proportion of activated devices in each layer which determines the presence of a functional output, and the number and input count of the virgin devices configured at each sleep cycle. These parameters were adjusted so that the system would on average generate five clusters in the course on experience. The actual number varied from four to nine depending on the detailed experience profile and statistical variations.

The population from which the inputs to virgin devices in a functioning cluster were selected was 50% an equal contribution from all devices in the previous level, 25% a contribution in proportion to how often those devices had contributed to firing a device in the peer level in the previous experience period, 12.5% a contribution from the experience period before that and so on. This population bias could be turned off and the results with fully random selection determined.

	run 1	run 3	run 4	run 5	run 6	run 7	run 8
	with dreaming statistical bias			**without statistical bias**			
							20% extra inputs
Early Experience Sequence	**ACE**	**ABDE**	**ABCDE**	**ACE**	**ABDE**	**ABCDE**	**ABDE**
	%	**%**	**%**	**%**	**%**	**%**	**%**
Output correlation	89.2	87.4	81.6	58.6	55.6	19.8	54.2
Similarity correlation	6.2	4.0	3.6	1.4	2.4	31.6	26.2
Total correlation	95.4	91.4	85.2	60.0	58.0	50.8	80.4

Figure 13. Results of simulations demonstrating correlation between heuristic clusters and cognitive categories. Also illustrated is the impact of the simulated sleep with dreaming function on effectiveness of correlation and resource usage.

The results of runs with and without the statistical bias are shown in figure 13. In those runs the experience profile was varied. In some runs the early experience only included conditions from groups A, C, and E. Later experience added D, then B. In other runs, conditions from groups A, B, D, and E were presented from the start and C was added later. Yet other runs included conditions from all five groups from the start. There is no separation between learning phase and operational phase in these simulations, and the system was constantly exposed to new conditions. Because limited imprinting resources are available, it sometimes took several experience/sleep cycles before the same conditions produced outputs, although most conditions produced an immediate output in a mature system. The runs reported in figure 13 were continued until for a final group of 150 conditions from each group about 90% produced output. The number of clusters varied from four to nine.

A strong but not perfect correlation was observed between groups and clusters as shown in figure 13. A rough correlation measure was established by first associating each cluster with a group. A cluster was associated with a group if in the final 150 presentations the cluster produced output in response to members of the group more frequently than in response to members of any other group. Several clusters could be associated with one group, but only one group with a cluster. The output correlation in figure 13 is the percentage of presentations in which the largest number of γ outputs came from a cluster associated with the group from which the presented object came. The similarity correlation is a comparable number but based on β activity when there was no γ output. Because β activity controls imprinting this is a good indication of where the strongest γ output will develop with further imprinting. These rough correlation numbers indicate that cluster outputs do provide an ambiguous indication of category membership.

The dramatic effect of the statistical bias of virgin device inputs can be seen from the figure. Providing virgin devices with a higher number of inputs could partially compensate for the lack of input bias, particularly for simpler experience profiles.

It might be possible to tune the various parameters to achieve very high correlations, but this would be to push the architecture towards unambiguous information both in terms of the cluster outputs and by requiring external cognitive knowledge of the groups. In the recommendation architecture the ambiguity of the information is handled through a competitive function to achieve high integrity of system actions. The simple competitive function was a simulated structure which received inputs from all γ devices in all clusters. Each input had a weight which could be any positive or negative value. The competitive function produced an output if the sum of the weights of all active inputs exceeded its current threshold. Such an output was interpreted as the system taking an action appropriate to members of one of the groups, and feedback was provided to the competitive function based on the actual group to which the current object belonged. This feedback could change the weights of all currently active inputs to the competitive function, but had no effect on the clusters. The clustering/competition separation required for the recommendation architecture was thus maintained.

The consequence of an action could be good, tolerable, fairly bad, or bad, with an associated numerical consequence of +1.0, +0.1. -0.5, or -1.0 respectively. Each of the five groups had a numerical consequence, typically one of each with two groups being bad. If the competitive function produced an output, the weights of all currently active inputs were changed by the following algorithm:

new weight = old weight + (0.1*consequence*(absolute value of old weight))

The threshold of the competitive function was varied with the objective of having the system take about ten actions in each wake period. An initial threshold value of 100 was adjusted in each sleep period to a primary threshold level at which ten actions would have been taken in the previous wake period, subject to a minimum primary threshold of 100. The current threshold at which the competitive function would produce an output was reduced below the primary threshold at any point in the wake cycle at which fewer actions had been taken than the expected proportion of the objective of ten expected for that point in the wake cycle given the proportion of experiences which has elapsed.

