Communicating Process Architectures 2009
P.H. Welch et al. (Eds.)
IOS Press, 2009

Toward Process Architectures for Behavioural Robotics

Jonathan SIMPSON and Carl G. RITSON

School of Computing, University of Kent,
Canterbury, Kent, CT2 7NZ, England.

{J.Simpson , C.G.Ritson} @kent.ac.uk

Abstract. Building robot control programs which function as intended is a challenging task. Roboticists have developed architectures to provide principles, constraints and primitives which simplify the building of these correct, well structured systems. A number of established and prevalent behavioural architectures for robot control make use of explicit parallelism with message passing. Expressing these architectures in terms of a process-oriented programming language, such as occam-π, allows us to distil design rules, structures and primitives for use in the development of process architectures for robot control.

Keywords. Concurrency, Robotics, Behavioural Control

Introduction

Robotics is inherently a concurrent problem: watching many sensors, and driving many effectors to sense and interact with the world. The ability to write control programs using parallelism is desirable, as it allows us to maintain the concurrent aspects of robot behaviour, without having to serialise the tasks required. It is natural to think about control tasks in the real world in parallel, and in unwinding these thoughts to a sequential control loop, mental overhead is added.

Control tasks within robot systems fall into one of two categories: reactive and deliberative. Reactive control involves the tight coupling of sensing and action to provide fast responses where action is critical and is somewhat analogous to the reflexes present in humans. Deliberative control involves tasks which require more extensive computation or sensing before action can take place, generally involving plans and state e.g. generating and maintaining a path to a given way-point, or feature identification of the world using camera data.

Most robot control programs are 'hybrid' systems, with reactive and deliberative behaviour layers along with a third layer performing arbitration and co-ordination [1]. This combination of reactive and deliberative components allows for both high-level planning and fast reaction to important stimuli from the environment. For example, a robot might be equipped with a deliberative path planner, which uses a pre-supplied map of its environment to calculate a safe path to the destination. This planner would work alongside a reactive component which works to avoid obstacles encountered trying to follow way points generated by the path planner. The third layer would act to ensure that both the path planner and the obstacle avoidance behaviour can influence the motion of the robot. Hybrid robot control systems employ parallel computation to allow both the deliberative and reactive components to be active at the same time. In addition, the reactive component may be internally parallel, as the objective of reactive systems is keeping sensing and action close together, a model that relies on critical information in the world not being missed.

Behavioural robotic control applies behaviour-based AI to robot control; using modular decomposition of the system's intelligence to structure the behaviours which compose its controls. Behaviour-based AI is heavily inspired by the agent-based decomposition of human intelligence presented in Minsky's Society of Mind [2]. Architectures for behavioural control act to support the development of the third, interfacing layer of systems, often providing support for parallelism and communication between components. When writing a control program using a language with explicit parallelism and message passing, these evolved and widely used behavioural architectures provide a natural source of ideas for process architectures.

We are specifically interested in the development of scalable process architectures for robot control using the occam-π programming language [3]. Using occam-π allows us to harness the Transterpreter, a portable virtual-machine developed expressly for running occam-π programs on small embedded platforms [4]. We have used occam-π for robot control successfully on a variety of mobile robot platforms in the past, including the LEGO MindStorms RCX [5], Surveyor SRV-1 [6] and Pioneer 3-DX [7], with additional support now added for the LynxMotion AH3-R 'Hexapod' walking robot [8]. In light of this growing availability of process-orientation on robotics platforms, we consider it valuable to explore traditional behavioural architectures in this context. The techniques developed for use in occam-π may also be applicable to robotic control in other process-oriented languages or language extensions such as JCSP [9], PyCSP [10] and gCSP [11].

In this paper we will elucidate, implement primitives for and compare a number of seminal behavioural control architectures in the context of their application in process architectures for robot control. Section 2 examines Brooks' Subsumption Architecture, with following sections 3, 4 and 5 dealing with the Connell's Colony Architecture, Maes' Action-Selection and Arkin's Motor Schemas respectively. Finally, in section 6 we will examine Rosenblatt's Distributed Architecture for Mobile Control (DAMN). After looking at each architecture in turn we conclude in section 7 and identify future extensions to this work in section 8.

1. Platforms

The implementation examples in this paper make reference to two robot platforms, the Pioneer 3-DX from Mobile Robots, Inc [12] and the AH-3R 'Hexapod' walking robot from LynxMotion [8].

