The Reactive Accompanist: Applying Subsumption Architecture To Software Design

Joanna Bryson, Alan Smaill, Geraint Wiggins *

Abstract

Many knowledge-based systems suffer from structural problems such as inefficiency rooted in overinformedness and inability to cope with the unexpected or exceptional nature of real-world data. Behaviour-based architectures are better suited for such problems, but are not yet widely applied, possibly because design strategies are not yet well established. In robotics, *subsumption architecture* has proven an effective framework for developing such systems. In this paper we suggest the techniques of subsumption architecture can be transferred to other areas of Artificial Intelligence, and present a project implemented in this fashion. The development strategies used and the types of problems approachable by this method are also discussed.

1 Introduction

One of the main purposes of research in Artificial Intelligence is to apply progress made in understanding natural intelligence to improving the performance of intelligent artifacts. This paper discusses the general utility of a relatively recent model of cognition, the behaviour-based paradigm, within software engineering. It also addresses the issue of a methodology for applying this model. We suggest that subsumption architecture, as developed by Rodney Brooks for robotic control, can be utilised successfully in software development with relatively little modification. As evidence we present a software system developed under this strategy: a computer "folk musician" which accompanies unfamiliar melodies in real time without knowledge of music theory or any form of rule base.

The first section of this paper discusses the potential problems of centralised, knowledge-based systems. It goes on to discuss the rise of distributed intelligence

*Email: `joanna@ai.mit.edu`, `smaill@aisb.ed.ac.uk`, `geraint@aisb.ed.ac.uk`

and behaviour-based systems, their impact on the area of robotics, and the advantages of subsumption architecture in particular as a design methodology. The next section describes the Reactive Accompanist, and the way subsumption architecture was used in its development. The third section discusses what we learned about this methodology, the nature of problems for which we would suggest this new approach, and a few samples of such problems. The final section is a summary.

2 The Utility of Subsumption Architecture

2.1 Software Design and Centralised Planning

The development of quality software products and systems is seen by many as the main bottle neck in computing today.[Ratcliff 87] Often software developers are faced with problems that consist of a large number of perfectly tractable, even trivial, subparts; but are overwhelmed by the task of organising them into a coherent system that can be easily run and maintained. Once a system is built, on installation it is often found that exceptions and unanticipated requirements make the program unusable. This situation is often addressed by wedging changes into the original framework, complicating both the design and the code sometimes beyond recognition; or sometimes by the more costly method of returning to the design stage. In the worst case, entire projects are abandoned.[Brooks 82]

These problems are symptomatic of the traditional, knowledge-based design. As programmers, we are naturally inclined to think that every problem is best addressed by breaking it down into constituent, sequential steps towards the optimal solution. But in fact, an overly centralised approach can have the same effect computer system that it does on an economy. Huge inefficiencies are introduced by having large parts of the system over-informed, while reactions to the unpredictable events of the real world can be slowed to the points of annoyance or even uselessness. Many everyday situations are likely to be completely unresolvable within the existing framework.

It is particularly strange that the design of Artificial Intelligence systems often falls into this predicament. After all, part of the goal of AI is to understand and model our own, biological intelligence. Human and animal intelligence show little of the fragility associated with traditional computer programs — we tend to adapt and react rapidly to changing environments and opportunities[Chater 92]. From a biological perspective, the complexity and number of steps we ascribe to performing tasks are completely intractable within the limited neural systems of insects that perform them, or even within our own brains given the slow nature of the chemical reactions we use for communicating between neurons. And introspectively, we should admit how little of our daily life conforms to a carefully planned procedure — even when playing chess[Agre & Chapman 88].

2.2 Behaviour-Based Systems and Robotic Control

AI *has* produced other concepts of organisation. Connectionism has provided an alternate theory of distributed intelligence since the 1950's[Rosenblatt 58]. Then, in the 1980's, behaviour-based systems first began to be proposed as a possible mechanism for both biological and artificial intelligence[Minsky 85]. The behaviour-based paradigm explains intelligence in terms of small, simple, and coherent behaviours or *competences*. These competences operate independently, each with its own goal. People who follow this methodology believe that most of what we perceive as complex intelligent behaviour is in fact *emergent* from the interactions between these simple behaviours, and need never be explicitly planned, represented, or even recognised by the overall agent.