When a cluster was first imprinted, the weights of the inputs from the new γ devices were set at the primary threshold divided by the number of devices. First imprinting of a cluster would therefore always result in action. For a new cluster the initial weights were multiplied by the consequence, and this new weight was also the initial weight assigned to any additional inputs from γ devices imprinted in that cluster in the future.

This competitive function has some limitations which could be removed by more complex algorithms. For example, individual weights cannot change from positive to negative or vice versa, and cluster outputs will all be either one or the other depending on the action at first imprinting. Despite the limitations, the combined clustering and competitive functions achieved high integrity system actions.

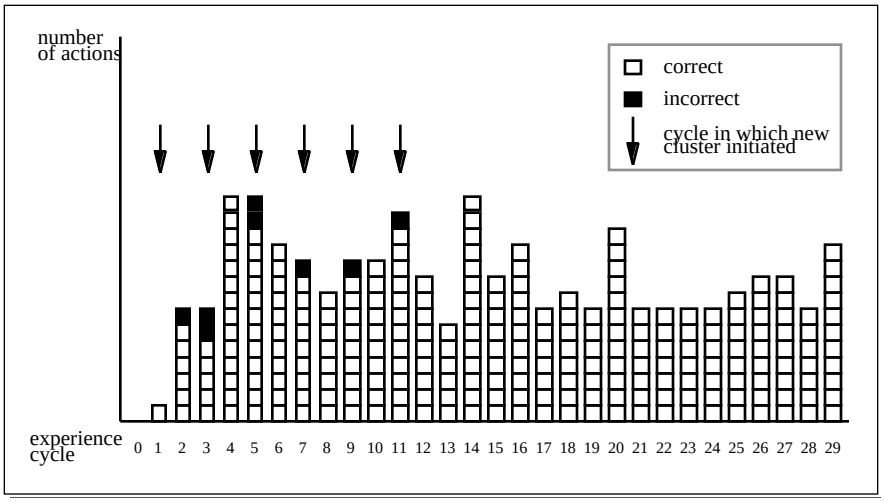

Figure 14. Simulation results demonstrating heuristic association between similarity clusters and behavior using a competitive function with feedback of the consequences of behavior. The feedback has no effect on the similarity clusters, this separation between clustering and competition is a critical architectural separation making a system performing complex functionality possible.

For the runs including competition, conditions from all groups were present from the start, twenty-five from each group in each experience period, and all condition experiences were different. The results of simulations of this type are shown in figure 14. Even in the early experiences while new clusters were being created, 93% of the actions taken were in response to appropriate conditions. Once enough clusters had been established to generate a similarity response to every condition, the system response accuracy increased to 100%. The same clusters could be used to generate appropriate behavior towards each of the five cognitive categories by appropriate consequence feedback. This was confirmed by the simulations. A simple version of the mechanism illustrated in figure 9 could be used to ensure that only one action was selected at one time. Multiple use of the same heuristic clustering for multiple action types has the disadvantage that the clustering cannot be functionally optimized for the different types of action. Such a functional optimization would be managed by a separate heuristic clustering which recommended reclustering in the primary action clusters. The information input to the separate management clusters would include information which correlated with similar outputs from the primary clusters producing contradictory consequences. The reclustering for best discrimination would differ for different action types. Note that this management process does not violate the clustering/competition separation.

The simulations demonstrate the ability of a system with the recommendation architecture to heuristically define its own functionality. The preconditions are definition of the information domains from which input will be taken and definition of a portfolio of possible actions, together with the ability to generate feedback of whether an action was followed by favorable or unfavorable consequences. This feedback is not information about appropriate behavioral targets, and any ambiguity in the feedback information could be handled by the competitive function through the averaging of weights over many experiences. It is not necessary to establish separate learning and operational phases for such a system, a new and sufficiently different condition would result in creation of new clusters which would be heuristically associated with appropriate behaviors.