The Pioneer 3-DX is a two wheeled robot mobile platform fitted with a ring of 16 sonar sensors covering its circumference. A SICK laser range-finder is also fitted, covering a forward 180 degree arc with a scan distance of eight meters. On-board computation is provided by a 700MHz PC104 board running Debian GNU/Linux.

The LynxMotion AH3-R 'Hexapod' is, as its name would suggest, a six legged walking robot. Each leg is roughly 60 degrees apart and driven by three servos, giving it three degrees of freedom: swinging forward or backward, raising or lowering, and extending or contracting. The Hexapod is fitted with ultrasound range finders on a rotating turret covering 360 degrees around the robot, controlled by an Arduino based micro-controller. There are also two tilt sensors, mounted at 90 degrees on the robot's body and pressure sensors on each of the robot's feet. The robot is controlled via a serial link to a host PC.

2. Subsumption Architecture

Brooks' Subsumption Architecture [13] was one of the first behavioural control systems, allowing robot programs to be expressed as a hierarchy of levels of competence which interact

with each other to control the robot. Subsumption uses a network of finite state machines known as "behaviours", along with asynchronous message passing over 'wires' between input and output "ports". Behaviours output values to a port and the most recent value output to that port is available for input to the receiver constantly, essentially providing a memory cell located in front of the port into which values are written. Behaviours are grouped into increasing levels of competence, with each level able to tap into wires in lower levels and *suppress* inputs and *inhibit* outputs as required.

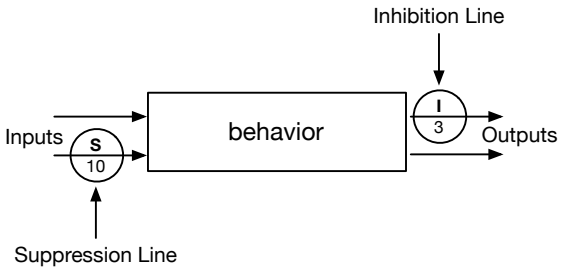

Figure 1. A behaviour module for a Subsumption Architecture, with a suppressor on an input line and an inhibitor on an output line.

Two key primitives are used in Subsumption to allow other behaviours making up different levels of competence to interact: *suppression* of inputs and *inhibition* of outputs. Higher levels of competence replace the input to lower level modules or inhibit their output to produce the desired result. If a suppressor is placed at the input to a module, values from a secondary "suppression" input may replace other input for a pre-defined period. If an inhibitor is placed at the output of a module, a secondary "inhibitory" input can send a signal to block all output from the module for a period of time. The diagrammatic representation of these two Subsumption primitives is shown in Figure 1, with an I or S indicating inhibition or suppression at the top of the circle and the activity period indicated by the number at the bottom. A more comprehensive explanation of these primitives, including code listings is available in [7].

2.1. Implementation

In Brooks' original implementation, these communicating state machines are compiled along with a scheduler which supports behaviours running concurrently on a single processor and in parallel across multiple processors. The ability to mix both simulated and real parallelism via a software scheduler was exploited in Brooks' six-legged walking robot "Genghis," which had a control system consisting of 57 finite state machines running on just four processors [14].

Process-oriented programming provides an advantage when implementing control programs using the Subsumption Architecture, as it gives us independent, parallel processes with message passing from which to build the Subsumption primitives. The authors have previously implemented these primitives in occam-π and used them successfully to construct a basic program to 'bump and wander'; navigating safely within an enclosed space [7].

The process network for a simple robot control program written using the Subsumption Architecture in occam-π is shown in Figure 2. This program is written to run on the the Pioneer 3-DX robot platform, as previously described in section 1. Our example program has three levels of competence, the first of which is to move into space and execute an emergency stop if it gets too close to any object, providing a degree of safety to the robot's motion. The second level of competence detects objects in the centre of the laser scanners' field