One branch of AI that behaviour-based systems have significantly affected is robotics. Robotics has been suffering since its inception from the difficulties described above as associated with centralised planning. Since the late 1980's, however, several laboratories have made considerable progress in overcoming such problems using behaviour-based strategies[Malcolm *et al* 89]. This is largely due to the impact by the research of Rodney Brooks and his development of *subsumption architecture*. Subsumption architecture, either in its original form or modified into "hybrids" with symbolic planners, has been used extensively in academic laboratories, and is now so widely accepted that it is being used by NASA for development of robots to send to Mars.[Brooks 91a]

Subsumption architecture is a robotic control methodology. It consists of two primary principles: that the control is decomposed into layers in turn composed of "task achieving behaviours" or *modules*, and that these layers are incrementally implemented and debugged in the "real world"[Brooks 91b]. The modules are solely simple augmented finite state machines. Important features of a subsumption architecture machines are that they are totally reactive, that is, any action is a consequence of a stimulus in the environment; and that they engage in no symbolic processing. Higher levels are able to observe the inputs and outputs of lower ones, and to subsume their behaviours. More detailed descriptions of subsumption architecture strategies are given in section 3.5.

2.3 The Advantages of Subsumption Architecture

We believe that many other problems currently approached with centralised, knowledge-based systems could be better addressed with behaviour-based control. The main difficulty is that, as a relatively new architecture, there are no established design strategies for creating such systems. We suggest that the methodology of subsumption architecture is well suited to providing guidelines for development of behaviour based systems for the following reasons:

- It has well defined principles and procedures for development[Brooks 91b].

- There are already many successful implementations documented[Maes 90].
- Its incremental, bottom-up approach almost necessitates that any solution developed with it will be
  - functional in the real world with real data, and
  - as minimally informed as possible,

To investigate the accuracy of these claims, we undertook a project to use subsumption architecture to implement a behaviour-based system which emulates a human cognitive capacity. The particular ability is that of a musician to provide chord accompaniment in real time to an unfamiliar piece of music.

3 The Reactive Accompanist

This section consists of a brief description of the functional objectives for the Accompanist, a brief review of related work, an overview of the decomposition used for approaching the task, and a description of the individual behaviours that compose the system. It concludes with a thorough discussion of subsumption architecture as it was used in this designing system.

3.1 The Task

The goal of this program is to derive chord structure from a melody in real time[1]. This mimics the ability of skilled musicians to accompany unfamiliar melodies on chord instruments, such as the piano or the guitar. The input melodies are played live on traditional instruments, and the processing is totally reactive, with no transcription of the melody nor rules of harmonisation applied.

3.2 Comparisons with Similar Work

There have been many applications of AI to the areas of musical transcription (e.g. [Longuet-Higgins 87]), or to tracking live musicians while following a score (e.g. [Matsushima *et al* 85],[Dannenberg & Mukaino 88]). These systems follow traditional, knowledge-based programming methodologies.

There have also been a large number of researchers applying straight connectionist tools to musical cognition (e.g. [Desain 92], or [Todd & Loy 91] for an overview). These projects tend to concentrate on deriving one aspect of the music, and to do it outside of real time.

[1] Further details of implementation, results, and comparisons with human subjects are available in [Bryson 92]

The musical AI work most closely related to the Accompanist is Tom Machover's Hyperinstruments[Machover 92] and Robert Rowe's Cypher[Rowe 92]. Both of these systems consist of two parts, one for interpreting input music, the other for playing either original or scored music along with it in real time. These projects use a Minskyesque behaviour-based approach in their listening halves which apparently works quite successfully. Unfortunately, it is difficult to compare either the performance or the ease of development of these systems with our own in order to demonstrate the effects of the subsumption strategy.

There do not seem to be any applications of subsumption architecture to music, nor to any other area outside of robotics at this point. One project that is closely related is Agre and Chapman's Pengi[Agre & Chapman 88] — a reactive video-game playing robot. In their paper, "What are Plans For?," they emphasise that the utility of plans is not as procedures to follow, but as methods of communication, either within an agent or between agents. Actual activity is based on reactive, opportunistic behaviours. Although their implementation was not strictly subsumption architecture, it was heavily influenced by Brooks' work.

