Citation for published version:
Wortham, RH, Gaudl, SE & Bryson, JJ 2019, 'Instinct: A Biologically Inspired Reactive Planner for Intelligent Embedded Systems', *Cognitive Systems Research*, vol. 57, pp. 207-215.
https://doi.org/10.1016/j.cogsys.2018.10.016

DOI:
10.1016/j.cogsys.2018.10.016

Publication date:
2019

Document Version
Peer reviewed version

Link to publication

Instinct: A Biologically Inspired Reactive Planner for Intelligent Embedded Systems

Robert H. Wortham[a], Swen E. Gaudl[c], Joanna J. Bryson[b]

[a]*Department of Electronic and Electrical Engineering, University of Bath, Claverton Down, Bath, BA2 7AY, UK*
[b]*Department of Computer Science, University of Bath, Claverton Down, Bath, BA2 7AY, UK*
[c]*Falmouth University, Games Academy, Penryn Campus, Treliever Road, Penryn TR10 9FE, UK*

Abstract

The Instinct Planner is a new biologically inspired reactive planner, based on an established behaviour based robotics methodology and its reactive planner component — the POSH planner implementation. It includes several significant enhancements that facilitate plan design and runtime debugging. It has been specifically designed for low power processors and has a tiny memory footprint. Written in C++, it runs efficiently on both Arduino (Atmel AVR) and Microsoft VC++ environments and has been deployed within a low cost maker robot to study AI Transparency. Plans may be authored using a variety of tools including a new visual design language, currently implemented using the Dia drawing package.

Keywords: Reactive Planning, Instinct, Arduino, POSH, BOD, Bio-Inspired
2010 MSC: 00-01, 99-00

---

1. INTRODUCTION

From the 1950's through to the 1980's the study of embodied AI assumed a cognitive symbolic planning model for robotic systems — SMPA (Sense Model Plan Act) — the most well known example of this being the Shakey robot project [1]. In this article, we present a novel approach that can be used to develop embodied agents beyond the scale of the original scope of SMPA cycles. In the SMPA model, the world is first sensed and a model of the world is constructed within the AI. Based on this model and the objectives of the AI, a plan is constructed to achieve the goals of the robot. Only then does the robot act. Although this idea seemed logical and initially attractive, it was found to be quite inadequate for complex, real world environments. Generally the world cannot be fully modelled until the robot plan is underway, since sensing the world requires moving through it. Also, where environments change faster than the rate at which the robot can complete its SMPA cycle, the planning simply cannot keep up. Brooks [2] provides a more comprehensive history, which is not repeated here.

In the 1990's Rodney Brooks and others [3] introduced the then radical idea that it was possible to have intelligence without representation [4]. Brooks developed his subsumption architecture as a pattern for the design of intelligent embodied systems that have no internal representation of their environment, and minimal internal state. These autonomous agents could traverse difficult terrain on insect-like legs, appear to interact socially with humans through shared attention and gaze tracking, and in many ways appeared to posses behaviours similar to that observed in animals. However, the systems produced by Brooks and his colleagues could only respond immediately to stimuli from the world. They had no means of focusing attention on a specific goal or of executing complex sequences of actions to achieve more complex behaviours. The original restrictions imposed by Brooks' subsumption architecture were subsequently relaxed with later augmentations such as timers, effectively beginning the transition to systems that used internal state in addition to sensory input in order to determine behaviour.

Following in-depth studies of animals such as the stickleback fish and gulls in their natural environments, ideas of how animals perform action selection were originally formulated by Tinbergen and other early ethologists [5, 6, 7]. Reactions are based on pre-determined drives and competences, but depend also on the internal state of the organism [8]. Bryson [9] harnessed these ideas to achieve a major step forwards with the POSH (Parallel Ordered Slipstack Hierarchy) reactive planner and the BOD (Behaviour Oriented Design) methodology, both of which are strongly biologically inspired. Bryson [9] uses BOD to successfully model and simulate primate behaviour in a variety of realistic scenarios such as Macaques.

It is important to understand the rationale behind biologically inspired reactive planning. It is based on the idea that biological organisms constantly sense the world, and generally react quickly to sensory input, based on a hierarchical set of behaviours structured as Drives, Competences and Action Patterns. Their reactive plan uses a combination of sensory inputs and internal priorities to determine which plan elements to execute, ultimately resulting in the execution of leaf nodes in the plan, which in turn execute real world actions. For further read-

---

Email addresses: `r.h.wortham@bath.ac.uk` (Robert H. Wortham), `swen.gaudl@gmail.com` (Swen E. Gaudl), `j.j.bryson@bath.ac.uk` (Joanna J. Bryson)

ing see Gurney et al. [10], Prescott et al. [11] and Seth [12]. At run-time, the reactive plan itself is essentially fixed. Various slower reacting systems may also be used to modify priorities or other parameters within the plan. These slower reacting systems might be compared with emotional or endocrinal states in nature that similarly affect reactive priorities [13, 14]. Similarly the perception of senses can be affected by the internal state of the plan, an example being the latching (or hysteresis) associated with sensing [15].

In nature, the equivalent process to a reactive plan is subject to possible learning that may change the plan parameters or even modify the structure of the plan itself as new skills and behaviours are learned. This learning may take place ontogenetically, i.e. within the lifetime of an individual, or phylogenetically, by the process of natural selection, across the lifetimes of many individuals. Bryson's BOD approach suggests that humans provide most of the necessary learning in order to improve the plan over time, in place of natural selection. However, Gaudl [16], Gaudl and Bryson [13] successfully uses genetic programming to automate part of this learning process, albeit within a computer game simulation.

