Advances in Cognitive Systems 2 (2013) 257-275 Submitted 12/2012; published 12/2013

# A Cognitive Architecture for Autonomous Robots

**Adam Haber** ADAMH@CSE.UNSW.EDU.AU
**Claude Sammut** CLAUDE@CSE.UNSW.EDU.AU
ARC Centre of Excellence in Autonomous Systems, School of Computer Science and Engineering, University of New South Wales, Sydney, NSW 2052, Australia

## Abstract

We introduce Mala, a cognitive architecture intended to bridge the gap between a robot's sensorimotor and cognitive components. Mala is a multi-entity architecture inspired by Minsky's *Society of Mind* and by experience in developing robots that must work in dynamic, unstructured environments. We identify several essential capabilities and characteristics of architectures needed for such robots to develop high levels of autonomy, in particular, modular and asynchronous processing, specialised representations, relational reasoning, mechanisms to translate between representations, and multiple types of integrated learning. We demonstrate an implemented Mala system and evaluate its performance on a challenging autonomous robotic urban search and rescue task. We then discuss the various types of learning required for rich and extended autonomy, and how the structure of the proposed architecture enables each type. We conclude by discussing the relationship of the system to previous work and its potential for developing robots capable of long-term autonomy.

## 1. Introduction

Our aim is to build a cognitive architecture that is suitable for a robot that must operate, over a long period of time, in a realistic, unstructured and dynamic environment. That is, we try to minimise the assumptions we make about the nature of the robot's environment. We do not have a map of the surroundings in advance. We do not have any easily identifiable landmarks known before the start of the mission. Objects are not conveniently coloured or shaped to make recognition and manipulation easier and they may move, sometimes quickly. The terrain is not assumed to be flat, but may be rough and uneven, necessitating complex locomotion. These are the kinds of conditions that are encountered in real applications of robots, such as mine automation, underwater monitoring, or health care. In these unstructured environments, most of the robot's hardware and software systems must be devoted to perception and actuation since the associated problems are usually very challenging and expensive, in terms of both hardware costs and computation. The algorithms that drive these computations are necessarily specialised. For example, simultaneous localisation and mapping (SLAM) is handled, almost always, by creating a probabilistic estimate of the robot's pose in relation to its surroundings and stitching together successive estimates to create a global map (Thrun, 2002). Perception is also usually handled by statistical estimation of the occurrences of certain high-dimensional features to identify objects in scenes. Locomotion, particularly for a legged

or segmented robot traversing rough terrain, also requires substantial, specialised computational resources to calculate feasible trajectories for the robot's motors. The same holds for manipulation.

Langley (2012) has asserted that integrated cognitive systems need not include such specialised machinery, since the intelligent abilities of humans or machines are decoupled from their ability to perceive and manipulate the environment. This amounts to the assumption that perceptual and motor mechanisms are not involved in higher-level reasoning. In humans, at least two processes belie this assertion. These are the process of mental imagery, in which high-level cognition makes use of perceptual and motor systems to determine feasibility of actions before performing them, and of top-down perception, in which abstract concepts and contexts influence which aspects of the world are perceived. These processes are equally crucial for robotic systems, which frequently utilise metric representations and path planning algorithms to 'visualise' whether a path to a goal is possible, or which select the type of low-level perceptual processing to perform in a manner sensitive to high-level context. Thus, to pursue human-level intelligence in robotic systems, perception and motor skills cannot be ignored (Sammut, 2012). Although it may be possible, in principle, to implement all the required algorithms for perception and actuation using a uniform representation, in practice this will introduce many inefficiencies. For example, it would be very inefficient, as well as inconvenient, to perform large-scale matrix manipulation, directly, in a rule-based system. It does, however, make sense to call specialised modules from a high-level system that serves as glue to join many components and performs tasks at a cognitive level.

At present there is a gap between robot software architectures and cognitive architectures. Most robotic architectures, such as ROS (Quigley, Conley, & Gerkey, 2009), follow a modular model, providing a communication mechanism between multiple nodes in a potentially complex and *ad hoc* network of software components. Cognitive architectures (Laird, 2012; Langley & Choi, 2006; Anderson, 2007) usually emphasise uniformity in representation and reasoning mechanisms. Almost all fielded robotics applications take the *ad hoc*, modular approach to controlling the robot. The drawback of this approach is that a custom-designed system is less retaskable, and less flexible than a structured framework capable of adapting its own control systems through introspection. However, the modular, loosely coupled approach taken by these architectures is ideally suited for integrating diverse representations in a flexible manner. Further, the asynchronous nature of such systems helps to process incoming sensory information as rapidly as possible, facilitating the highest possible degree of reactivity. Thus, such a processing model is ideal for integrated robotic software frameworks, and we adopt it here.

Although asynchronous subsymbolic processing is necessary for an autonomous robot, it is not sufficient, in general, to solve complex problems, reason introspectively, or explain its actions to human operators. Further, systems capable of abstract representation and planning are capable of being adapted to new domains much more easily, which is essential to produce a retaskable robotic system (Sammut, 2012). Thus, symbolic, relational reasoning is an essential capability for integrated robotic architectures. However, architectures that integrate relational reasoning with subsymbolic processing must provide mechanisms to translate processing content between representations.

Our initial statement was that we wish our robot to be able to operate in an unstructured and dynamic environment over a long period of time. This is sometimes called *long-term autonomy*. In its current usage (Meyer-Delius, Pfaff, & Tipaldi, 2012), this term generally means continuing to

perform a task, usually navigation, despite changing conditions. Again, we try to make fewer assumptions. We do not constrain the robot to perform only one kind of task, but wish it to be capable of performing a variety of tasks, under changing environmental conditions, and in the face of possible changes to the robot itself. Adaptation to changing conditions necessarily entails learning, either from failures, experimentation, or demonstration. Because we adopt a heterogeneous structure, with different modules using different representations and algorithms, different learning methods are also required. For example, a low-level skill such as locomotion requires adjusting motor parameters, so numerical optimisation may be appropriate, whereas learning action models for high-level task planning is more likely to be accomplished by symbolic learning. Thus, the cognitive architecture should be able to support different kinds of learning in its different components.