Some researchers believe that strict subsumption architecture can only be used for tasks that would be reflexive in humans, while traditional symbolic programming is necessary for the higher cognitive functions[Arkin 90]. It may be that such opinions over-emphasise the use of symbolic processing in human intelligent behaviour[Kirsh 91]. For further information on related theories of human behaviour and cognition, see "The Society of Mind"[Minsky 85], "Consciousness Explained"[Dennett 91], and "Intelligence without Reason"[Brooks 91a].

3.3 Project Description

3.3.1 Organisation

A subsumption architecture program is considered to be organised in terms of *layers*. These layers consist of one or more *modules* or units of behaviour. In designing the system, the task is first broken into layers, then the layers implemented one at a time, starting with the most basic and working up. At any point of development, only the modules for the current layer are being designed and tested.

In our implementation, we chose to make Pitch recognition the most basic level of competence. The next layer is Chord recognition, which transforms the output from pitches into chords. The following level is Time recognition. Although the basic competence of this level is to recognise the beats, the output of the program at this level is not just rhythmic information, but the modified output of the chord layer. Now instead of a single chord, it outputs chords in a timed sequence.

Should this project be continued, the next layer would try to recognise Structure

SONG
STRUCTURE
TIME
CHORD
PITCH

Figure 1: Basic Layers of a Complete Machine.
Bold face script indicates implementation.

in the piece. That is, it would bias chord choice towards recurring patterns[2]. The final level of competence would be Song recognition, whereby the program would learn to recognise song types. This would subsume Structure in its ability to anticipate the number of times a pattern would repeat, and what the next pattern is likely to be.

3.4 The Individual Behaviours

The system as it now stands consists of six independent modules, each of which embodies a behaviour or "competence". For a diagram of the interaction of the implemented modules and the anticipated layers, see figure 3.4. The purpose of this section is to illustrate to some extent the way the behaviours interact, and also the diversity of the strategies that are used in the actual workings of the components. The modules are listed in the order they were developed and implemented.

1. NOTE: derives pitch class from the auditory input. This module consists primarily of a competitive neural network[J. Hertz & Palmer 91]. This network transforms the input melody[3] into a weighted array representing the results of all the output units. The output units of the net correspond to the pitch classes the net has been trained on (currently but not necessarily the traditional 12 tone western scale).

2. CHORD: identifies a chord from an input of pitch class information. It uses a hybrid associative and competitive neural net, the weights of which are determined by predefined chord types. The inputs are of the type output by the NOTE module, and the output is a similar weighted array of potential chord types.

[2]Rhythmic pattern bias is already implemented in the Time layer by the TIMED module.

[3]Microphone input run through the sound board of a SUN SPARCstation, converted by csound's pvanal utility[CSO91]into frequency/gain pairs per incremental time frame using Fast Fourier Transformations.