A reactive plan is re-evaluated on every plan cycle, usually multiple times per second, and this requires that the inquiries from the planner to the senses and the invocation of actions should respond quickly. This fast cycling enables the reactive plan to respond quickly to changes in the external environment, whilst the plan hierarchy allows for complex sequences of behaviours to be executed. Applying these ideas to robots we can see that for senses, this might imply some caching of sense data. For actions, it also implies that long running tasks (relative to the rate of plan execution), need to not only return success or failure, but also another status to indicate that the action is still in progress and the plan must wait at its current execution step before moving on to its next step. The action may be executing on another thread, or may just be being sampled when the call to the action is made. This is implementation specific and does not affect the functioning of the planner itself. If re-invoked before it completes, the action immediately returns an In Progress response. In this way, longer running action invocations should not block the planner from responding to other stimuli that may still change the focus of attention by, for example, releasing another higher priority Drive.

Each call to the planner within the overall scheduling loop of the robot starts a new *plan cycle*. In this context an action may be a simple primitive, or may be part of a more complex pre-defined behaviour module, such as a mapping or trajectory calculation subsystem. It is important to note that the BOD methodology does not predicate that all intelligence is concentrated within the planner. Whilst the planner drives action selection, considerable complexity can still exist in sensory, actuation and other probabilistic or state based subsystems within the overall agent [9].

The digital games industry has advanced the use of AI for the simulation of non player characters [17]. The *BehaviorTree* (BT) concept is similarly hierarchical to POSH plans, but have additional elements that more easily allow logical operations such as AND, OR, XOR and NOT to be included within the plan. For example it is possible for a goal to be reached by successfully executing only one of a number of behaviours, trying each in turn until one is successful. Bryson's original design of POSH does not easily allow for this kind of plan structure.

BTs are in turn simplifications of Hierarchical Task Network (or HTN) planners [18]. Like POSH, HTN planners are able to create and run plans that contain recursive loops, meaning that they can represent any computable algorithm.

2. THE INSTINCT PLANNER

The Instinct Planner[1] is a reactive planner based on Bryson's POSH [19, 9]. It includes several enhancements taken from more recent work extending POSH [15, 13], together with some ideas from other planning approaches, notably *BehaviorTree* [BT — 20, 17, 21]. A POSH plan consists of a *Drive Collection (DC)* containing one or more *Drives*. Each *Drive (D)* has a priority and a releaser. When the *Drive* is released as a result of sensory input, a hierarchical plan of *Competences*, *Action Patterns* and *Actions* follows.

- *Action (A)*: Actions represent the leaf nodes in our reactive plan hierarchy. Actions invoke behaviour primitives encoded within the Agent. These behaviours may be simple, such as halting robot motion, or may be more complex, such as initiating a process to turn a robot in a specific direction. Instinct adds the concept of an *Action Value*, a parameter stored within the Action and passed as a parameter to the underlying primitive behaviour. This allows specific Actions to be encoded within the Instinct plan that invoke more general purpose behaviour primitives. A simple example would be a primitive to turn a robot by an angle specified in the Action Value parameter. In the case of simple and immediate actions, the primitive behaviour returns either SUCCESS or FAIL. For more complex, longer running behaviours the immediate return value would be IN PROGRESS, indicating that the requested behaviour has commenced, but is not yet complete. Subsequent invocations of the behaviour request will return IN PROGRESS until the behaviour finally returns SUCCESS. These return values are returned by the Action itself, and use by the higher levels in the reactive plan to determine the plan execution path.

- *Action Pattern (AP)*: Action patterns are used to reduce the computational complexity of search within the plan space and to allow a coordinated fixed sequential execution of a set of elements. An action pattern—$AP = [\alpha_0, \ldots, \alpha_k]$—is an ordered set of Actions that does not use internal precondition or additional perceptual information. It provides the simplest plan structure in POSH and allows for the optimised execution of behaviours. An example would be an agent that always shouts and moves its hand upwards when touching an hot object. In this

---

[1] The Instinct Planner was first presented at the ICAPS PlanRob Workshop, London, June 2016.

case, there is no need for an additional check between the two Action primitives if the agent should always behave in that manner. An AP needs to execute all child elements before it completes successfully.

- *Competence (C)*: Competences form the core part of POSH plans. A competence $C = [c_0, \ldots, c_j]$ is a self-contained basic reactive plan (BRP) where $c_b = [\pi, \rho, \alpha, \eta], b \in [0, \ldots, j]$ are tuples containing π, ρ, α and η: the priority, precondition, child node of C and maximum number of retries. The priority determines which of the child elements to execute, selecting the one with the highest priority first. The precondition is a concatenated set of senses that either release or inhibit the child node α. The child node itself can be another Competence or an Action or Action Pattern. To allow for noisy environments a child node can fail a number of times, specified using η, before the Competence ignores the child node for remaining time within the current cycle. A Competence sequentially executes its hierarchically organised child-nodes where the highest priority node is the competence goal. A Competence fails if no child can execute or if an executed child fails.