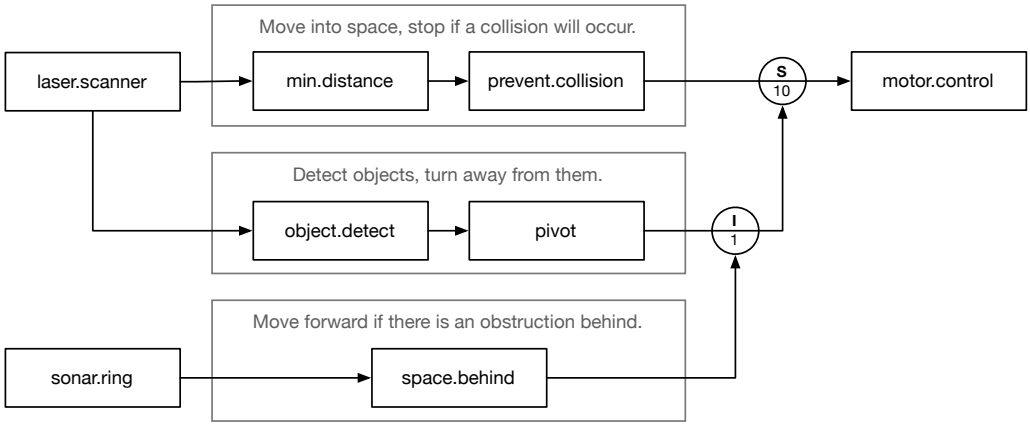

Figure 2. A Subsumption Architecture-based bump and wander program for a robot with three levels of competence.

(i.e. directly in front of it) and backs the robot up while turning, to avoid them. This level upgrades the control system from simply driving into an area until it stops due to proximity to actually being able to navigate around the space. However, as the robot's emergency stop behaviour uses the laser range-finder at the front, the robot might back into walls while trying to avoid obstacles in tight spaces. Our third level uses the sonar sensors at the back of the robot to determine if there is space for the robot to back up. If there is no room, it inhibits the outputs from the module attempting to back the robot up and allows the lower-level forward driving behaviour to take over. This three competence network allows the robot to perform multi-point turns simply by engaging its "back up and turn" and lower level "go forward" behaviours alternately, with no explicit statement of the composite action.

Scaling this early model of Subsumption to larger systems poses difficulties. Increasingly complex interdependencies form as higher levels of competence intercept values being passed between individual modules in lower levels, replacing their input or inhibiting output. In his specification of the Colony Architecture (see Section 3), Connell identifies that Subsumption requires a holistic view and correct behavioural decomposition from the very beginning of design so as to offer the correct inputs and outputs for higher layers to interact with. A later revision to the Subsumption Architecture incorporates a number of changes which attempt to resolve these problems, changing the functioning of inhibition and suppression to require frequent communication and short delay periods [14]. These changes have been implemented in our versions of the Subsumptive primitives and further enhance the suitability of these primitives when using a paradigm that supports message passing.

3. Colony Architecture

Connell's Colony Architecture is a refinement on the early Subsumption Architecture which removes explicit inhibition and replaces it with predicates inside the behaviour modules, allowing only suppression of outputs in lower level behaviours [15]. The Colony Architecture uses a "soup" of modules which are related to one another by the actuator they control, rather than a layered ordering across the system. Unlike Subsumption, additional layers do not necessarily increase in competence; we might introduce modules in a higher layer which provide more general solutions to lower-level control problems for cases where more specific lower-level modules cannot establish the correct action to take. The Colony Architecture removes the Subsumption's ability to "spy" on inputs and replace outputs of internal modules

in lower layers, enhancing the system's modularity by only allowing the input and output of entire layers of behaviour to interact.

The Colony Architecture was used on a multi-processor robot which could also run programs written for the Subsumption Architecture. This multiprocessor robot used 24 loosely-coupled processors to run a system with over 41 behaviours, with software scheduling like that of Brooks' to run multiple behaviours per processor. Both Brooks and Connell identified that the explicit parallelism of behaviours allowed the control system to scale through the addition of more processors, maintaining reactivity and performance even with the introduction of more behaviours.

3.1. Implementation

The Colony Architecture uses very similar primitives to the Subsumption Architecture by virtue of being based on it. A *retriggerable monostable* primitive is added, which is used when an event (defined as a single point of activity which does not persist in the environment) should trigger a behaviour. The detection of these events is performed using initiation and satisfaction predicates, which set the monostable true or false depending on the condition of the event. The monostable itself maintains a true value for a period of time, but eventually resets itself if not reset by the satisfaction predicate, acting much like a piece of memory with a watchdog timer. The monostable is used to persist the point state from the environment, allowing it to have time to influence the system even if the state ceases to exist in the environment.

Modifications to the Subsumption primitives involve both inhibition and suppression requiring continued sending of messages over the control channels. Early Subsumption tended to use a single message and long delays, whereas the Colony Architecture and Brooks' later Subsumption both use short delays with regular messaging along the control channel.

4. Action-Selection

Maes' Action-Selection architecture relies on a network of independent competence modules and the use of activation levels to control which modules are executed [16,17]. Activation levels in the network are propagated such that an executable module primes modules which can run after it in a task, while a non-executable module will prime those which run before it, causing activation to pool in the first behaviour in a task which is suitable for execution. Inhibition, or "conflictor" lines are connected between modules that oppose each other's behaviour, when a module with such a connection becomes active it inhibits the activation of the other modules which would impede completion of its task.