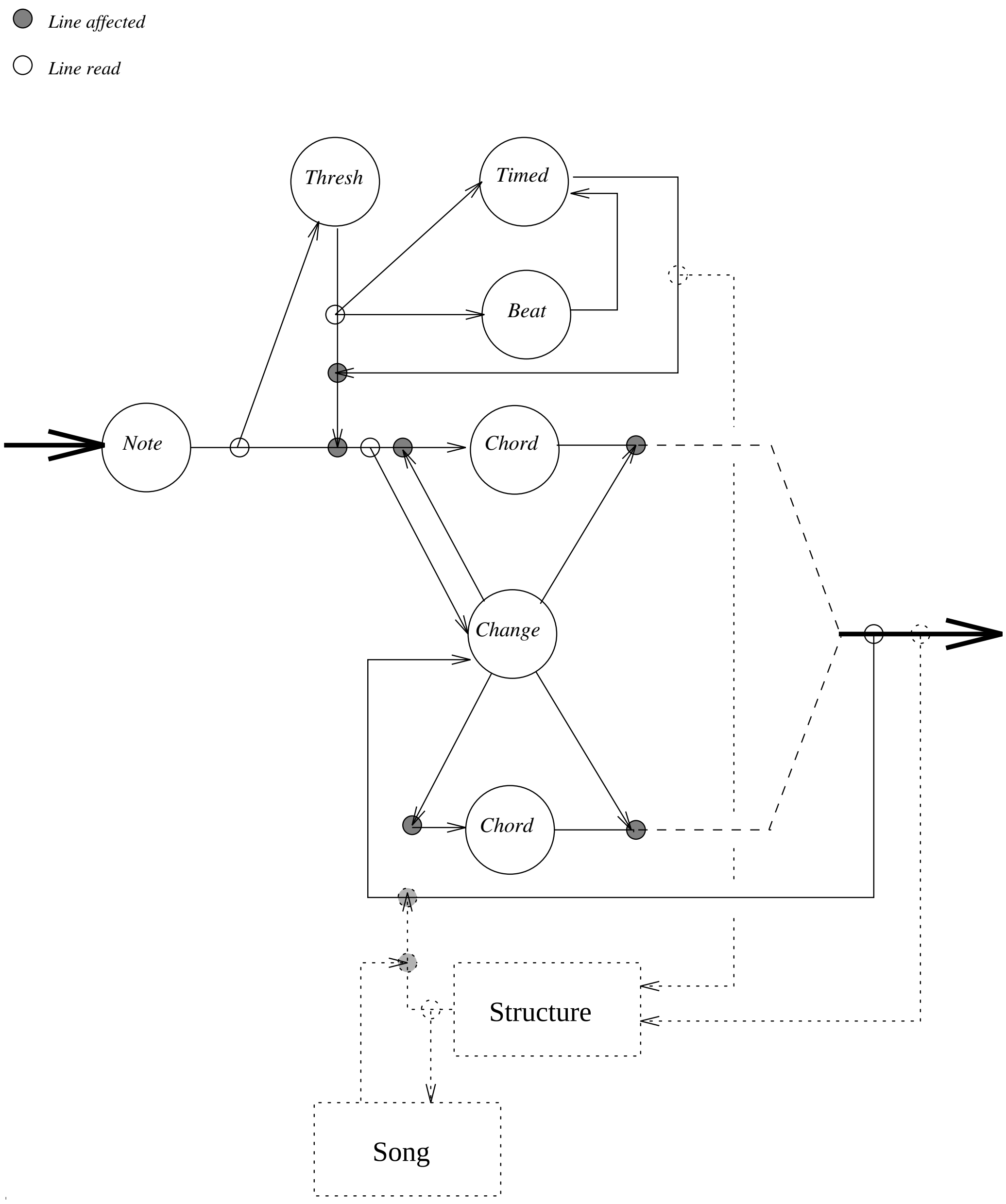

Figure 2: The Interactions of the Components of the Reactive Accompanist. The rectangles represent layers, the circles modules.

3. Thresh: finds the threshold in time for a new note. This is a simple finite state machine reacting to the output of note and outputs a True/False value.

4. Beat: finds the most likely times for an occurrence of a thresh event. It is a single layer neural network which takes as input boolean information like that of the thresh module. It creates an array of time intervals between positive signals, with associated weights for how often they occur The existing weights decay over time, so that time changes are tracked. The output is the weights and indices of the net.

5. Change: uses rhythmic information and chord to find likely locations for chord changes. If chord is run without interference, it soon stabilises to reporting the key for the melody. The change module modifies the inputs for multiple incarnations of chord representing the possibility that chords have changed at various time events. Change's output is the output of the chord module that seems most strengthened by the current trend in the melody.

6. Timed: uses information from beat and thresh to anticipate the next rhythmic event. It consists of a finite state machine. Timed uses the beat information in order to build and maintain a guess at what the dominating time intervals are, and the thresh input to try to determine when these events occur. Its output is a boolean stream similar to the output of thresh, but the True values represent the actual anticipated beats rather than the detection of a random note threshold. The output of change is improved when timed subsumes thresh as its source of rhythmic information.

Notice that none of these modules actually produces sound, rather their output is a simple transformation of their input. An executable program called **noise** was constructed which translated the output of the change module into a soundfile. This output can be either the real time output of change, or the final chord determined by change at the reaching of a new beat threshold, retroactively applied across the entire previous beat. This latter information is *not* real time, but it is the information that would be observed by higher level pattern seeking programs, and is also most useful to humans evaluating the program.