- *Drive (D)*: A Drive—$D = [\pi, \rho, \alpha, A, v]$—allows for the design and pursuit of a specific behaviour as it maintains its execution state. The Drive Collection determines which Drive receives attention based on each Drive's π, the associated priority of a Drive. ρ is the *releaser*, a set of preconditions using senses to determine if the drive should be pursued. α is either an Action, Action Pattern or a Competence and A is the root link to the Drive Collection. The last parameter v specifies the execution frequency, allowing POSH to limit the rate at which the Drive can be executed. This allows for coarse grain concurrency of Drive execution (see below).

- *Drive Collection (DC)*: The Drive Collection—DC—is the root node of the plan—$DC = [g, D_0, \ldots, D_i]$. It contains a set of Drives $D_a, a \in [0 \ldots i]$ and is responsible for giving attention to the highest priority Drive. To allow the agent to shift and focus attention, only one Drive can be active in any given cycle. Due to the parallel hierarchical structure, Drives and their sub-trees can be in different states of execution. This allows for cooperative multitasking and a quasi-parallel pursuit of multiple behaviours at the Drive Collection level.

For a full description of POSH and BOD see Bryson [9].

2.1. Enhancements and Innovations

The Instinct Planner is engineered to be of practical use in simple, low cost robots. In this section we describe specific enhancements and innovations that support this objective. In addition, we chose low cost, widely available, contemporary development environments and tools, in order to make it as easy as possible to build robots using Instinct.

The Instinct Planner includes a full implementation of what we term *Drive Execution Optimisation (DEO)*. DEO avoids a full search of the plan tree at every plan cycle which would be expensive. It also maintains focus on the task at hand. This corresponds loosely to the function of consciousness attention seen in nature [22]. A form of this was in Bryson's original POSH, but has not been fully implemented in subsequent versions. The Drive, Competence and Action Pattern elements each contain a *Runtime Element ID*. These variables are fundamental to the plan operation. Initially they do not point to any plan element. However, when a Drive is released the plan is traversed to the point where either an Action is executed, or the plan fails at some point in the hierarchy. If the plan element is not yet completed it returns a status of In Progress and the IDs of the last successful steps in the plan are stored in Runtime Element ID in the Drive, Competence and Action Pattern elements. If an action or other sub element of the plan returns success, then the next step in the plan is stored. On the next cycle of the drive, the plan hierarchy is traversed again but continues from where it got to last plan cycle, guided by the Runtime Element IDs. A check is made that the releasers are still activated (meaning that the plan steps are still valid for execution), and then the plan steps are executed. If a real world action fails, or the releaser check fails, then the Runtime Element ID is once again cleared. During execution of an Action Pattern (a relatively quick sequence of actions), sensory input is temporarily ignored immediately above the level of the Action Pattern. This more closely corresponds to the reflex behaviour seen in nature. Once the system has started to act, then it continues until the Action Pattern completes, or an element in the Action Pattern explicitly fails. Action Patterns are therefore not designed to include Actions with long running primitive behaviours.

In addition to these smaller changes there are three major innovations in the Instinct Planner that increase the range of plan design options available to developers:

- Runtime alteration of drive priority — This closely follows the RAMP model of Gaudl and Bryson [13, 14] which in turn is biologically inspired, based on spreading activation in neural networks. Within the Instinct Planner we term this *Dynamic Drive Reprioritisation (DDR)*. DDR is useful to modify the priority of drives based on more slowly changing stimuli, either external or internal. For example, a recharge battery drive might be used to direct a robot back to its charging station when the battery level becomes low. Normally this drive might have a medium priority, such that if only low priority drives are active then it will return when its battery becomes discharged to say 50%. However, if there are constantly high priority drives active, then the battery level might reach a critical level of say 10%. At that point the recharge battery drive must take highest priority. A comparison can be drawn here with the need for an animal to consume food. Once it is starving the drive to eat assumes a much higher priority than when the animal experiences normal levels of hunger. For example, it will take more risks to eat, rather than flee from predators.

- Flexible latching — This provides for a more dynamic form of sense hysteresis, based not only on plan configuration, but also the runtime focus of the plan, following the work of Rohlfshagen and Bryson [15]. Within the Instinct Planner we term it *Flexible Sense Hysteresis (FSH)*. This hysteresis primarily allows for noise from sensors and from the world, but Rohlfshagen's paper also has some basis in biology to avoid dithering by prolonging behaviours once they have begun. If the Drive is interrupted by one of a higher priority, then when the sense is again checked, it will be the Sense Flex Latch Hysteresis that will be applied, rather than the Sense Hysteresis.

- Enhanced Competence — It is now possible to group a number of competence steps by giving them the same priority and add logical conditions to the grouping. We refer to this grouping as a *priority group*. Items within a group have no defined order. Within a priority group, the Competence itself can specify whether the items must all be successfully executed for the Competence to be successful (the AND behaviour), or whether only one item need be successful (the OR behaviour). In the case of the OR behaviour, several items within the group may be attempted and may fail, before one succeeds. At this point the Competence will then move on to higher priority items during subsequent plan cycles. A Competence can have any number of priority groups within it, but all are constrained to be either AND or OR, based on the configuration of the Competence itself. This single enhancement, whilst sounding straightforward, increases the complexity of the planner code significantly, but allows for much more compact, sophisticated plans, with a richer level of functionality achievable within a single Competence than was provided with the earlier POSH implementations.