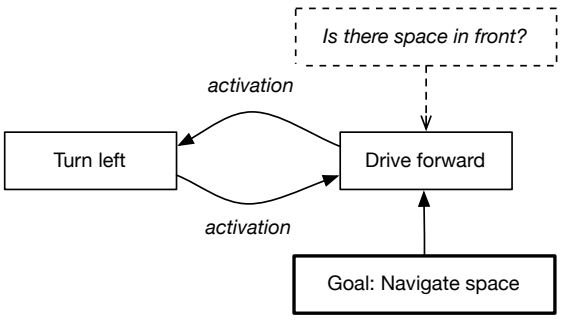

Figure 3. A set of Action-Selection competence modules to move within a space. Activation spread is accomplished via bi-directional connections between modules, as shown.

A simple example of Action-Selection is shown in Figure 3, a robot control program which has a goal to navigate into space. The competence modules for this program are "drive forward" and "turn left", the "drive forward" module is preconditioned on there being space in front of the robot. The goal "navigate space" raises the activation of the "drive forward" module and it will become active, moving the robot forward. Once the robot runs out of space in front of it, the "drive forward" module will become inactive, as the "has space in front" precondition will become false. The "drive forward" behaviour will then pass its activation on to the "turn left" module, causing a turn until the precondition for "drive forward" becomes true once more (i.e. there is free space in front of the robot).

4.1. Implementation

Use of the Action-Selection mechanism in occam-π is most easily achieved through the creation of a second decision-making network consisting of a number of 'cells' which propagate the activation levels for each behaviour to calculate activation values for each cell. Cells which have an activation level over their threshold can be activated if they are 'executable'. To be 'executable' all of a cell's preconditions must be met, and the decision-making network will therefore capture all of the preconditions to allow it to arbitrate between behaviours and activate those that should be activated. These activated cells in the decision layer trigger the execution of behaviours in the robot control system itself, and free those control behaviours from having to incorporate the propagation of activation or management of preconditions. Goals are connected into the decision layer after the last module in a behaviour that completes them, and feed activation back through the modules responsible for completing the task until it pools in the first task, raising it above the activation threshold.

Partly due to its inspiration from neural networks, the process architectures that result from the implementation of Action-Selection are complex and require a second decision network to make their implementation relatively neutral to behaviours.

5. Motor Schema

Arkin's Motor Schema approach to control uses multiple concurrent schemas (behaviours) active during the completion of a high level task [18]. Two types of schema are employed in the architecture: motor and perceptual. Motor schemas are behaviours that control the motion or activity of the robot, such as 'stay on path' or 'avoid obstacles'. Perceptual schemas identify features and conditions in the environment that provide data necessary for a given motor schema to function, for example "find terrain" might supply a clear path vector to "stay on path". In Arkin's system, multiple schemas may effect action at the same time, and these actions are merged through vector addition of potential fields.

Groupings of perceptual and motor schemas which achieve a given task are known as assemblages. Some assemblages may be present throughout the entire runtime of the control program, such as those that provide emergency stop or hazard avoidance facilities. Additionally, a planner module may load and unload different assemblages based on the input from the perceptual schemas connected to it. This mechanism provides effective re-use of components, as we can re-use parameterised perceptual and motor schemas across multiple assemblages.

To illustrate the Motor Schema approach, an example program is presented in Figure 4, written for the LynxMotion AH3-R Hexapod robot [8] as described in section 1. The sample program shown in Figure 4 has two assemblages which are loaded constantly: one to detect body tilt (`detect.tilt`) and keep the robot level and another to detect obstacles in the path of the robot (`detect.obstacles`) and generate a vector away from them. The planner is able to load additional assemblages based on perceptual schemas which are connected to it. In this