3.4.1 Results

This program was evaluated in two ways — by comparing its output to the chords derived for the same input by several musicians, and by having musicians judge whether the chord structure of the piece "sounds reasonable". These evaluations are of necessity qualitative and subjective, for one thing even the strict harmonisation rules of music theory are nondeterministic. However, within the domain of the melodic input (Celtic folk music), certain chord conventions are well standardised, and the musicians used could be considered fairly expert in recognising them.

The Reactive Accompanist tends to use the correct chords slightly more than half the time, and to change chords away from the dominant at the same points that human musicians do. "Incorrect" chords are seldom totally dissonant, but rather are *unexpected* because they fail to fit into a pattern. This problem is a consequence of the fact that the program is only informed with respect to the notes currently played and the notes in the previous chord, and would be corrected by the higher levels of Structure and Song. Extra cues, such as polyphonic input, also help with this problem.

The program has difficulty with some instrumentations, apparently because they are too different from the instrument used to train the neural net in the NOTE module. The trained instrument, the mandolin, and the penny whistle are the best. Performance degrades over the Scottish small pipes and the recorder, and is useless for the human voice. Also, slow moving songs with few notes per beat often do not provide sufficient cues for accurate chord choice. However, given favourable input, each of the modules performs its task, though not with complete reliability or "correctness", to a sufficient standard that a) musicians consider its output to be reasonable and intelligent and b) that the modules above it can use its output to operate to the same standard.

3.5 Subsumption Architecture and Development

The development of this system took place within the framework of several criteria for subsumption architecture derived from [Brooks 91b]. Below are the summaries of these criteria as we derived them from Brooks' paper. These were used as guidelines for this project. After each summary is a description of the extent to which the project conformed to this goal, and then an analysis of how this factor affected the actual project.

3.5.1 Components

A layer should be a fixed topology network of finite state machines possibly augmented with a few registers (for data storage) and timers. It should be data driven, with no global data and no dynamic communication. "Data driven" means that the system is *reactive*. Activity by a competence is a consequence of events or opportunities in the environment, not of instructions from another competence. "No dynamic communication" means that layers do not engage in two way communication — when a message is sent, there is no indication whether it has been received.

The modules of this project are fairly well "augmented finite state machines" as specified, although their data objects are more complex than "a few registers" would imply. On a functional level, however, even the most complex of these objects is essentially a two dimensional array of floating point numbers. The modules are all data driven - their internal states are only affected by their input. And there is no global data — data objects are all associated with modules and can only be altered by their own module.

Conformity to this set of criteria results in a program constructed of simple, self-contained, independent, reactive units. It helps ensure robustness in the program, in that if one unit malfunctions, the others will carry on, though the overall performance will be degraded. This was evident in the performance of the accompanist. For example, when interpreting the Scottish small pipes, the notes from the continuous underlying drone pipes confused the THRESH module to the point that rhythmic events were almost unreported. The program simply plays along with the drone until towards the end of the piece, when it has enough rhythm information that it catches a few of the chord changes human guitarists made in accompaniment.

3.5.2 Interaction

> Layers may only interact with other (lower) layers via their input and output wires. A higher layer may suppress the input of a lower layer, and it may inhibit the output. "Suppression" of the input may include replacing it, but output may not be altered, only left unaffected or discarded.

Although, wires are not involved here, this principle was carefully followed in concept. It might appear that the CHANGE module has more influence over CHORD than this principle would dictate, because CHANGE can cause multiple instantiations of CHORD to be created. This is a slight modification of subsumption architecture, (and not something likely to occur in a robot). However, once the module is created it has no special dependencies or status; it is never stopped or restarted or in any other way treated like a called function.

This principle is one of the strengths of subsumption architecture as a development tool. Thinking of previously developed modules only in terms of fixed input and output makes it much clearer how they should be interfaced with. Thinking of a current module in terms of the input and output that higher modules will be able to see helps clarify both the requirements of the module, and whether in fact the module should be reclassified into smaller or larger competences. At the outset of the project, we anticipated that creating a strict hierarchy for a music system would be difficult. In fact, this never became an issue, though this may be a feature of our particular decomposition.