2.2. Multi Platform

The Instinct planner itself is able to run both within Microsoft Visual C++ and the Arduino development environments [23] as a C++ library. The Arduino uses the Atmel AVR C++ compiler [24] with the AVR Libc library [25] — a standards based implementation of `gcc` and `libc`. This arrangement harnesses the power of the Visual C++ Integrated Development Environment (IDE) and debugger, hugely increasing productivity when developing for the Arduino platform, which has no debugger and only a rudimentary IDE. We have a complete implementation of the Instinct Planner on an Arduino based robot named R5, see figure 1. The robot runs using various Instinct plans and has been successfully used in several robot transparency experiments [26]. Due to the very compact memory architecture of Instinct, the planner is able to store plans with up to 255 elements within the very limited 8KB memory (RAM) available on the Arduino Mega (Atmel AVR ATmega2560 microcontroller). The 255 element limitation arises from the use of a single byte to store plan element IDs within the Arduino environment.

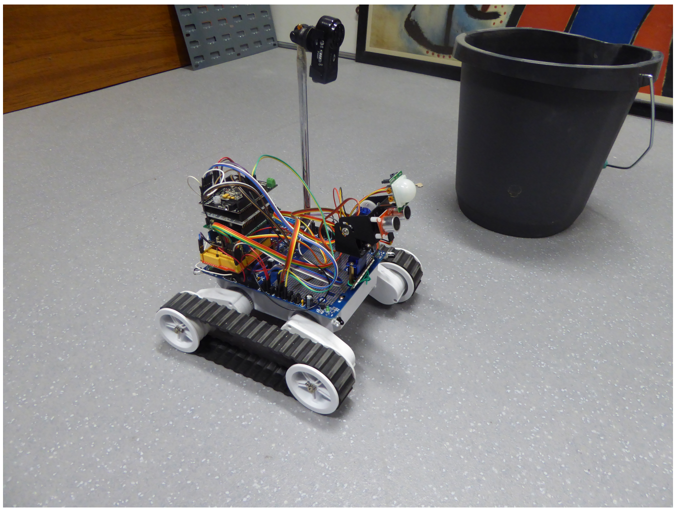


Figure 1: The R5 Arduino based Maker Robot in a laboratory test environment. The camera mounted on the robot is used to record robot activity, but is not used by the robot itself.

The robot itself has active infrared and ultrasonic distance sensors, a head capable of scanning its environment, a passive infrared (PIR) sensor to assist in the detection of humans interacting with it, and proprioceptive sensing of odometry (distance travelled) and drive motor current. It has simple and more complex underlying behaviours that can be invoked by the planner, such as the ability to turn in the direction of the most clear pathway ahead, or to use its head to scan for the presence of a human. It also has a multicoloured headlight that may be used for signalling to humans around it. Finally, it has an electronically erasable programmable read only memory (EEPROM) that permanently stores both the robot's configuration parameters and the Instinct plan. This leverages the planner's ability to serialise plans as a byte stream, and then reconstitute the plan from that stream at startup.

2.3. Memory Management

In order to produce a planner that operates effectively in an environment with severely limited working memory resources (RAM), considerable design effort has been applied to the memory management architecture within the planner. There are 6 separate memory buffers, each holding fixed record length elements for each element type in the plan — Drives, Competences, Competence Elements, Action Patterns, Action Pattern Elements and Actions. An instance of Instinct has a single Drive Collection — the root of the plan.

Within each plan element, individual bytes are divided into bit fields for boolean values, and the data is normalised across elements to avoid variable length records. This means, for example, that Competence Elements hold the ID of their parent Competence, but the Competence itself does not hold the IDs of each of its child Competence Elements. At runtime a search must be carried out to identify which Competence Elements belong to a given Competence. Thus, the planner sacrifices some search time in return for a considerably more compact memory representation. Fortunately this search is very fast, since the Competence Elements are stored within a single memory buffer with fixed length records. Testing shows the time taken by this searching was negligible in comparison with the plan cycle rate of the robot. Plan elements, senses and actions are referenced by unique numeric IDs, rather than by name. The memory storage of these IDs is defined within the code using the C++ `#typedef` preprocessor command, so that the width of these IDs can be configured at compile time, depending on the maximum ID value to be stored. This again saves memory in an environment where every byte counts. Consideration of stack usage is also important, and temporary buffers and similar structures are kept to a minimum to avoid stack overflow.

Fixed strings (for example error messages) and other data defined within programs are usually also stored within working memory. Within a microcontroller environment such as Arduino this is wasteful of the limited memory resource. This problem has been eliminated in the Instinct Planner implementation by use of AVR Libc functions [25] that enable fixed data to be stored in the much larger program (flash) memory. For code compatibility these functions have been replicated in a pass-through library so that the code compiles unaltered on non-microcontroller platforms.

2.4. *Instinct Testing Environment*

As a means to test the functionality of the Instinct Planner within a sophisticated debugging environment, we have an implementation of the planner within Microsoft Visual C++, and have tested the simulation of agents within a grid based world. The world allows multiple robots to roam, encountering one another, walls and so on. This simulator has also been extended with a graphical user interface to better show both the world and the real time monitoring available from within the plan [27]. However, the authors' primary use of the Instinct Planner has been within the R5 robot, as a means of investigating the real time debugging, and transparency of actual robots [28]. Building transparency into robot action selection can help users build a more accurate understanding of the robot, see Section 2.5 below.

The Instinct Planner code is not fundamentally limited to 255 plan elements, and will support much larger plans on platforms with more memory. In Microsoft Visual C++ for example, plans with up to 65,535 nodes are supported, simply by redefining the `instinctID` type from `unsigned char` to `unsigned int`.