The simulated system was exposed to a sequence of different conditions. It heuristically created a set of six to ten clusters that generated outputs in response to internally defined repetition conditions. These outputs are ambiguous information about the categories of conditions experienced by the system. Through a competitive function the system used these inputs to generate high integrity system actions with respect to the five condition categories. The system demonstrates true parallel processing capability: each component can perform its function in parallel on a self selected information space, and ambiguity is resolved to high integrity system actions.

15. "Memory" in a System with the Recommendation Architecture

There are a number of properties of a system with heuristically defined functionality that could be described as memory. Within the repetition similarity clustering functions, all repetitions that are active at the time an object or condition was experienced are permanently recorded. There are several consequences of this record. One is that there is an instantaneous trace created which means that the system response is different for an object which has been experienced before and a novel object, because a previously experienced object will require much less imprinting to generate an output. The system can therefore distinguish between familiar and unfamiliar objects. The second consequence is that if somehow the same population of recorded repetitions could be reactivated, the system condition would be identical with its condition during the original experience. A third consequence is that because outputs in response to an object or condition are generated across multiple clusters and superclusters, physical damage will not remove the ability of the system to distinguish between familiar and unfamiliar objects. Because some capability to generate behavioral recommendations is damaged, system behavior will be affected. The competitive process by which repetition clusters are associated with behaviors involves tuning of repetitions and no record of past states is recorded. A system will therefore not be able to access such past states, which can be interpreted as past conditions of system skills.

Another "memory" related property of the system derives from the ambiguous nature of the repetition information. Although the attention competitive function favors repetitions extracted from an object or

condition at the focus of attention, some repetitions from other objects will in general penetrate to the second repetition clustering. Such information could then be included in the more complex repetitions imprinted in the course of generating alternative recommendations with respect to the primary object. Hence information from objects which happened to be present at the same time could be included. Such cross-linking of information could lead to recommendations in response to secondary objects when only the primary object is present. Such recommendations would be experienced by the system as a weak reactivation of its state during experience of the secondary object, i.e. associative memory. However, unless a mechanism exists to amplify these activations the associative images would be very weak. A way in which such activations could be amplified in a human brain, generating a constant stream of activations independent of sensory input, and a functional value of such activations, was described in Coward 1990.

16. The Recommendation Architecture in Biology

Although biological brains have not been created by an intellect driven design process, they are subject to many of the same functional architecture constraints as commercial systems. There is a requirement to construct biological systems from DNA "blueprints". There is a requirement to be able to recover from construction errors and damage. There is biological advantage in multiple use of components, and to evolve it must be possible for small, random changes to occasionally result in significant functional changes that leave most functionality unaffected. These requirements generate large selection pressures in favor of simple functional architectures. Coward (1990) therefore argued that the vital role which information repetition plays in living systems has resulted in the ubiquitous appearance of the recommendation architecture in biological brains.

In any system with a functional architecture, physical structure will reflect functional separations, and activation states will reflect functional separations, and activation states will reflect functional component activations. For example, in an instruction system, memory and processing can typically be identified in physically separable structures, and smaller structures implement more detailed functional separations. Individual electronic systems can generate 40 million million device changes of state per second, and the resultant patterns of activation can only be understood given the perspective that any coherent activation is the operation of a functional component at some level of detail.

In biological brains, physical separations can be observed which resemble the functional separations of a recommendation architecture. At the device level, neurons respond to different combinations of repetitions of information represented by their inputs. Cortex columns correspond functionally with similarity clusters (Coward 1990, Tanaka 1993) and the layering in the cortex resembles the functional separation into layers required within such clusters. DeFilip (1997) has described strong cross connectivity of spiny stellate cells in layer IV of the cortex, and inhibitive back projection by bouquet cells close to output. This resembles the functional outputs from γ and β layers discussed earlier. The average cortex neuron has around 100 thousand inputs and fires in response to around 75 (Cauller 1997b), indicating that such devices must be programmed with many repetitions. The backward projections from higher cortex areas to lower that terminate in layer I, for example in the primary sensory cortex (Cauller 1995) resemble the feedback connectivity required to maintain repetition orthogonality. The dynamic self organization phenomena proposed by dynamical systems theory (e.g. Cauller 1997a) and observed by Bressler (1994) can more plausibly from a functional point of view be interpreted as a process of convergence on a consistent set of repetition activations.