We argue that an architecture that can provide this type of long-term autonomy must possess several qualities, particularly, specialised representations, loosely coupled asynchronous processing components, relational reasoning, mechanisms to translate between task-level and primitive representations, and integrated multi-strategy learning. This paper presents Mala, an architecture that possesses these characteristics. We first describe the components of Mala and how information flows between them, and then present an application of the architecture to control an autonomous robot for urban search and rescue, which demonstrates all these capabilities with the exception of learning. We then discuss the architecture's relationship to earlier work. Finally, we describe in detail the learning mechanisms that we plan to embed at multiple levels and within multiple components of the architecture, how they are enabled by its structure, and how we plan to implement them.

## 2. The Mala Architecture

In this section we describe the operation of the Mala architecture in detail, beginning with the communication mechanisms and global representation. We then discuss internal representations used within and information flow between individual components. In particular, we outline how raw sensory information is abstracted into a world model, that can be used to generate task-level actions, and how they can then be implemented by a physical robot. At the finest level of detail, we describe an implemented system and demonstrate the architecture's behaviour on a task in urban search and rescue.

### 2.1 Architectural Components

Mala consists of two types of modules: blackboards, which provide storage and communication between other components, and agents, which perform all of the systems computation. We describe each of these in turn.

#### *2.1.1 Blackboards*

Blackboards perform several important functions in the architecture. They serve as a communication mechanism between the agents that sense, plan, and act. They are also the primary storage mechanism for short-term and long-term memory. Objects on a blackboard are represented by

frames (Roberts & Goldstein, 1977). Agents interact with each other indirectly by posting frames to and receiving them from a blackboard. The system may consist of more than one blackboard and associated agents. The blackboard and frame systems described here have evolved from the MICA blackboard system (Kadous & Sammut, 2004a) and the FrameScript programming language (McGill, Sammut, & Westendorp, 2008), which were originally designed and used to implement a smart personal assistant in a ubiquitous computing environment (Kadous & Sammut, 2004b).

#### *2.1.2 Agents*

Connected to the blackboard are the components that perform all computational functions of the system. Following Minsky's (1988) original terminology, we call these *agents*. In contrast to other cognitive architectures, which often make commitments to particular representations of knowledge and reasoning algorithms, there is no restriction on the internal workings of an agent. Thus each agent's implementation uses the most appropriate method for its particular task.

Agents are grouped into several categories. Low-level perception agents operate directly on raw sensor data to extract useful features. Higher-level perception agents operate on these features, and fuse them into more abstract representations such as first-order predicates describing objects and their poses. These representations then form the input to further high-level perception agents that update the architecture's current knowledge of the world. Goal generation agents encode the drives and objectives of the system, and operate upon the description of the world state produced by perception agents. Action generation agents include action planners, constraint solvers, motion planners, and motion controllers. Together, these agents provide the necessary processing to generate task-level plans to achieve goals and to execute the behaviours that implement these plans.

#### *2.1.3 Communication*

When an agent connects to a blackboard it requests to be notified about certain types of objects. The blackboard then activates the agent when objects of that type are written to the blackboard. The active agent performs its task and posts its results to the blackboard. A new object written to the blackboard may trigger another agent into action and this process continues. This is somewhat similar to the way a rule-based system operates when rules update working memory, triggering further rule activation. However, agents are larger computational units than rules. Another difference is that there is no explicit conflict resolution mechanism, as all agents that are enabled by the current state of the blackboard may activate concurrently. Agents communicating indirectly through the blackboard do not have to be aware of each other. The isolation of agents enforced by the blackboard makes it possible to introduce new agents or replace existing ones without affecting other agents. Furthermore, indirect communication allows the information going through the blackboard to be observed by other agents, including ones that learn from the succession of blackboard events.

### 2.2 Architectural Framework and Implementation

Now we can describe in more detail the functional arrangement of agents within Mala. The collective action of the perception agents results in increasingly abstract representations on the blackboard. The collective action of the action generation agents results in converting task-level plans into mo-

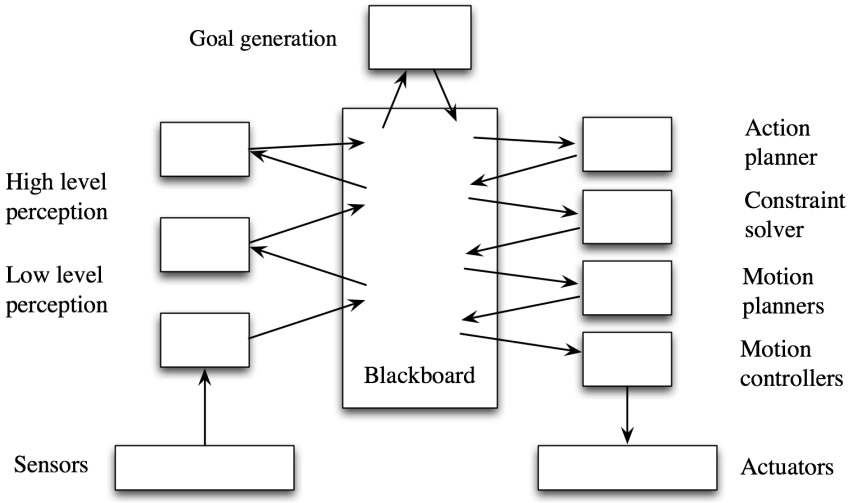


*Figure 1.* Information flow within Mala. Smaller boxes indicate agents or groups of agents, while arrows indicate posting and listening to frames, mediated by the blackboard. Abstraction increases as information flows up the stack of perceptual agents on the left, and decreases again as task-level action plans are made operational by the action generation agents on the right.

tor commands. The agent hierarchy, shown in Figure 1, that is constructed by the interactions between these types of agents represents one of the theoretical commitments of the architecture. This hierarchy is related to Nilsson's (2001) triple-tower architecture. Although the types of agents and their interactions are constant, the specific agents that populate each group depend on the domain to which the system is being applied. Another application-specific concern is the number of blackboards, which may increase with the complexity of the system, to reduce unnecessary communication between unrelated agents, and to provide the possibility of specialised representations being used among groups of agents.