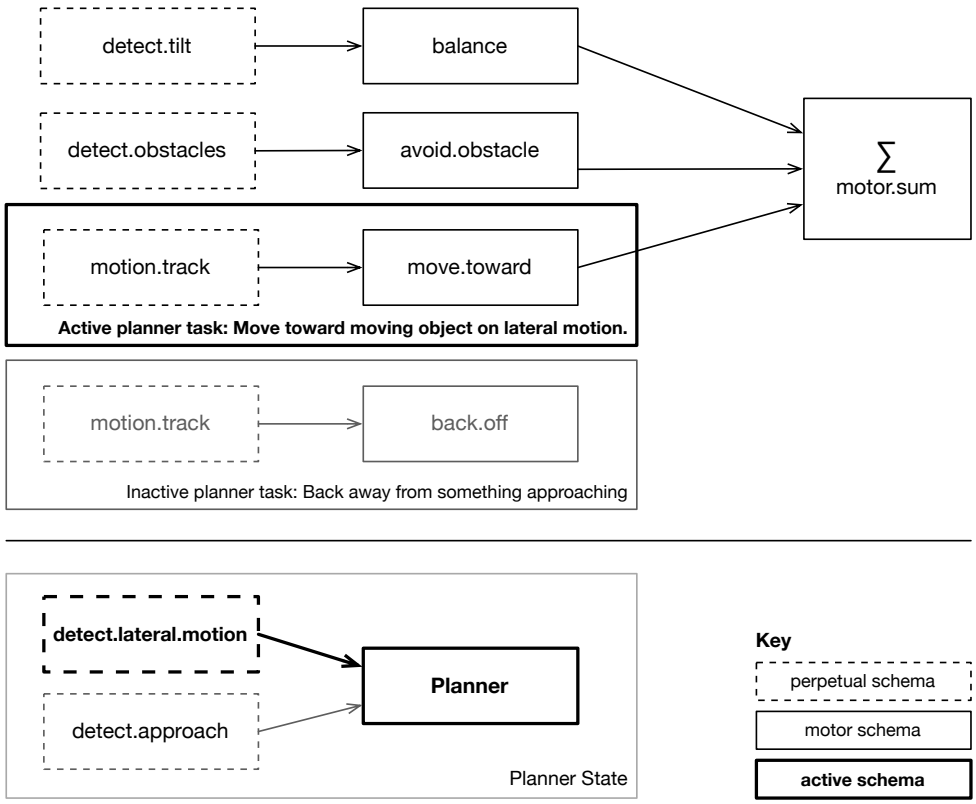

Figure 4. A motor-schema based control program to navigate a robot to investigate motion and run away if approached.

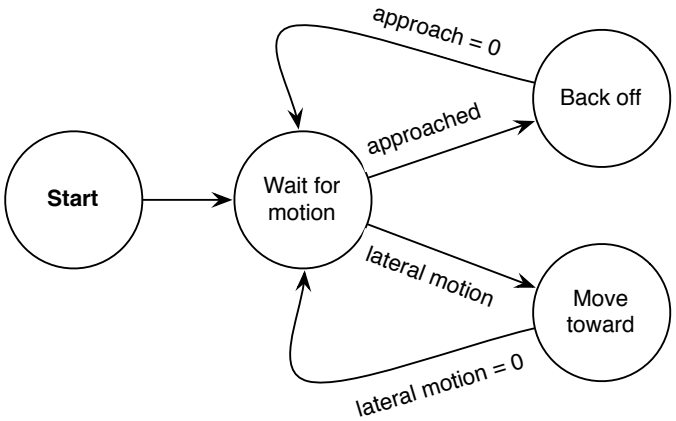

Figure 5. State machine of the planner for an example control program using Motor Schemas which investigates motion and runs away if approached.

case we have two perceptual schemas connected to the planner: one to detect lateral motion (`detect.lateral.motion`) which loads an assemblage to move towards (investigate) the source of the motion, and another which loads an assemblage which backs away from the source of the approach (`detect.approach`). The state machine for the planner is shown in Figure 5.

Motor Schemas provide the reactive component of Arkin's Autonomous Robot Architecture (AuRA), which combines the aforementioned planner with a spatial reasoner and other

deliberative levels of function [19].

5.1. Implementation

The key primitive for the implementation of Motor Schemas is the vector sum, an implementation of which is shown in Listing 1. The example provided is suitable for controlling motion in a 2D plane, it is straightforward to add more components to allow control in a 3D space (x,y,z), allowing the system's behaviours to influence height or tilt. The planner in a Motor Schema based system is a custom-built state machine which uses perceptual schemas to determine when to change between states.

```
PROTOCOL VECTOR IS REAL32; REAL32:

PROC motors (VAL []REAL32 gain, CHAN []VECTOR in?)
  MOBILE []REAL32 x, y:
  SEQ
    x := MOBILE [SIZE in]REAL32
    y := MOBILE [SIZE in]REAL32
    SEQ i = 0 FOR SIZE in
      x[i], y[i] := 0.0, 0.0

    WHILE TRUE
      INITIAL REAL32 x.v IS 0.0:
      INITIAL REAL32 y.v IS 0.0:
      SEQ
        ALT i = 0 FOR SIZE in
          in[i] ? x[i]; y[i]
            SEQ
              x[i] := x[i] * gain[i]
              y[i] := y[i] * gain[i]

        SEQ i = 0 FOR SIZE in
          SEQ
            x.v := x.v + x[i]
            y.v := y.v + y[i]

        -- drive motors ...
:
```