The configuration of devices to be able to efficiently record functionally useful repetitions in future experience was proposed in Coward 1990 as the functional role of dream sleep in mammals. The prediction was made that it would be expected to take the form of a fast rerun of recent experience at the neuron level. There is recent evidence (Skaggs and McNaughton 1996) to confirm this view. The mapping of major physiological structures into the separations of the recommendation architecture is shown in figure 15. Some very suggestive evidence from the functional deficits resulting from damage and specific physiological problems is discussed in Coward 1990. The ability of a system to go through a series of small steps, each with a behavioral advantage, reaching functionality with a strong resemblance to human cognition is discussed in Coward 1997b and Coward 1999b.

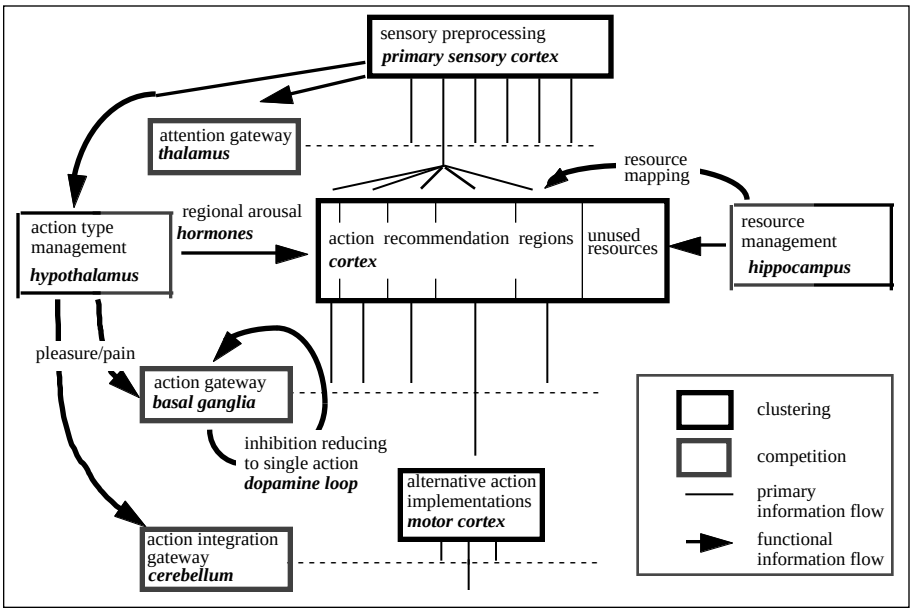

Figure 15. Major physiological structures in the brain mapped into clustering and competition subsystems following Coward 1990. Information generated by an initial clustering in the primary sensory cortex is competitively reduced through the thalamus structures. The remaining information is reclustered in the higher cortex to generate alternative action recommendations. These alternatives complete in the basal ganglia structures, and the information making up one alternative is allowed to proceed to the motor cortex where it is reclustered to specific muscle movements. Competition in the cerebellum results in an integrated physical movement. Parallel clustering subsystems manage major system management functions.

The proposal that biological brains exhibit the recommendation architecture, and specifically that mammal brains exhibit heuristic definition of functionality capabilities within that architecture, results in some specific physiological and psychological predictions originally indicated in Coward 1990. One prediction centres around the repetition imprinting mechanisms. The prediction is that in a column in the neocortex, neurons will exhibit a learning mechanism in which a neuron will fire in response to a new combination of inputs from the preceding layer. This firing in response to a new input combination will only occur in the presence of activity in an intermediate layer and absence of output from the column, but once it has occurred, the neuron will fire in the future in response to the same input combination without qualifications. In addition, column activity will exhibit an initial phase during which imprinting will not occur. Activity will only persist if there is strong feedback signals form higher cortex columns, and if activity persists then imprinting may occur if the layer activities are appropriate. A second prediction centres around the role of dream sleep. The prediction applies to a mammal which has been trained for different types of tasks, each with a set of conditions and appropriate responses, with overlap between conditions and responses for different types of task minimized. After a period in which such a mammal is deprived of dream sleep but with intensive activity during wake periods of only one type, it will tend to respond to conditions of the activity type which have some degree of novelty with behaviors of other types. This prediction relates to the need for appropriate imprinting resources in order to develop behavioral responses.