To demonstrate the architecture, we have implemented a control system for a simulated autonomous mobile robot performing urban search and rescue (USAR). This task requires exploration within a complex environment, multi-objective search, challenging perceptual tasks, and obstacle avoidance. We perform experiments in a test arena closely modelled on the RoboCup Rescue Robot competition. Experiments are conducted using a simulation environment built upon the open source jMonkeyEngine3 game engine (Vainer, 2013), described in detail by Haber et al. (2012). The robot is an autonomous wheeled robot modelled on the "Emu", which has competed in RoboCup Rescue (Sheh et al., 2009). The simulated Emu and its sensor outputs are shown in Figure 2, along with a screen shot of the simulation environment. The robot is tasked with exploring a complex, relatively unstructured, but largely static, disaster site. The environment consists of a maze-like collection of corridors, various obstacles, with several human-shaped "victims" scattered through the arena. The goal of the robot is to navigate through the arena, exploring the maximum amount of territory, finding the maximum number of victims, and return to the start point within a fixed time limit. We now detail the representations, inputs, and outputs of each type of agent in the architecture, first de-

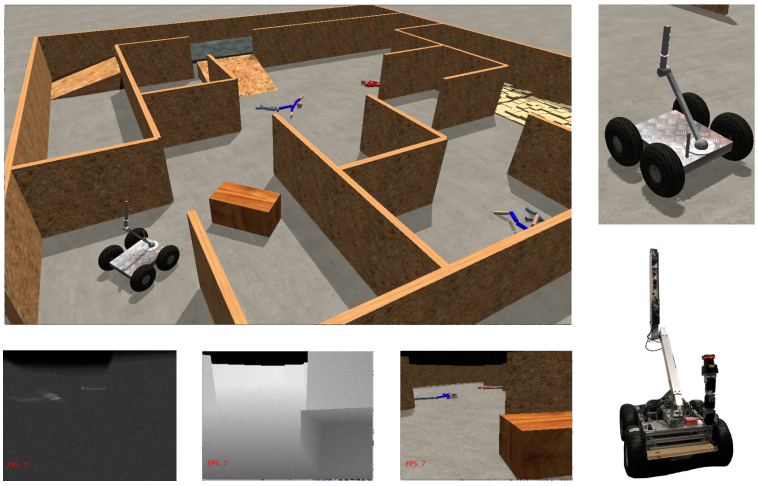

*Figure 2.* Rescue robot simulation. Clockwise from top left: simulated Emu robot exploring an environment confronted with a block obstructing its path; simulated Emu robot; the real Emu robot; simulated colour, depth, and thermal camera images.

scribing the function that the general architecture commits it to, and then using the specific agents required for the autonomous rescue control architecture as concrete examples.

Each sensor is controlled by an agent that posts raw data to the blackboard to be processed. The sensor agents present in the implemented architecture are: a colour video camera, depth camera, thermal camera, all of which have a resolution of $640 \times 480$ pixels and frame rates of approximately 15 Hz, a 180 degree laser range finder with 0.3 degree angular resolution and 30m maximum range, and wheel encoders. Gaussian noise of an amplitude based on actual sensor performance is added to all simulated sensor measurements.

As discussed in the introduction, an essential aspect of intelligent robotics is to process multi-modal sensor data asynchronously to obtain information about the environment. Thus, in Mala, each low-level perceptual agent performs a specialised computation. In the implemented system, there are several such agents. A simultaneous localisation and mapping (SLAM) agent operates on laser data to produce a map of the environment as the robot explores. This agent incorporates a modified iterative closest point algorithm for position tracking and an implementation of the FastSLAM algorithm for mapping (Milstein et al., 2011), and posts frames containing the latest occupancy grid map as it becomes available. In the rescue arena, victims are represented by simple manikins with skin-coloured heads and arms, which may move. Since the robot's task is to identify and visit victim locations, the implemented architecture incorporates an agent for victim finding, which detects the colour and warmth of human skin. The victim detection agent agent scans the colour camera image for skin coloured pixels based on an adaptive hue thresholding algorithm (Bradski & Kaehler, 2008), and matches them to pixels that have approximately human skin temperature in the thermal camera image. For each pixel in the heat image, a simple absolute difference from a reference temperature determines skin temperature. The agent then uses information from the depth camera to determine the location of the detected victim, and posts a frame containing this information to the blackboard.

When the results of low-level perception computations are posted to the blackboard, they may trigger further processing by higher-level perceptual agents, such as classification, recognition, or abstraction into more symbolic representations. One high-level perception agent in the implemented system is a victim assessment agent that stores a database of known victim locations and compares each new victim detection frame with this database to determine whether it is a previously unknown victim. The other high-level perception agent is the obstacle recognition agent. In the present implementation, this agent retrieves the identity and location of obstacles directly from the simulation whenever obstacles are in view. This is not a significant simplification of the problem, since software components to recognise objects from point cloud data have been developed independently of this work (Farid & Sammut, 2012).

The highest-level perception agents build an abstract state description to explicitly represent the complex properties and relationships that exist between objects in the world, expressed in first-order logic. The exact poses of individual objects are not always represented explicitly in this manner. For example, an abstract state defined by `on(book,table)` describes many different primitive states in which the book is in different positions on the table. In the rescue system, a simple topological mapper defines regions of space and labels them as explored or unexplored. This produces an abstract description of the robot's spatial surroundings.