2.5. *Instinct Transparency Enhancements*

The planner has the ability to report its activity as it runs, by means of callback functions to to a monitor C++ class. There are six separate callbacks monitoring the Execution, Success, Failure, Error and In-Progress status events, and the Sense activity of each plan element. In the Visual C++ implementation, these callbacks write log information to files on disk, one per robot instance. This facilitates the testing and debugging of the planner. In the Arduino robot, the callbacks write textual data to a TCP/IP stream over a wireless (wifi) link. A Java based Instinct Server receives this information, enriches it by replacing element IDs with element names, and logs the data to disk. This communication channel also allows for commands to be sent to the robot while it is running.

With all nodes reporting all monitor events over wifi, a plan cycle rate of 20Hz is sustainable. By reducing the level of monitoring, we reduce the volume of data sent over WiFi and plan cycle rates of up to 40Hz are achievable. In practice a slower rate is likely to be adequate to control a robot, and will reduce the volume of data requiring subsequent processing. In our experiments a plan cycle rate of 8Hz was generally used.

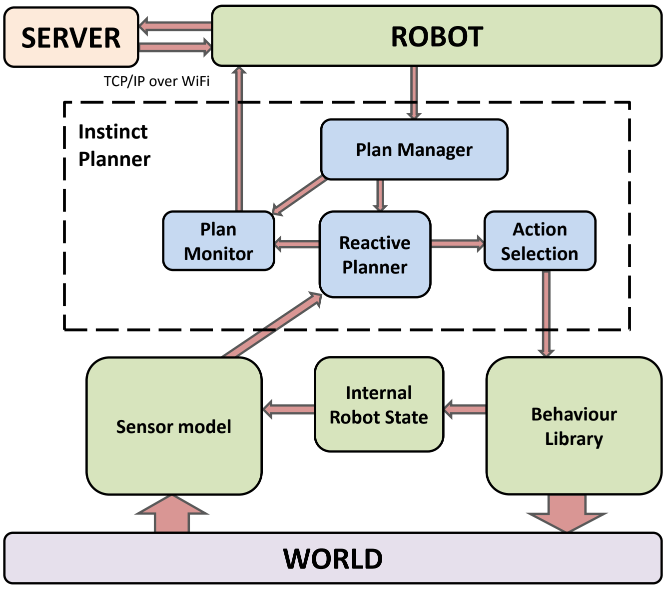


Figure 2: Software Architecture of the R5 Robot showing interfaces with the World and the Instinct Server. The Instinct Planner provides the action selection subsystem of the robot.

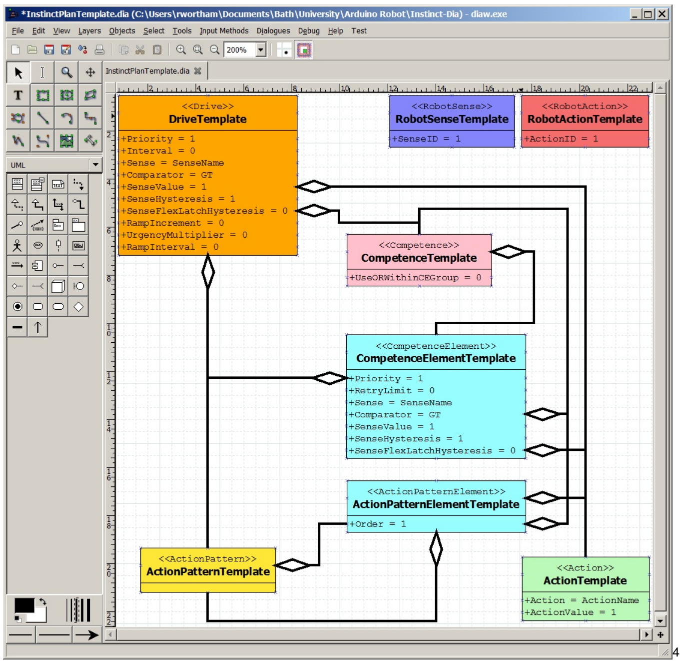
Figure 2 shows how the planner sits within the robot software environment and communicates with the Instinct Server.

2.6. *Instinct Command Set*

The robot command set primarily communicates with the planner which in turn has a wide range of commands, allowing the plan to be uploaded and altered in real time, and also controlling the level of activity reporting from each node in the plan. When the robot first connects to the Instinct Server, the plan and monitoring control commands are automatically sent to the robot, and this process can be repeated at any time while the robot is running. This allows plans to be quickly modified without requiring any re-programming or physical interference with the robot.

2.7. *Creating Reactive Plans with iVDL*

POSH plans are written in a lisp like notation, either using a text editor, or the ABODE editor [29, 30]. However, Instinct plans are written very differently, because they must use a much more compact notation and they use IDs rather than names for plan elements, senses and actions. We have developed the *Instinct Visual Design Language (iVDL)* based on the ubiquitous Unified Modelling Language (UML) notation. UML is supported by many drawing packages and we have developed a simple Python export script to allow plans to be created graphically within the Dia drawing tool [31]. The export script takes care of creating unique IDs and allows the plans to use named elements, thus increasing readability. The names are exported alongside the plan, and whilst they are ignored by the planner itself, the Instinct-Server uses this export to convert IDs back into names within the log files and interactive display.