Listing 1. An occam-π implementation of the vector sum primitive which allows for the control of motion in a 2D plane

6. Distributed Architecture for Mobile Navigation (DAMN)

Rosenblatt's Distributed Architecture for Mobile Navigation (DAMN) combines independent, asynchronous modules with arbiters performing command fusion via a voting mechanism [20]. The overall goals of the system are prioritised via the weighting of votes placed by each module. Arbiters make a decision on the set of votes which have been received within their time step. This provides asynchronous operation of system components and allows the behaviours to be a mix of deliberative and reactive modules, emitting decisions at their natural rates. Arbiters in DAMN offer a set of commands to behaviours; a steering arbiter might offer varying degrees of turn and the behaviour modules would then be able to vote on each possibility. Votes made by behaviours are normalised, and the choice which has the highest vote amount is chosen to occur.

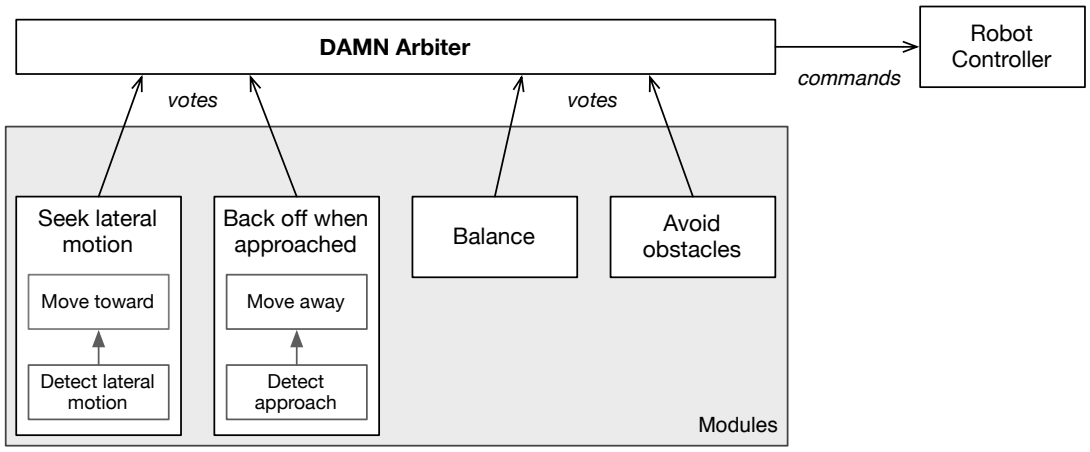

Figure 6. A DAMN based control program to navigate a robot to investigate motion and run away if approached. The arbiter sends commands to the robot itself based on the votes made by behaviours.

Arbiters may be connected to an adaptive mode manager, allowing the weighting of different behaviours to be changed while the system is running. A mode manager such as SAUSAGES altering the vote weightings allows for sequential action [21]. For example, a robot might have a primary stage of operation where it locates all target objects (soda cans, red balls etc.) and a secondary stage where it retrieves all of those target objects. A mode manager would first weight highly all of those behaviours responsible for the target finding abilities, then reduce those weights and increase those to the behaviours responsible for retrieving the objects.

A sample program implemented using DAMN is shown in Figure 6. It performs the same task as the earlier example implemented using a Motor Schema approach, using the six-legged walker to approach moving objects and back away from objects that approach it.

6.1. Implementation

The development of DAMN-based robot control systems in occam-π requires the implementation of 'arbiter' processes for each actuator to be controlled by the system. Each of these arbiters will offer appropriate choices for actions to take with the actuator that they control to the behaviour modules of the system.

7. Conclusions

Whilst examining a number of behavioural architectures in the light of process-orientation, we have found a number of design strategies and primitives which "fit" well with our programming model.

Brooks' Subsumption Architecture has previously shown promise when used for robotics in occam-π [7], and the adaptations to rules introduced by Brooks' later work involving the use of communication patterns to control inhibition and suppression further enhance its suitability. A lack of modularity, identified by both the ourselves and authors of other related control systems mean that systems structured using the Subsumption architecture tend to run into scaling problems. This is due to the tight bindings established between behaviours inside the layers themselves.