Goal generator agents produce desired world states that, if achieved, would result in progress towards the system's global objective. The goal generation agent uses a simple set of rules for generating first-order world states that the agent should seek to make true and places frames containing these predicates on the blackboard. The constructed first-order world state and current goal provide input to an action planner agent, along with a collection of task-level action models. The planner then attempts to solve this classical planning problem, producing a sequence of actions the robot can then attempt to execute. The task planner agent in the rescue control system uses an implementation of the Fast Forward planner (Hoffmann & Nebel, 2001), which places action frames on the blackboard when it has constructed a plan.

All systems that integrate abstract reasoning with robotic execution must provide a mechanism to translate from logical to metric representations. We adopt the method introduced by Brown (2011), who showed that actions can be made operational using a combination of constraint solving and motion planning. A constraint solver agent enables each task-level action to be implemented by the robot by treating the effects of the action as a constraint satisfaction problem and solving for a pose of the robot that satisfies this problem. Motion planner agents then search for a trajectory through metric space that can achieve this pose. This trajectory is then followed by motion controller agents, which implement reactive control of actuation. It is worth noting that, because of the multiplicity of sensors and actuators that most robots possess, Mala specifies that there should be multiple agents of each type, each encoding a specialised algorithm to implement the processing required for each sensor or actuator.

The constraint solver agent, implemented using the constraint logic programming language ECLiPSe (Apt & Wallace, 2006), accepts an action frame as input. The constraints are extracted from the spatial subgoals in the effects slot of the action frame, shown in Figure 1. Each predicate in the effects of an action specifies a constraint on final poses of the objects involved. The constraint solver produces a metric pose that satisfies the spatial constraints. This is placed in a metric goal

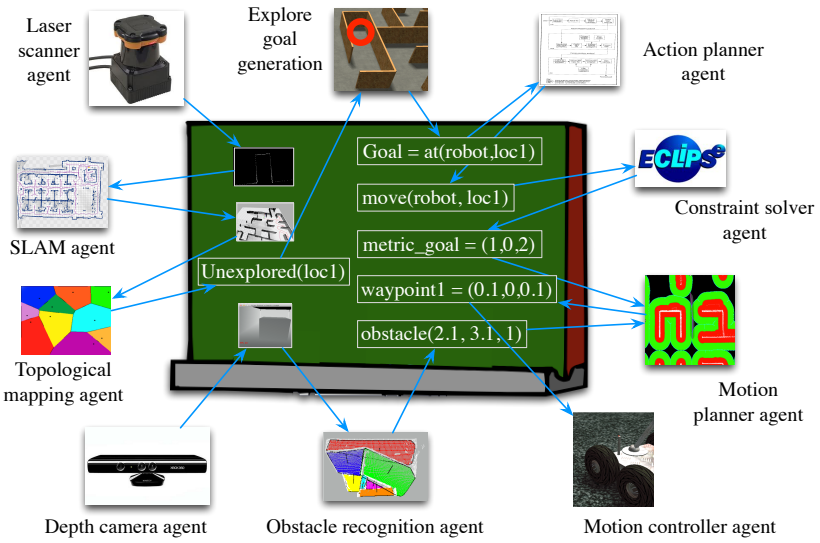


*Figure 3.* Agents communicate by posting frames to a blackboard, building an abstract spatial model from sensor data, generating a goal, and constructing a plan to achieve it. Task-level actions are translated to motor actions via a constraint solver that provides parameters to a path planner and reactive controller.

frame that contains a metric goal pose for the robot or its actuators and that can be input to a motion planner. The motion planner implemented in the current system uses an A* search to find drivable trajectories to reach the goals specified by metric goal frames, which it posts to the blackboard as trajectory frames that contain a list of waypoints. The drive controller agent provides reactive obstacle avoidance using the sensor data from the laser range finder. It is an implementation of the "almost-Voronoi" controller described in detail by Sheh et al. (2009).

### 2.3 System Operation

We now illustrate how these agents interact to perceive the world and produce goal directed robot behaviour by describing the flow of blackboard frames involved in trying to reach a location for exploration. Figure 3 shows the succession of objects posted to the blackboard by the agents during this scenario.

1. Placed at the arena entrance, the robot begins its exploration of an unknown environment. An agent that controls a scanning laser range finder writes a frame on the blackboard that contains range data.
2. This frame triggers the SLAM agent, a listener for laser scans, to update its occupancy grid map of the corridor, which it posts to the blackboard.
3. A topological mapper listens for the occupancy grid map and splits the metric spatial data into regions, each of which it labels as unexplored, such as region `loc1` in the figure. The topological map agent posts an unexplored location frame to the blackboard.

4. The explore goal generation agent listens for unexplored location frames and, in response, generates a goal frame `at(robot,loc1)`.
5. The motion planner agent listens for goal frames and occupancy grid maps; when both appear on the blackboard, it determines that a path to the goal is possible.
6. The task planner agent listens for goal frames, determines that the preconditions of `move(robot, loc1)` are satisfied (`path_to_goal = true`), and that its post conditions will achieve the current goal (`at(robot,loc1)`). As a result, it places `move(robot, loc1)` on the blackboard.
7. The constraint solver agent listens for action frames and converts the action's effects predicates into constraints. After the constraints have been enforced by the solving process, it selects a pose within the resultant region of space and posts the pose as a metric goal frame to the blackboard.
8. Concurrently, an object recognition agent analyses depth-camera frames and posts the locations of any obstacles to the blackboard. The SLAM agent listens for obstacles and incorporates them into the map.
9. The motion planner listens for metric goal frames and using the SLAM map, plans a path that avoids the obstacle and posts a trajectory frame containing a list of primitive waypoints.
10. Finally, the reactive drive controller agent accepts the trajectory and operates the robots motors to follow it.

We have now described the components of Mala and their interaction, and seen how autonomous robot control is achieved through their asynchronous operation. Now, we move on to demonstrate an implementation of Mala controlling an autonomous mobile rescue robot robot.