3.5.3 Test Environments

There should be no simplified test environments. Subsumption applications should cope for themselves with all the noise of inaccurate sensors and all the unpredictability of the real world. If the input becomes increasingly further from the ideal, the performance of the agent should gradually degrade, rather than completely failing.

For our project we conformed to this principle completely. The project was run on live input from acoustic instruments recorded through the SUN soundtool utility. The Accompanist has to cope with all the errors of time, note, and intonation and also microphone distortion. Even when these factors caused considerable garbling of the information, so that a human might be able not recognise the accompanist's conception of the melody, the actual output of the program is only partly degraded. There are, however, some inputs, for example human singing, which the program can not comprehend at all.

The benefit this has on the development process is that once a module works at all, it "really" works. There is no question of how performance will change or what exceptions will be met when the program is exposed to the real world. However, the combination of imperfect data and expecting some performance degradation as a consequence made the issue of fine tuning a module's performance difficult, as it made it was unclear when an optimal level had been reached. (See the next section.)

3.5.4 Independence in Engineering

Each layer should be engineered separately, then tested and debugged until flawless before proceeding. Debugging a multilayer system is difficult; this approach should reduce the potential problem sources to being either in the current layer or the interface between the current layer and the previous one.

The approach used for this project was to implement the modules as separate classes in an object-oriented language (C++). Each module was developed, implemented, and tested in its own executable program. There were some changes made to lower levels after development had moved on to higher levels, but these mostly took the form of parameter adjustments. Fine tuning of a layer was found to be difficult without the layers that would be utilising its output. (See previous section on No Test Environments.)

This incremental, bottom-up approach is useful for ensuring the workability of the program design and simplifying the debugging process. It also makes the program less likely to be over-informed, as one can determine the earliest point at which a level is functional, although of course extraneous modules could already be present. Altering lower levels is not so much of an issue in a software project, because the executables associated with those modules can be recompiled and retested incrementally again, unlike on a robot where regression is more difficult. Therefore violating this part of this principle did not actually have the negative effects on debugging it would have in robotics.

3.5.5 Representation

In [Brooks 91b], Brooks not only advocates against central representation, but says there should be no variables instantiated, no rules matched, and no choices made on any level.

The issue of "no representation" has been controversial. For example, in the Accompanist, aren't the chords a symbol system? However, from the three criteria above, it should be clear that the issue here is no representation of *knowledge* from the perspective of the program, that is, no symbolic processing. For example, in [Brooks 91a] he discusses a navigation module that builds a structure that from the external perspective is essentially a map, but since it is this structure *itself* that builds, maintains, and causes the robot to follow the learned trajectories, all as a consequence of the reactive behaviour of the network, this does not violate his representational principles.

This project generally followed this principle well, and should fully follow it in the final version. One violation was made in the implementation of TIMED, whereby the module looked for only one time segment, the one that represented the beat. This could easily be considered variable instantiation, and did in fact result in a loss of robustness, since this module is particularly easy to fool when melodic input has exceptional speeds. This was a hack partly necessitated by the need to cap the project without the higher levels. Our intention on continuing the project is to have multiple possible times and associated chord changes as the output of the Time level, just as multiple notes and chords are the outputs of the Pitch and Chord levels. The next level will weight the possibilities with respect to their repeatability in a form of pattern matching. We intend to implement this also in terms of non-symbolic structures, using strategies much like those used in the navigating robots described in [Mataric 90] and [Smart 92].

4 Subsumption Architecture and Software Design

4.1 Lessons Derived from the Accompanist Experiment

The significance of the success of the Reactive Accompanist is that we have produced a system that emulates what is normally considered a higher cognitive function in humans in a totally reactive framework. But our primary aim project was to explore the use of subsumption architecture within a software development context. This section describes the successes and drawbacks of the approach, and modifications we made to it.

We had entered this project expecting to run into difficulty with subsumption architecture in two ways. First, that the nature of music might be too interdependent for an approach that is so strictly hierarchical, and second that traditional knowledge-based techniques would be needed at some level, presumably when pattern recognition became an issue. As explained in section 3.5, both of these concerns proved unjustified. Notice that our conclusion with regard to the lack of necessity for a symbolic hybrid is based only partly on the Accompanist, which at this point has only rhythmic pattern capabilities, but also on a review of other subsumption projects.