17. Conclusions

Functionally complex system design is based on separation of functionality into components which exchange information, with careful management of information context to ensure that information exchanged is always unambiguous to the receiving component. Design with this paradigm has been very successful but encounters difficulties in implementing parallel processing and extreme difficulty in implementing heuristically controlled functionality. The paradigm shift from exchange of unambiguous

information to exchange of ambiguous information between functional components makes it possible to design systems to perform complex combinations of functionality within an architecture which is qualitatively different from the instruction architecture ubiquitous in commercial systems. In particular, such systems can heuristically define their own functionality and naturally perform parallel processing. The inability of neural networks to scale to complex functional combinations is because of their failure to make a complete break from the unambiguous information paradigm. Simulations demonstrate that systems with the recommendation architecture can indeed heuristically define their own functionality, and the physical separations and functionality of biological brains suggest that the recommendation architecture is ubiquitous in such systems.

References

Andersen, J.A. and Sutton, J.P. (1997). If we compute faster, do we understand better? *Behavior Research Methods, Instruments & Computers* 29 (1), 67-77.

Avrunin, G.S., Corbett, J.C. and Dillon, L.K. (1998). Analysing Partially Implemented Real-Time Systems. *IEEE Transactions on Software Engineering* 24, 8, 602-614.

Bressler, S. L. (1994). Dynamic self-organization in the brain as observed by transient cortical coherence, in Pribam, K. (ed.) *Origins: brain and self-organization*, Hillsdale, NJ: Erlbaum.

Buschmann, F., Meunier, R., Rohnert, H. Sommerlad, P. and Stal, M. (1996). *Pattern Oriented Software Architecture: A System of Patterns*. New York: Wiley.

Card, H.C., Rosendahl, G.K., McNeill, D.K. and McLeod, R.D. (1998). Competitive Learning Algorithms and Neurocomputer Architecture. *IEEE Transactions on Computers* 47, 8, 847-858.

Carpenter, G.A. and Grossberg, S. (1988). The ART of Adaptive Pattern Recognition by a Self-Organizing Neural Network, *IEEE Computer*, 3, 77-88

Cauller, L. (1995). Layer I of primary sensory neocortex: Where top-down converges with bottom-up, *Behavioural Brain Research* 71: 163-170

Cauller, L. (1997a). NeuroInteractivism: Explaining Emergence without Representation, to be published.

Cauller, L. (1997b). Private communication.

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

Coward, L. A. (1997a). The Pattern Extraction Hierarchy Architecture: a Connectionist Alternative to the von Neumann Architecture, in Mira., J., Morenzo-Diaz, R., and Cabestanz, J. (eds.) *Biological and Artificial Computation: from Neuroscience to Technology*, 634-43, Berlin: Springer.

Coward, L.A. (1997b). Unguided Categorization, Direct and Symbolic Representation, and Evolution of Cognition in a Modified Connectionist Theory, in Riegler, A. and Peschl, M. (eds.), *Proceedings of the International Conference New Trends in Cognitive Science*, 139-146, Vienna: Austrian Society of Cognitive Science.

Coward, L.A. (1999a). The Application of Concepts from the Design of Extremely Complex Real-Time Electronic Systems to Cognitive Science, in Riegler, A. and Peschl, M. (eds.) *New Trends in Cognitive Science* , Plenum Press.

Coward, L.A. (1999b). A physiologically based theory of consciousness. To be published in Jordan, S. (ed.), *A Multidisciplinary Approach to Consciousness*.

De Felipe, J., (1997). Microcircuits in the Brain, in Mira., J., Morenzo-Diaz, R., and Cabestanz, J. (eds.) *Biological and Artificial Computation: from Neuroscience to Technology*, 195-206, Berlin: Springer.

Dijkstra, E.W. (1968). The Structure of the THE Multiprogramming System. *Communications of the ACM* 18, 8, 453-457.

Fernandez, J.L. (1993*). A Taxonomy of Coordination Mechanisms Used in Real-Time Software Based on Domain Analysis*. Carnegie-Mellon University Software Engineering Institute Technical Report 93-TR-34.

Garlan, D. and Shaw, M. (1993). An Introduction to Software Architecture. In Ambriola, V. and Tortora, G., eds., *Advances in Software Engineering and Knowledge Engineering vol. 1*. New Jersey: World Scientific Publishing Company.

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.