## 3. Experimental Demonstration and Evaluation

We tasked the implemented system with exploring five randomly generated environments, one of which is shown in Figure 4 (right). Within each environment there are five victims, placed in random locations, that are the objects of the search. Along with these victims, there are also three obstacles that can trap the robot if it does not avoid them. These are grids of blocks of random heights known as *stepfields*, intended to simulate the difficulty of driving over rubble or complex terrain. Each environment is 256 square metres in area, and the robot has 15 minutes to explore as much as possible before it must return to the start or stop moving, to model the finite battery life of real robots. The goal of the robot is to produce an accurate map of the environment, labelling the location of victims and visiting as many of their locations physically as possible. To assess the performance of the system, we record the number of victims visited and the fraction of the environment successfully explored and mapped. Table 2 shows the performance of the system on the five environments, while Figure 4 (left) shows the generated map and path of the robot during a single run.

This demonstration of the architecture in operation highlights several of the characteristics essential for robotic cognitive architectures identified in the introduction. First, asynchronous processing of agents enables a high degree of reactivity by allowing interrupts of different types to propagate to various levels of the architecture. At the lowest level of reactivity, in the event that

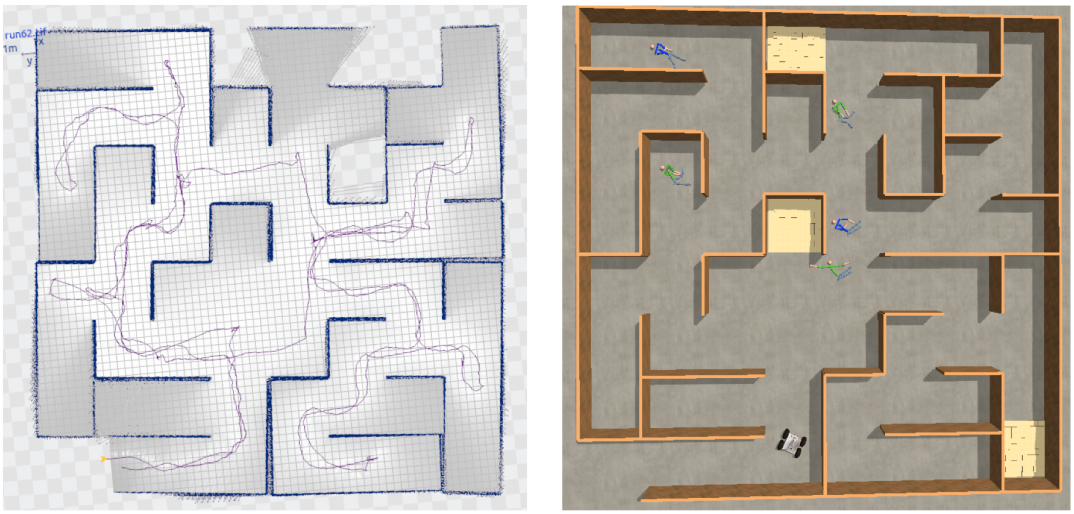

*Figure 4.* Trace of a demonstration run of the system on Map 1, with the robot beginning in the bottom left hand corner. Left: The map generated and robot's path during the run. Right: Overhead view of the run that provides ground truth, with victims visible as blue and green figures, and with stepfield obstacles as lighter areas of the floor.

laser data indicates that the robot is too close to a wall, the drive controller agent listens for laser data directly and can react rapidly to keep the robot away from obstacles. An example of higher level interrupts occurs as the robot detects obstacles while navigating towards a goal and the system responds by replanning its route around the obstacle, as described in steps 8 and 9 of the process described in Section 2.3. The highest level of interrupt is apparent on inspection of the trace of blackboard frames in Figure 1. While navigating towards an exploration goal, a victim is detected and a detection frame is placed on the blackboard. After the victim assessment agent determines that the victim is previously unknown, the goal generation agent places a goal frame on the blackboard, which causes the architecture to switch goals and attempt to reach the victim's location.

As noted earlier, the low-level perception agents in this implementation are specialised for interpreting camera frames and laser scans. The ability of the architecture to recognise victims and obstacles, as well as to explore and map the environment, demonstrates how these components can be integrated in a robotic architecture. Autonomous robots require the ability to generate plans that solve novel and complex problems. This capability is particularly important for retaskability and introspection (Sammut, 2012). The operation of the action planner in constructing plans to achieve goals in the rescue environment demonstrates this ability, but, for robots to use symbolic reasoning, abstract plans must be made operational by translating high-level actions into motor primitives. In Mala, this is done by a novel combination of constraint solving and motion planning. This is particularly important if action models are acquired online, which is necessary for robots that learn to perform new tasks. The translation process is explicitly demonstrated in the trace of frames from a task-level action model to a metric goal representation shown in Figure 1.

*Table 1.* Abridged trace of frames on the blackboard during the run shown in Figure 4. The frames illustrate one mechanism for interrupts in the architecture; as the system navigates towards an exploration goal, a victim is detected, and it is determined to be previously unknown. A goal is generated to reach the victims location, and the system attempts to reach it.

```
Emu_Robot_Camera executes, and writes to
blackboard:
    Emu_Robot_Cam_Img4212 isa [cam_img] with
        img_data: reference_cam_img_file,
        width: 640,
        height: 480;

Victim_Detect is a listener for [cam_img]
frames. Victim_Detect executes, and writes to
blackboard:
    victim_detection1204 isa
[victim_detection] with
        width: 98,
        height: 98,
        screen_y: 128,
        screen_x: 228,
        location: point(11.55, 1.22, 15.11);

Victim_Assess is a listener for
[victim_detection] frames. Victim_Assess
executes, and writes to blackboard:
    victim_assessment1430 isa
[victim_assessment] with
        id: 1,
        visited_already: false,
        location: point(11.55, 1.22, 15.11);

GoalGenerator is a listener for
[victim_assessment] frames. GoalGenerator
executes, and writes to blackboard:
PDDL1803 isa [PDDL] with
    goal:
[near(robot,victim1), location(victim1,
point(11.55, 1.22, 15.11))],
    init_state:
    [near(robot,explore_goal3),
location(explore_goal3, point(5.31, 0, 12.62)),
location(robot, point(4.92, 0.41, 13.39))];

Planner is a listener for [PDDL] frames. Planner
executes, and writes to blackboard:
action8072 isa [action] with
    action: move(robot,explore_goal3, victim1),
    objects: [
        [explore_goal3, point(5.31, 0, 12.62)],
        [victim1, point(11.55, 1.22, 15.11)],
            [robot, point(4.92, 0.41, 13.39)]],
        effects: near(robot, victim1);

Solver is a listener for [action] frames. Solver
executes, and writes to the blackboard:
    metric_goal5650 isa [metric_goal] with
        goal: point(10.89, 0, 14.43);

Motion is a listener for [motion_planner_goal]
frames. Motion_Planner executes, and writes to
the blackboard:
    trajectory2251 isa [trajectory] with
        waypoints: [point(5.65, 0, 13.10),
                point(5.78, 0, 13.05),
                ...,
                point(10.89, 0, 14.43)];

Driving_Controller is a listener for [trajectory]
frames. Driving_Controller executes..
```