The largest problem we *did* encounter with the project was solid evaluations of the performance of the individual modules, given that they were tested on real-world data. An "ideal" result was consequently both difficult to predict and not expected to occur. This resulted in multi-module parameter tweaking when higher levels showed weaknesses in the system. Also, the methods of decomposition of the problem into levels and modules is still nebulous. In general, our difficulties were not with the hierarchical order, but with the definitions of the units.

Our general practice for decomposition was to err in the direction of making the units too large, and then redivide the elements if necessary. For modular decomposition the rule was, if a module is becoming more complex than the requirements of subsumption architecture indicate, the task it addresses needs to be broken into more modules. For levels, the guideline concerning incremental development seems best applied in reverse — modules that belong in the same level will probably need to be tested and developed together.

4.2 The Nature of Tasks Suited to this Approach

Subsumption architecture was developed as part of an effort to replicate animal intelligence[Brooks 89]. We, as animals, are constantly evaluating and manipulating extremely complex environments with only simple high-level goals [Agre & Chapman 88]. A subsumption-based machine achieves this by having multiple simple behaviours operating in parallel. Low level behaviours constantly sense and react to the environment, while higher level modules pursue the primary goals. All behaviours are constantly receptive to the inputs that trigger their actions, making it possible for the machine to pursue multiple goals simultaneously and opportunistically.[Brooks 91b]

Considering these attributes, we can make a list of some of the characteristics systems that could best be approached using subsumption architecture might have:

- the system is required to operate in real time,
- it needs to pursue multiple goals simultaneously,
- it exploits a parallel architecture,
- it needs to be aware of real time events external to it,
- it needs to process sensor data,
- there are conditions involving details not important to the system's central goals that need to be met and maintained in order for those goals to be pursued.

4.3 Suggested Applications

This list is included as examples of some of the areas we believe could be approached and possibly advanced by application of a behaviour-based system through a subsumption architecture design. Obviously, the potential for further research is extensive.

- *Monitoring systems.* This includes industrial, financial, and medical applications. Medical monitoring devices are actually duplicating the exact kind of reflexive biological behaviours that subsumption architecture robots model so well. This functionality can be extrapolated into other procedurally related areas.
- *Computer Operating Systems.* It is not much more of an abstraction to move from a monitoring system to a scheduler. The only difference here is that the environment with which the agent will be interacting is entirely contained in the computer. This would be particularly interesting in a multi-cpu system, where the parallel nature of subsumption architecture could be more fully exploited.
- *Face recognition* This is actually similar to the task of the reactive accompanist — interpreting perceptual information in a way similar to the way humans do. An advantage in this case is that extensive developmental psychology research has been done on the way infants learn to recognise faces, and this work can be used to order the implementation of levels of competence.
- *Oral Communication* It was the similarities between the difficulties experienced in natural language research and classical robotics research that provided the original impetus for the reactive accompanist project. As with face recognition, research in language development could provide significant

cues for the proper behaviour-based approach. One could build small projects like the Accompanist to explore specific subproblems such as deriving emotional content from inflection. Semantics is particularly closely linked to subsumption architecture in its need for grounding and minimally informed higher levels.

5 Summary

Behaviour-based systems developed under subsumption architecture can be used to address many problems that are currently met only with traditional, knowledge-based systems. The advantages of systems designed under this approach is that they are able to work effectively with real-world data, and at the same time the complexities of the real world are handled by low-level competences, which allows the higher, goal-pursuing levels to be kept simple and efficient. We have created such a system ourselves, and have shown it is possible to transfer the techniques of subsumption architecture into a new and fertile domain. We look forward to seeing both new applications and new methodologies developed within the reactive, behaviour-based paradigm.

6 Acknowledgements

We would like to thank the following individuals for their assistance with this paper: Gill Kendon, Phil Kime, and Ashley Walker. Special thanks to William Smart who also contributed background research and his expertise in subsumption architecture.

I [Joanna] would like to thank the many people besides the above who helped the MSc project which resulted in the Reactive Accompanist; especially Ian Porter, Sheila Tuli, and David Willshaw, and above all my supervisors Alan and Geraint.

References

[Agre & Chapman 88] Philip E. Agre and David Chapman. What are plans for? AI memo 1050, MIT, Cambridge, Massachusetts, September 1988.