<<Drive>>
DriveTemplate
+Priority = 1
+Interval = 0
+Sense = SenseName
+Comparator = GT
+SenseValue = 1
+SenseHysteresis = 1
+SenseFlexLatchHysteresis = 0
+RampIncrement = 0
+UrgencyMultiplier = 0
+RampInterval = 0

<<RobotSense>>
RobotSenseTemplate
+SenseID = 1

<<RobotAction>>
RobotActionTemplate
+ActionID = 1

<<Competence>>
CompetenceTemplate
+UseORWithinCEGroup = 0

<<CompetenceElement>>
CompetenceElementTemplate
+Priority = 1
+RetryLimit = 0
+Sense = SenseName
+Comparator = GT
+SenseValue = 1
+SenseHysteresis = 1
+SenseFlexLatchHysteresis = 0

<<ActionPatternElement>>
ActionPatternElementTemplate
+Order = 1

<<ActionPattern>>
ActionPatternTemplate

<<Action>>
ActionTemplate
+Action = ActionName
+ActionValue = 1

Figure 3: Instinct Plan element types and their relationship, shown within the DIA drawing tool.

Figure 3 shows the Instinct plan template within Dia. We use the UML class notation to define classes for the six types of element within the Instinct plan, and also to map the external numerical identifiers (IDs) for senses and robot actions to names. We use the UML aggregation connector to identify the connections between the plan elements. This can be read, for example, as "A Drive can invoke an Action, a Competence or an Action Pattern".

Figure 4 shows a plan for the R5 robot. At this level of magnification the element details are not legible, but this screen shot gives an impression of how plans can be laid out. This particular plan searches the robot's environment, avoiding objects and adjusting its speed according to the space around it. As it moves around it attempts to detect humans within the environment. The robot also temporarily shuts down in the event of motor overload, and it will periodically hibernate when not in open space to conserve battery power. Such a plan might be used to patrol hazardous areas such as industrial food freezers, or nuclear facilities.

The plan was designed and debugged within the space of a week. During the debugging, the availability of the transparency data logged by the Instinct Server was extremely useful, because mere observation of the robot's emergent behaviour is frequently insufficient to determine the cause of plan malfunction. For example, there were specific cases where the robot would unexpectedly hit an obstacle, rather than correctly avoid it. These cases could not easily be reproduced, but observing the robot together with the real-time debug trace during the error scenario quickly isolated the problem as an interaction between two Drives within the plan. The actual positioning of plan elements within the drawing is entirely up to the plan designer. Since Dia is a general purpose graphical editor, other symbols, text and images can be freely added to the file. This is useful at design time and during the debugging of the robot. It also provides an additional vehicle for the creation of longer term project documentation. We suggest that an in-house standard is developed for the layout of plans within a development group, such that developers can easily read one another's plans.

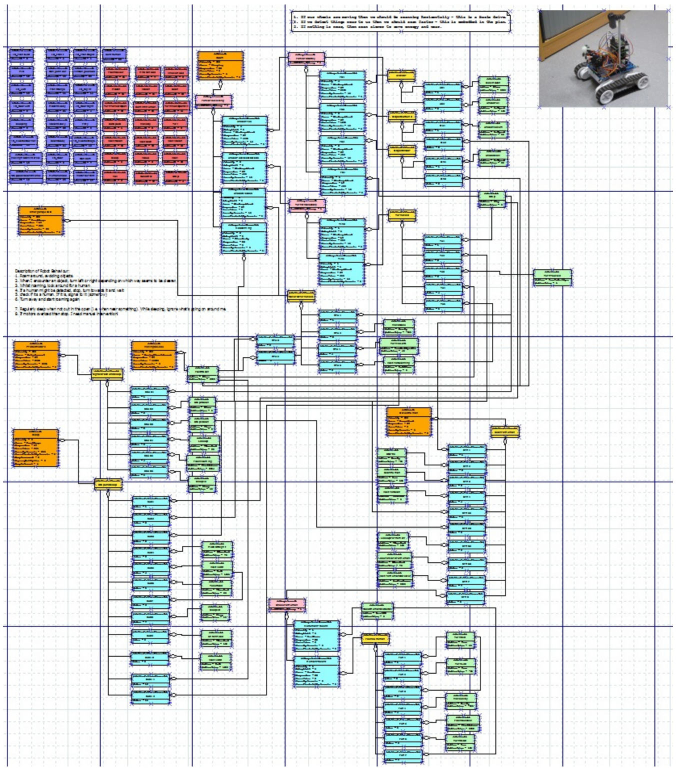


Figure 4: The plan used by the R5 robot to enable it to explore an environment, avoid obstacles, and search for humans. The plan also includes emergency behaviours to detect and avoid excessive motor load, and to conserve battery by sleeping periodically.

2.8. *Plan Debugging and Transparency*

Currently, work is underway within the Artificial Models of Natural Intelligence (AmonI) research group at the University of Bath[2] to create a new version of the ABODE plan editor [32]. This version directly writes Instinct plans, and also reads the real-time transparency data emanating from the Instinct Planner, in order to provide a real-time graphical display

---

[2] AmonI — http://www.cs.bath.ac.uk/ai/AmonI.html

of plan execution. In this way we are beginning to explore both runtime debugging and wider issues of AI Transparency.