The results of this demonstration, summarised in Table 2, illustrate that the control architecture constructed by these agents is able to explore the majority of an unknown rescue arena, and to identify and reach almost all victims, while avoiding obstacle hazards that can trap the robot. However, during these demonstration runs, several limitations emerged. In particular, when an interrupt caused the switching of one goal for another, as described in the text above and detailed in Figure 1, the current system simply discards the older goal. This leads to a somewhat greedy exploration strategy that can lead to some areas of the map being neglected, especially when all victims are located in a single region. This was the case for Map 5, which caused a lower than expected coverage of the rescue environment. This can be corrected by pushing the previous goal onto a stack if it remains consistent with the world model state after the interruption. A second problem the system faced was that, once a victim was reached, it was then identified as an obstacle to prevent the robot from driving over it. If the victim lies in the centre of a corridor, this can make areas of the map inaccessible to the robot. Although this meant that the robot was often unable to map the

*Table 2.* Performance on the autonomous rescue task for five randomly generated environments, including the one shown in Figure 4.

| Map | Victims Found | Environment Mapped |
|---|---|---|
| 1 | 5.0 | 94 % |
| 2 | 5.0 | 86 % |
| 3 | 4.0 | 81 % |
| 4 | 3.0 | 84 % |
| 5 | 5.0 | 70 % |
| **Average** | 4.4 | 85 % |

complete environment, it is a problem that is common to real rescue and does not indicate a flaw in the architecture's performance.

## 4. Comparison to Other Architectures

Two architectures, Soar (Laird, 2012) and ACT-R (Anderson, 2007), have been particularly influential in research on cognitive systems. Both are production systems that emphasise the role of a shared workspace or memory and serial models of processing. ICARUS is a more recent cognitive architecture that, like Soar[1] and ACT-R, commits to a unified representation for all its knowledge (Langley & Choi, 2006). However, unlike Soar and ACT-R, whose focus is on high level, abstract cognition, ICARUS is explicitly constructed as an architecture for physical agents and thus supports reactive execution. These cognitive architectures provide accounts of higher cognitive functions such as problem solving and learning, and have shed light on many aspects of human cognition. However, as discussed earlier, some aspects of robotics are inconvenient or inefficient to implement within these frameworks. Thus, in parallel with the development of cognitive architectures, robotic architectures have attempted to implement cognitive behaviour and processing on robotic platforms.

The most successful early example of a robotic architecture was developed as part of the Shakey project (Nilsson, 1984). Shakey constructed a complete, symbolic model of its environment which was used as the basis for the composition of plans to achieve goals. However, the improved responsiveness of systems developed by the behaviour-based robotics community demonstrated the utility of tight sensor-actuator control loops for reactivity (Brooks, 1986).

Hybrid architectures (Gat, 1997), which include both types of processing in a layered configuration, are now common. In this arrangement, processes that operate on a slow timescale populate the highest, deliberative, layer, while fast, responsive processes are relegated to the reactive layer. Typically, such architectures have three layers of abstraction, including a sequencing layer that interfaces between deliberative and reactive subsystems.

The hybrid paradigm has proven a powerful architectural guide for constructing robotic architectures, demonstrating that, for a robot to be flexible and reliable, both deliberative and reactive

1. Recent versions of Soar, such as that reported by Nason and Laird (2005), abandon this commitment.

processes are necessary. However, as new sensors, actuators, and representations are developed, architectures are increasingly forced to incorporate highly specialised processing of sensory data and action schema. This characteristic enforces the inclusion of multiple, concurrent channels for sensory input, and thus leads to the modular, asynchronous computation model that has been adopted by several recent cognitive/robotic architectures. Since Mala has much in common with these newer systems, we now discuss its relationship to the most similar designs, the CoSY Architecture Schema (Hawes & Wyatt, 2010), DIARC (Scheutz, Schermerhorn, & Kramer, 2007), and T-REX (McGann et al., 2008).

The CoSY Architecture Schema (CAS) shares many features with Mala, such as the presence of concurrent processing components that utilise shared memories for communication. The framework adopts a communication paradigm in which each component specifies the particular changes to working memory elements about which it should be notified. This is equivalent to our listen requests, in which an agent specifies the type of blackboard object it requires. More recent iterations of CAS have integrated a novel mechanism for component execution management using planning. This allows a continual planner to treat each subarchitecture as a separate agent in a multi-agent planning problem (Hawes, Brenner, & Sjöö, 2009). This is closely related to Mala's use of a planner to generate task-level actions that may be implemented by different agents. However, our planner uses a classical STRIPS formalism to construct a complete plan and does not reason about incomplete knowledge, while CAS uses *collaborative continual planning* (Brenner & Nebel, 2009). This is a strength, as it allows CAS to interleave planning with acting and sensing by postponing those parts of the planning process that concern currently indeterminable contingencies.