[Arkin 90] Ronald Arkin. Integrating behavioural, perceptual and world knowledge in re active navigation. *Robotics and Automation*, 6(1):105–122, 1990.

[Brooks 82] Frederick P. Brooks, Jr. *The Mythical Man-month.* Addison-Wesley, Reading, Massachusetts, 1982.

[Brooks 89] R. A. Brooks. A robot that walks : Emergent behaviours from a carefully evolved network. A.I. Memo 1091, MIT, Cambridge, Massachusetts, February 1989.

[Brooks 91a] R. A. Brooks. Intelligence without reason. A.I. Memo 1293, MIT, Cambridge, Massachusetts, April 1991.

[Brooks 91b] R. A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47:139–159, 1991.

[Bryson 92] Joanna Bryson. The subsumption strategy development of a music modelling syst em. Unpublished M.Sc. thesis, Department of Artificial Intelligence, 1992.

[Chater 92] Nick Chater. Neural networks: Motivation from psychology. Lecture notes from the Neural Networks MSc module, Department of Cognitive Science, 1992.

[CSO91] *The Csound Reference Manual.* Cambridge, MA, 1991.

[Dannenberg & Mukaino 88] R. Dannenberg and H. Mukaino. New techniques for enhanced quality of computer accompaniment. In *Proceedings of the ICMC*, pages 241–249, 1988.

[Dennett 91] Daniel C. Dennett. *Consciousness Explained.* Allan Lane, The Penguin Press, London, UK, 1991.

[Desain 92] P. Desain. A (de)composable theory of rhythm perception. to appear in Music Perception, 1992.

[J. Hertz & Palmer 91] A. Krogh J. Hertz and R. G. Palmer. *Introduction to the Theory of Neural Computation.* Addison-Wesley Publishing Company, Redwood City, CA, 1991.

[Kirsh 91] David Kirsh. Today the earwig, tomorrow man? *Artificial Intelligence*, (47):161–184, 1991.

[Longuet-Higgins 87] H. C. Longuet-Higgins. The perception of music. In M. Boden, editor, *Mental Processes : Studies in Cognitive Science*, chapter 13, pages 169–187. MIT Press, Cambridge, Massachusetts, 1987.

[Machover 92] T. Machover. Hyperinstruments : A progress report 1987-1991. Media lab document, MIT, Cambridge, Massachusetts, 1992.

[Maes 90] P. Maes. *Designing Autonomous Agents : Theory and Practice from Biology to Engineering and back.* MIT Press, Cambridge, Massachusetts, 1990.

[Malcolm *et al* 89] C. Malcolm, T. Smithers, and J. Hallam. An emerging paradigm in robot architecture. In *Proceedings of IAS*, volume 2, Amsterdam, Netherlands, 1989.

[Mataric 90] M. J. Mataric. A distributed model for mobile robot environment-learning and navigation. Lab report, MIT, Cambridge, Massachusetts, May 1990.

[Matsushima *et al* 85] T. Matsushima, T. Harada, I. Sonomoto, K. Kanamori, A. Uesugi, Y. Nimura, S. Hashimoto, and S. Ohteru. Automated recognition system for musical score – the vision system of WABOT-2. *Bulletin of Science and Engineering Research Laboratory*, 112:25–52, 1985.

[Minsky 85] M. Minsky. *The Society of Mind.* Simon and Schuster Inc., New York, NY, 1985.

[Ratcliff 87] B. Ratcliff. *Software Engineering: Principles and Methods.* Blackwell Scientific Publications, Oxford, UK, 1987.

[Rosenblatt 58] F Rosenblatt. The perceptron: A probabilistic model for information storage and organisation in the brain. *Psychological Review*, 65:386–408, 1958.

[Rowe 92] Robert Rowe. Machine listening and composing with cypher. *The Computer Music Journal*, 16(1), 1992.

[Smart 92] William D. Smart. Location recognition wih neural networks in a mobile robot. Unpublished M.Sc. thesis, Department of Artificial Intelligence, 1992.

[Todd & Loy 91] P. M. Todd and D. G. Loy, editors. *Music and Connectionism.* MIT Press, Cambridge, MA, 1991. Based on two special issues of the Computer Music Journal.