3. CONCLUSIONS AND FURTHER WORK

In this article, we presented the Instinct planner, a reactive planner for designing behaviour-based AI, including but not limited to embodied agents such as the presented R5 robot. The Instinct planner is a major re-engineering and enhancement of Bryson's original work, focusing on an extremely tiny memory footprint and specifically low power CPUs. It is the first POSH-based planner that focuses on embedded systems, and allows deployment in practical real time physical environments such as our ARDUINO based maker robot. By using a very lean coding style and efficient memory management, we maximise both the size of plan that can be dynamically loaded, and the performance in terms of execution rate.

The transparency capabilities, novel to this implementation of POSH, provides the necessary infrastructure to deliver real time plan debugging. Work is currently underway to leverage this architecture with a real time visual debugging tool, initially to assist the work of reactive plan designers, but also as a research tool for the investigation of wider AI Transparency issues. The Visual Design Language (iVDL) is a novel representation of reactive plans, and we demonstrate that such plans can be designed using a standard open source drawing package and exported with a straightforward plug-in script. We envisage the development of similar plug-ins for other commercial drawing tools such as MICROSOFT VISIO.

Although primarily developed for physical robot implementations, the Instinct Planner has obvious applications in teaching, simulation and game AI environments. For example, the planner is embedded within a graphical agent simulation environment, the *Instinct Robot World* [27], where the planner is used to control simple agents within a grid based world. A previous POSH implementation has been used to control game agents in domains such as real-time strategy games (RTS) [33] or first-person shooters (FPS) [34], which shows further potential application domains for Instinct and related approaches. We would like to see the implementation of Instinct on other embedded and low cost Linux computing environments such as the RASPBERRY PI [35]. With more powerful platforms such as the PI, much larger plans can be developed and we can evaluate both the runtime performance of very large plans, and the design efficiency of iVDL with multi-user teams.

4. Acknowledgements

This work is funded by the University of Bath and in parts by EC FP7 grant 621403 (ERA Chair: Games Research Opportunities).

1. Nilsson NJ. Shakey the Robot. Tech. Rep.; SRI International; Technical Note 323; 1984.
2. Brooks RA. Intelligence Without Reason. In: Steels L, Brooks RRA, eds. *The Artificial Life Route to Artificial Intelligence: Building Embodied, Situated Agents*. Mahwah, New Jersey, USA: L. Erlbaum Associates. ISBN 0805815198, 9780805815191; 1991:25–81. URL: `http://idlebrain.yolasite.com/resources/Article-AI.pdf`.
3. Breazeal C, Scassellati B. Robots that Imitate Humans. *Trends in Cognitive Sciences* 2002;6(11):481–7.
4. Brooks RA. Intelligence Without Representation. *Artificial Intelligence* 1991;47(1):139–59.