Connell's Colony Architecture deals with some of the scalability issues from Subsumption, structuring behaviours around the effectors they control instead of into tight layers. This structuring means that the suppressor becomes the main primitive, with priority hierarchies

of behaviour built up around the effectors. A major plus of the Colony Architecture is that behaviours do not end up tightly bound to each other, allowing it to offer better scalability than Subsumption.

Maes' Action-Selection, being a neural network influenced approach is significantly different from the Subsumption and Colony architectures because it does not rely on message passing or finite state machines in its definition. This architecture takes no advantage of the communicating process model in the actual robot control code, which is not ideal for meshing with other architectural components and process-oriented hardware interfaces. Additionally the need to implement a separate neural network based decision layer from the actual robot control processes and the highly connected nature of the decision layer makes implementation complex. In support of this view, Arkin notes in [22] that "no evidence exists of how easily the current competence module formats would perform in real-world robotic tasks", due to lack of implementation on actual robot platforms.

Arkin's Motor Schemas offer a method of command fusion which is simple and effective, matching with our typical use of process-orientation in robotic control and having flexibility for the kinds of motion possible with different platforms. In the context of the wider architecture AuRA, the finite-state machine based planner and re-use of perceptual schemas make this approach to control modular and flexible, allowing both deliberative and reactive behaviours to be expressed in the same way.

Where Motor Schemas offer a blending approach to resolving the action of many behaviours into one coherent choice, Rosenblatt's DAMN allows the use of an arbitration-based approach. Behaviours vote on potential actions and decide on the correct action to take via a centralised arbiter, customised for the potential actions that can be taken for each effector. DAMN provides a framework which can exploit message passing for decision making without the implementation overhead and connection complexity of a neural network, whilst allowing complete freedom to the internal structure of the voting behaviours. The asynchronous nature of DAMN also allows a seamless combination of deliberative and reactive components each working at their own frequency, with the relative proportion of their inputs able to be counteracted through weighting. An AuRA-like planner or other mode manager could also easily be used to provide sequential or adaptive behaviour, providing a coherent framework.

8. Future Work

This paper has reviewed and introduced a number of behavioural robotics architectures in the context of process-oriented programming, specifically using occam-π. It would be highly beneficial to implement a single control program using each architecture in turn, to provide metrics for detailed comparison and evaluation of their process-oriented implementations.

The creation of a library and documentation providing primitives and development methodologies for Subsumption, Colony, Motor Schema and DAMN architectures would provide useful assistance in writing robotics programs using occam-π. Further exploration of hybrid approaches, using multiple architectures in the homogenous process-oriented environment may yield useful combinations for achieving particular goals.

The authors envisage that a unified library of components for building programs using different architectures would be greatly beneficial in the context of a visual robotics programming environment, such as that described in [23].

Architectures could be considered in the design of hardware interfaces to platforms for use in process-oriented robotics. For example, effective use of DAMN could be achieved via the provision of arbiters inside the hardware interface itself, building scalability and the design paradigm in from the hardware up.

Having examined existing architectures for their suitability to process-oriented control,

it would be interesting to explore the development of systems that fully take advantage of the language features present in occam-π such as barriers and dynamic network creation. Additionally, the development of architectures and design patterns which lend themselves to modelling in CSP [24] and formal verification via tools such as Formal Methods' FDR [25] may allow the creation of provably reliable control applications.

Acknowledgements

Jon Simpson is funded by the Computing Lab at the University of Kent and the EPSRC DTA. Carl Ritson is funded by EPSRC grant EP/D061822/1. We thank all of those who continue to work on the Transterpreter project [26], especially Matt Jadud and Christian Jacobsen whose ideas on robot architectures have been formative to this work. We also thank the anonymous reviewers for their comments, which helped us improve and shape this paper.

References

[1] Erann Gat. On Three-Layer Architectures. In David Kortenkamp, R. Peter Bonnasso, and Robin Murphy, editors, *Artificial Intelligence and Mobile Robots*, pages 195–210. AAAI Press, 1997.

[2] Marvin Minsky. *The Society of Mind*. Simon & Schuster, Inc., New York, NY, USA, 1986.

[3] P.H. Welch and F.R.M. Barnes. Communicating Mobile Processes: Introducing occam-π. In A.E. Abdallah, C.B. Jones, and J.W. Sanders, editors, *25 Years of CSP*, volume 3525 of *Lecture Notes in Computer Science*, pages 175–210. Springer Verlag, April 2005.

[4] Christian L. Jacobsen and Matthew C. Jadud. The Transterpreter: A Transputer Interpreter. In *Communicating Process Architectures 2004*, pages 99–107, 2004.