Harlow, T. M. (1868). Recovery from passage of an iron bar through the head, *New England Medical Society* 2, 327-46

Hebb, D. C. (1949). *The Organization of Behaviour*, New York: Wiley

Honkela, T, Pulkki, V. and Kohonen, T. (1995). Contextual Relations of Words in Grimm Tales, Analyzed by Self-Organizing Map, in Fogelman-Soulie, F. and Gallinari, P. (eds.), *ICANN-95 Proceedings of International Conference on Artificial Neural Networks*, 2, 3-7. Paris: EC2 et Cie.

Keck, D.O. and Kuehn, P.J. (1998). The Feature and Service Interaction Problem in Telecommunications Systems: a Survey. *IEEE Transactions on Software Engineering* 24, 10, 779-796.

Kruchten 1995, Ng, T. and Patel, V. (1994). Timely Failure Detection in a Large Distributed Real-Time System. *Proceedings of the International Workshop on Object Oriented Real-Time Dependable Systems*, Daria Point, California.

Llinas, R., Ribary, U., Joliot, M., and Wang, X.-J. (1994). Content and Context in Temporal Thalamocortical Binding, in Buzsaki G. et alii (eds), *Temporal Coding in the Brain*, Berlin: Springer.

Marr, D. (1982). *Vision*, New York: W.H. Freeman.

Mira, J., Herrero, J.C., and Delgado A.E. (1997). A Generic Formulation of Neural Nets as a Model of Parallel and Self Programming Computation, in Mira., J., Morenzo-Diaz, R., and Cabestanz, J. (eds.) *Biological and Artificial Computation: from Neuroscience to Technology*, 195-206, Springer: Berlin.

Parnas, D.L., Clements, P.C. and Weiss, D.M. (1985). The Modular Structure of Complex Systems, *IEEE Transactions on Software Engineering* SE-11, 3, 259-266.

Paul, C.J., Holloway, L.E., Yan, D., Strosnider, J.K., and Krogh, B.H. (1992). An Intelligent Reactive Monitoring and Scheduling System, *IEEE Control Systems* 12, 3, 78-86.

Perry, D.E. and Wolf, A.L. (1992). Foundations for the Study of Software Architecture, *ACM SIGSOFT Software Engineering Notes*, 17, 4, 40-52.

Ramos-Thuel, S. and Strosnider, J.K. (1994). Scheduling Fault Recovery Operations for Time-Critical Applications. *Proceedings of the Fourth IFIP Working Conference on Dependable Computing for Critical Applications*, San Diego.

Rosenblatt, F. (1961) *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*, Washington D.C.: Spartan

Sabisch, T., Ferguson, A. and Bolouri, H. (1997). Automatic registration of complex images using a self organizing neural system. *IEEE International Joint Conference on Neural Networks*, Anchorage, Alaska.

Sabisch, T., Ferguson, A. and Bolouri, H. (1998). Rotation, translation and scaling tolerant recognition of complex shapes using a hierarchical self organizing neural network. *Proceedings of International Conference on Neural Information Processing* 2, 1174-78. Berlin: Springer.

Sha, L. and Sathaye, S.S. (1993). *Distributed Real-Time System Design: Theoretical Concepts and Applications*. Carnegie-Mellon University Software Engineering Institute Technical Report 93-TR-2.

Skaggs, W. E. and McNaughton, B. L. (1996). Replay of neuronal firing sequences in rat hippocampus during sleep following spatial experience, *Science*, 271:1870-1873.

Soni, D., Nord, R.L. and Hofmeister, C. (1995). Software Architecture in Industrial Applications. *Proceedings of the 17th International Conference in Software Engineering*, 196-207. New York: ACM.

Strosnider, J.K. and Setcliff, D.E. (1997). Supporting Embedded System Design Capture, Analysis and Navigation. *Proceedings of the Australian Software Engineering Conference.*

Tanaka, K., (1993). Neuronal Mechanisms of Object Recognition, *Science*, 262, 685-88.

Taylor, J.G. and Alavi, F.N., (1993). Mathematical Analysis of a Competitive Network for Attention, in Taylor, J.G. (ed.), *Mathematical Approaches to Neural Networks*, Elsevier.

Yu, W.D. (1998). A Software Fault Prevention Approach in Coding and Root Cause Analysis. *Bell Labs Technical Journal* April-June 1998, 3-21.