DIARC (Scheutz, Schermerhorn, & Kramer, 2007), an early example of modular robotic architectures, incorporates concurrent, autonomous processing modules, each using specialised representations. The perceptual subsystems perform low-level processing and abstraction into more meaningful content in a manner similar to Mala. However, communication within DIARC is point to point, with direct links between computation nodes, that are implemented using the ADE middleware (Scheutz, 2006). Although this is potentially a more efficient form of connection as it relaxes the restriction that communication objects must pass through the blackboard intermediary, any potential for online reconfiguration is greatly reduced, because components must be explicitly aware of each other's existence at run time. Further, constructing introspective monitoring processes is made more difficult, as each node-node pair must be monitored independently. An important difference between DIARC and Mala involves behaviour generation through planning. DIARC relies on an action interpreter that executes action scripts analogous to the STRIPS plans used in Mala. For each goal that a DIARC agent maintains, a corresponding script interpreter manages the execution of events, analogous to task-level actions, to achieve a goal. However, if a DIARC agent encounters a goal for which it does not have a premade script, it has no mechanism to generate a script by concatenating action primitives.

A third architecture, T-REX (McGann et al., 2008), is designed to integrate automated planning with adaptive execution in the tradition of deliberative-reactive hybrids. The framework is composed of a set of encapsulated control programs known as *teleo-reactors*. Although this decomposition lends T-REX a modular character, it does not exhibit the extensive modularity of systems like CAS that are based on networks of computation nodes. In addition, there is more consistency

about the internal structure of T-REX reactors, as they adopt a common representation. Unlike heterogenous architectures such as Mala, CAS, and DIARC, all reactors run the same core algorithm to synchronise with the rest of the architecture. This reflects the T-REX claim that behaviour should be generated by similar deliberative processes operating on multiple time scales. Communication between nodes is highly structured, consisting of only two types of objects, *goals* and *observations*, that are expressed in a constraint-based logical formalism. This formalism provides a more expressive framework than the STRIPS representation used in Mala, with the ability to reason about temporal resources and conditional effects. This lets T-REX produce effective plans for time-critical missions. Reactors are organised hierarchically according to their time scale in a manner reminiscent of hybrid three-layer architectures (Bonasso et al., 1997).

We can summarise Mala's relationship to previous architectures in three main points:

- Like other node-based architectures, Mala differs from traditional cognitive architectures by structuring processing into autonomous computational agents. This lets it support specialised representations and, in particular, specialised forms of learning, such as for perceptual recognition or low-level motor control.
- Unlike DIARC and T-REX, but like CAS, Mala restricts information to flow through blackboards and working memories. This facilitates the inclusion of architecture-wide introspective learning mechanisms, because monitoring the blackboard provides access to the system-wide flow of information.
- In contrast to other node-based architecture, including CAS, DIARC, and T-REX, which use ad-hoc methods for generating robotic behaviour from task-level action models, the combination of constraint solving and motion planning for action generation in Mala provides a method of generating behaviour online, that is general enough to let a robot execute new and possibly complex task-level actions as they are acquired.

Thus, Mala's design enables it to support a richer collection of learning methods than other integrated robotic architectures, through its node-based design, loosely coupled communication structure, and novel action generation pipeline.

## 5. Plans for Future Work

Although the Mala architecture has demonstrated successful operation in a challenging domain, our eventual goal is to operate an autonomous robot in more realistic and complex environments over a long time. To achieve this, we must extend the architecture in two ways: by embedding multiple types of learning algorithms and by augmented the system's capabilities through the addition of more agents.

### 5.1 Learning Mechanisms

We have argued that, for robust long-term autonomy, learning should be embedded within all components of the architecture. The features of the Mala framework enable it to accommodate several types of learning, to which we now turn.

### *5.1.1 Learning within Agents*

The world of an autonomous robot is not only dynamic in the sense that objects may move, but there may be fundamental changes in the environment and the robot itself. For example, the lighting conditions may shift, rendering a robot unable to distinguish objects using colour; a motor may stop working or the robot may enter a region containing unfamiliar hazards. To continue operating successfully, the robot must adapt its behaviours through learning. However, the best learning paradigm and algorithm for a given task depends strongly on the nature of the task (Sammut & Yik, 2010), and the Mala architecture contains many different components and representations. This suggests that we should incorporate different types of learning mechanisms to support this diversity.

For example, the current implementation uses a motion planner to generate trajectories for robotic execution, but these trajectories only specify the final pose; they do not extend to actions that specify the velocity of an actuator, such as using a hammer to hit a nail. In such cases, it may be necessary to replace the motion planner with a primitive behaviour learner that can acquire the ability to generate the required behaviour. The learning method suitable for such a learner may be reinforcement learning or numerical optimisation (Ryan, 2002; Sammut & Yik, 2010), which may be incorporated into Mala's drive controller agent.

### *5.1.2 Task-level Action Model Learning*

Learning within Mala is not restricted to the scope of a single agent. It is also possible for the collection of agents in the architecture to acquire task-level knowledge through experimentation in the physical world. This could be achieved by observing the effects of actions when percepts – representing the resultant world state after execution – are written to the blackboard. Training examples could be accumulated by pairing each action with either the frames representing the state in which they were executed, to learn action preconditions, or the observed effects of their execution, to learn action effects. These examples form a data set that a generalisation agent could use to build and refine action models that expand the system's range of abilities and the space of plans it can generate. One way to approach this idea is to generalise the learning system devised by Brown (2009, 2011). This requires a critic, problem generator, and problem solver, like those in Mitchell et al.'s (1983) LEX system, to be wrapped within agents that are connected to a blackboard. These would guide the learning process by placing goals on the blackboard and labelling training examples by observing the results of experiments. This would let our earlier work on autonomous learning of robotic skills in simplified environments be coupled to specialised robotic perceptual and actuation modules.