[5] Jonathan Simpson, Christian L. Jacobsen, and Matthew C. Jadud. A Native Transterpreter for the LEGO Mindstorms RCX. In Alistair A. McEwan, Steve Schneider, Wilson Ifill, and Peter H. Welch, editors, *Communicating Process Architectures 2007*, volume 65 of *Concurrent Systems Engineering*, Amsterdam, The Netherlands, July 2007. IOS Press.

[6] Matthew C. Jadud, Christian L. Jacobsen, Carl G. Ritson, and Jonathan Simpson. Safe parallelism for behavioral control. In *IEEE International Conference on Technologies for Practical Robot Applications (TePRA)*, November 2008.

[7] Jonathan Simpson, Christian L. Jacobsen, and Matthew C. Jadud. Mobile Robot Control: The Subsumption Architecture and occam-π. In Frederick R. M. Barnes, Jon M. Kerridge, and Peter H. Welch, editors, *Communicating Process Architectures 2006*, pages 225–236, Amsterdam, The Netherlands, September 2006. IOS Press.

[8] Lynxmotion, Inc. AH3-R Walking Robot. http://www.lynxmotion.com/Category.aspx?CategoryID=92.

[9] Peter H. Welch and Neil Brown. The JCSP Home Page: Communicating Sequential Processes for Java. http://www.cs.kent.ac.uk/projects/ofa/jcsp/, March 2008.

[10] Otto J. Anshus, John Markus Bjørndalen, and Brian Vinter. PyCSP - Communicating Sequential Processes for Python. In Alistair A. McEwan, Wilson Ifill, and Peter H. Welch, editors, *Communicating Process Architectures 2007*, volume 65 of *Concurrent Systems Engineering*, pages 229–248, Amsterdam, The Netherlands, jul 2007. IOS Press.

[11] Jan F. Broenink and Dusko S. Jovanovic. Graphical Tool for Designing CSP Systems. In Ian R. East, David Duce, Mark Green, Jeremy M. R. Martin, and Peter H. Welch, editors, *Communicating Process Architectures 2004*, pages 233–252, September 2004.

[12] Mobile Robots, Inc. Pioneer 3-DX Mobile Robot. http://www.activrobots.com/ROBOTS/p2dx.html.

[13] Rodney A. Brooks. A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, March 1986.

[14] Rodney A. Brooks. A robot that walks; emergent behaviors from a carefully evolved network. Technical report, MIT, Cambridge, MA, USA, 1989.

[15] Jonathan H. Connell. A colony architecture for an artificial creature. Technical report, Cambridge, MA, USA, 1989.

[16] Pattie Maes. The dynamics of action selection. In *IJCAI*, pages 991–997, 1989.

[17] Pattie Maes. How to do the right thing. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1989.

[18] Ronald C. Arkin. Motor schema based navigation for a mobile robot: An approach to programming by behavior. In *IEEE International Conference on Robotics and Automation*, volume 4, pages 264–271, Mar 1987.

[19] Ronald C. Arkin and Tucker Balch. AuRA: Principles and Practice in Review. *Journal of Experimental and Theoretical Artificial Intelligence*, 9:175–189, 1997.

[20] Julio K. Rosenblatt. Damn: A distributed architecture for mobile navigation. In H. Hexmoor and D. Kortenkamp, editors, *proceedings of the 1995 AAAI Spring Symposium on Lessons Learned from Implemented Software Architectures for Physical Agents*, Menlo Park, CA, March 1995. AAAI Press.

[21] Jay Gowdy. Sausages: Between planning and action. Technical Report CMU-RI-TR-94-32, Robotics Institute, Pittsburgh, PA, September 1994.

[22] Ronald C. Arkin. *Behavior-based Robotics*. MIT Press, Cambridge, MA, USA, 1998.

[23] Jonathan Simpson and Christian L. Jacobsen. Visual Process-oriented Programming for Robotics. In Peter H. Welch, Susan Stepney, Fiona A.C. Polack, Frederick R. M. Barnes, Alistair A. McEwan, Gardiner S. Stiles, Jan F. Broenink, and Adam T. Sampson, editors, *Communicating Process Architectures 2008*, volume 66 of *Concurrent Systems Engineering*, pages 365–380, Amsterdam, The Netherlands, September 2008. IOS Press.

[24] C. A. R. Hoare. *Communicating sequential processes*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1985.

[25] Formal Systems (Europe) Ltd., 3, Alfred Street, Oxford. OX1 4EH, UK. *FDR2 User Manual*, May 2000.

[26] The Transterpreter Project. The Transterpreter Project - Concurrency, Everywhere. http://www.transterpreter.org/, July 2009.