5. Tinbergen N. The Study of Instinct. Oxford, UK: Oxford University Press; 1951. ISBN 9780198577225. URL: `https://books.google.co.uk/books?id=WqZNkgEACAAJ`.
6. Tinbergen N. The Curious Behavior of the Stickleback. *Scientific American* 1952;187(6):22–6. doi:`10.1038/scientificamerican1252-22`.
7. Tinbergen N, Falkus H. Signals for Survival. Oxford: Clarendon Press; 1970. URL: `http://books.google.co.uk/books?id=5LHwAAAAMAAJ`.
8. Bryson JJ. The Study of Sequential and Hierarchical Organisation of Behaviour Via Artificial Mechanisms of Action Selection. M. phil.; University of Edinburgh; 2000. URL: `https://www.cs.bath.ac.uk/~jjb/ftp/mphil.pdf`.
9. Bryson JJ. Intelligence by Design: Principles of Modularity and Coordination for Engineering Complex Adaptive Agents. Ph.d; MIT; 2001. URL: `https://dspace.mit.edu/bitstream/handle/1721.1/8230/50172578-MIT.pdf?sequence=2`.
10. Gurney KN, Prescott TJ, Redgrave P. The Basal Ganglia Viewed as an Action Selection Device. In: *Eighth International Conference on Artificial Neural Networks*. London, UK: Springer; 1998:1033–8. URL: `http://www.shef.ac.uk/~abrg/publications/icann98.pdf`.
11. Prescott TJ, Bryson JJ, Seth AK. Introduction. Modelling Natural Action Selection. *Philosophical transactions of the Royal Society of London Series B, Biological sciences* 2007;362(1485):1521–9. doi:`10.1098/rstb.2007.2050`.
12. Seth AK. The Ecology of Action Selection: Insights from Artificial Life. *Philosophical Transactions of the Royal Society of London Series B, Biological Sciences* 2007;362(1485):1545–58. doi:`10.1098/rstb.2007.2052`.
13. Gaudl SE, Bryson JJ. The Extended Ramp Goal Module: Low-Cost Behaviour Arbitration for Real-Time Controllers based on Biological Models of Dopamine Cells. In: *Computational Intelligence in Games (CIG) 2014 IEEE Conference On*. Dortmund, Germany; 2014:1–8. URL: `https://ieeexplore.ieee.org/stamp/stamp.jsp?tp={&}arnumber=6932887`.
14. Gaudl SE, Bryson JJ. The Extended Ramp Model: A Biomimetic Model of Behaviour Arbitration for Lightweight Cognitive Architectures. *Cognitive Systems Research* 2018;50:1–9. URL: `http://swen.fairrats.eu/research/papers/2018-CognitiveSystems-ExtendedRamp-Gaudl-Bryson-preprint.pdf`.
15. Rohlfshagen P, Bryson JJ. Flexible Latching: A Biologically-Inspired Mechanism for Improving the Management of Homeostatic Goals. *Cognitive Computation* 2010;2(3):230–41. doi:`10.1007/s12559-010-9057-0`.
16. Gaudl S. Building Robust Real-Time Game AI: Simplifying and Automating Integral Process Steps in Multi-Platform Design. Phd; University of Bath; 2016. URL: `http://swen.fairrats.eu/research/papers/Gaudl-PhD-RobustGameAI.pdf`.
17. Champandard AJ, Dunstan P. The Behavior Tree Starter Kit. In: Rabin S, ed. *Game AI Pro: Collected Wisdom of Game AI Professionals*. Game Ai Pro; A. K. Peters, Ltd.; 2013:72–92.
18. Ghallab M, Nau DS, Traverso P. Automated Planning: Theory and Practice. Morgan Kaufmann Series in Artificial Intelligence; Elsevier/Morgan Kaufmann; 2004. ISBN 9781558608566. URL: `https://books.google.co.uk/books?id=eCj3cKC{_}3ikC`.
19. Bryson J. Parallel-rooted, Ordered Slip-stack Hierarchy. http://www.cs.bath.ac.uk/˜jjb/web/posh.html; 2008. URL: `http://www.cs.bath.ac.uk/~jjb/web/posh.html`.
20. Champandard AJ. AI Game Development. New Riders Publishing; 2003.
21. Lim BY. Improving Understanding and Trust with Intelligibility in Context-Aware Applications. Ph.d; Carnegie Mellon University; 2012. URL: `http://repository.cmu.edu/cgi/viewcontent.cgi?article=1147{&}context=dissertations`.
22. Bryson JJ. A Role for Consciousness in Action Selection. In: Chrisley R, Clowes R, Torrance S, eds. *Proceedings of the AISB 2011 Symposium: Machine Consciousness*. York: SSAISB; 2011:15–.
23. Arduino . Arduino Website. https://www.arduino.cc/; 2016. URL: `https://www.arduino.cc/`.
24. Atmel Corporation . Atmel Studio Website. http://www.atmel.com/Microsite/atmel-studio/; 2016. URL: `http://www.atmel.com/Microsite/atmel-studio/`.
25. Atmel Corporation . AVR Libc Reference Manual. http://www.atmel.com/webdoc/avrlibcreferencemanual/; 2016. URL: `http://www.atmel.com/webdoc/avrlibcreferencemanual/`.
26. Wortham RH, Theodorou A, Bryson JJ. Robot Transparency: Improving Understanding of Intelligent Behaviour for Designers and Users. In: Gao Y, Fallah S, Jin Y, Lakakou C, eds. *Towards Autonomous Robotic Systems: 18th Annual Conference, TAROS 2017, Guildford, UK, July 1921, 2017*; vol. 10454. Berlin: Springer. ISBN 978-3-319-64106-5; 2017:274–89. URL: `http://link.springer.com/10.1007/978-3-319-64107-2`. doi:`10.1007/978-3-319-64107-2`.
27. Wortham RH, Bryson JJ. A Role for Action Selection in Consciousness: An Investigation of a Second-Order Darwinian Mind. *CEUR Workshop Proceedings* 2016;1855:25–30. URL: `http://ceur-ws.org/Vol-1855/`.
28. Wortham RH, Theodorou A, Bryson JJ. What Does the Robot Think? Transparency as a Fundamental Design Requirement for Intelligent Systems. In: *IJCAI-2016 Ethics for Artificial Intelligence Workshop*. New York, USA; 2016:URL: `http://opus.bath.ac.uk/50294/1/WorthamTheodorouBryson{_}EFAI16.pdf`.
29. Brom C, Gemrot J, Bida M, Burkert O, Partington SJ, Bryson JJ. POSH Tools for Game Agent Development by Students and Non-Programmers. In: Mehdi Q, Mtenzi F, Duggan B, McAtamney H, eds. *The Nineth International Computer Games Conference: AI, Mobile, Educational and Serious Games*. Dublin, Ireland; 2006:1–8. URL: `http://opus.bath.ac.uk/5366/`.
30. Theodorou A. ABOD3: A Graphical Visualization and Real-Time Debugging Tool for BOD Agents. *CEUR Workshop Proceeding* 2016;1855. URL: `http://opus.bath.ac.uk/53506/`.
31. Macke S. Dia Diagram Editor. http://dia-installer.de/; 2014. URL: `http://dia-installer.de/`.
32. Theodorou A, Wortham RH, Bryson JJ. Why is my Robot Behaving Like That? Designing Transparency for Real Time Inspection of Autonomous Robots. In: *AISB Workshop on Principles of Robotics*. Sheffield, UK; 2016:URL: `http://opus.bath.ac.uk/49713/`.
33. Gaudl S, Davies S, Bryson JJ. Behaviour Oriented Design for Real-Time-Strategy Games: An Approach on Iterative Development for STARCRAFT AI. *Foundations of Digital Games Conference* 2013;:198–205URL: `http://opus.bath.ac.uk/34386/`.
34. Partington SJ, Bryson JJ. The Behavior Oriented Design of an Unreal Tournament Character. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 2005;3661 LNAI:466–77. doi:`10.1007/11550617_39`.
35. Raspberry Pi Foundation . Raspberry Pi Website. https://www.raspberrypi.org/; 2016. URL: `https://www.raspberrypi.org/`.