## 5.2 Extended Capabilities

We also plan to introduce several additional agents to implement new capabilities. First, different types of manipulation are useful for an autonomous rescue robot, such as the ability to place shoring material to reinforce damaged walls or to transport water and survival gear to victims. These can be implemented within our framework by adding grasp planner and grasp controller agents that execute task-level manipulation actions in the same way as the motion planner and drive controller described earlier. A further extension of the architecture would be to add active top-down perception in which

tasks such as inspecting an area closely or visually checking the status of a victim are guided by models of expected objects in the scene. A planner would reason about actions that can provide additional information to assist perception. Top-down perception is frequently observed in human cognition, but it has fallen out of favour in computer vision due to the success of global feature-based methods. However, model-based reasoning is beneficial in cluttered environments, such as disaster sites, where occlusion is very common.

## 6. Conclusion

A central goal of cognitive systems research is to study systems that are the products of multiple interacting components. This stems from the recognition that the phenomenon of intelligence is not restricted to any one of its component abilities, but results from their interactions and the framework in which they are embedded (Langley, 2012). An autonomous mobile robot is a classic example of an agent composed of many interacting parts, and thus provides an excellent analogue for the study of integrative cognitive systems. However, despite many impressive successes and considerable progress, current robotic systems are brittle and overly task specific. This has historically been caused by the difficulty of perception and actuation. However, it has also resulted from a lack of symbolic reasoning capabilities in robotic architectures, which enable more flexible retasking and greater possibilities for introspective monitoring and failure recovery. Although cognitive architectures can provide the integrated frameworks and reasoning capabilities to address the latter need, they are often difficult to implement on real robots, not least because they do not model primitive action or perception.

Thus, to achieve greater levels of autonomy, architectures for robots must merge these symbolic reasoning and planning capabilities with specialised perception and actuation so they can obtain information about the world without convenient labelling of objects, manipulate nontrivial objects, and drive over difficult terrain. In this paper we outlined the characteristics that an architecture must possess to address the above issues. We claim that these requirements cannot be achieved without specialised representations, modularity, and loosely coupled, concurrent processing elements. We presented the Mala architecture, that enables the fusing of symbolic processing at the level of abstract tasks using a classical planner with the sensory-actuation of a robot, and we demonstrated its successful operation in a challenging robotics domain. Our current focus is the implementation of the learning mechanisms and evaluation of them during extended autonomous operation.

## Acknowledgements

We thank Matthew McGill for his invaluable assistance in much of the programming of the robot software system and of the blackboard system. This work was partially supported by the Australian Research Council Centre of Excellence for Autonomous Systems.

## References

Anderson, J. R. (2007). *How can the human mind occur in the physical universe?* New York: Oxford University Press.

Apt, K. R., & Wallace, M. (2006). *Constraint logic programming using ECLiPSe*. New York: Cambridge University Press.

Bonasso, P. R., James Firby, R., Gat, E., Kortenkamp, D., Miller, D. P., & Slack, M. G. (1997). Experiences with an architecture for intelligent, reactive agents. *Journal of Experimental & Theoretical Artificial Intelligence*, *9*, 237–256.

Bradski, G., & Kaehler, A. (2008). *Learning OpenCV: Computer vision with the OpenCV library*. Sebastopol, CA: O'Reilly.

Brenner, M., & Nebel, B. (2009). Continual planning and acting in dynamic multiagent environments. *Autonomous Agents and Multi-Agent Systems*, *19*, 297–331.

Brooks, R. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, *2*, 14–23.

Brown, S. (2009). *A relational approach to tool-use learning in robots*. Doctoral dissertation, School of Computer Science and Engineering, University of New South Wales, Sydney, Australia.

Brown, S., & Sammut, C. (2011). Tool use learning in robots. *Proceedings of the 2011 AAAI Fall Symposium Series on Advances in Cognitive Systems*. Arlington, VA: AAAI Press.

Farid, R., & Sammut, C. (2012). A relational approach to plane-based object categorization. *Proceedings of the 2012 Robotics Science and Systems Workshop on RGB-D Cameras*. Sydney, Australia.

Gat, E. (1997). On three-layer architectures. In R. Bonnassso, R. Murphy, & D. Kortenkamp (Eds.), *Artificial intelligence and mobile robots*. Menlo Park, CA: AAAI Press.

Haber, A. L., McGill, M., & Sammut, C. (2012). jmeSim: An open source, multi platform robotics simulator. *Proceedings of the 2012 Australasian Conference on Robotics and Automation*. Wellington, New Zealand.

Hawes, N., Brenner, M., & Sjöö, K. (2009). Planning as an architectural control mechanism. *Proceedings of the Fourth ACM/IEEE International Conference on Human Robot Interaction*. La Jolla, CA.

Hawes, N., & Wyatt, J. (2010). Engineering intelligent information-processing systems with CAST. *Advanced Engineering Informatics*, *24*, 27–39.

Hoffmann, J., & Nebel, B. (2001). The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, *14*, 253–302.

Kadous, M., & Sammut, C. (2004a). MICA: Pervasive middleware for learning, sharing and talking. *Proceedings of Perware'04: Middleware Support for Pervasive Computing Workshop at the Second Conference on Pervasive Computing* (pp. 176–180). Orlando, FL.

Kadous, M. W., & Sammut, C. (2004b). Inca: A mobile conversational agent. *Proceedings of the Eighth Pacific Rim International Conference on Artificial Intelligence* (pp. 644–653). Auckland, New Zealand: Springer.

Laird, J. E. (2012). *The Soar cognitive architecture*. Cambridge, MA: The MIT Press.

Langley, P. (2012). The cognitive systems paradigm. *Advances in Cognitive Systems*, *1*, 3–13.

Langley, P., & Choi, D. (2006). A unified cognitive architecture for physical agents. *Proceedings of the Twenty-First AAAI Conference on Artificial Intelligence*. Boston, MA: AAAI Press.

McGann, C., Py, F., Rajan, K., Thomas, H., Henthorn, R., & McEwen (2008). A deliberative architecture for AUV control. *Proceedings of the 2008 IEEE International Conference on Robotics and Automation*. Pasadena, CA.

McGill, M., Sammut, C., & Westendorp, J. (2008). *Framescript: A multi-modal scripting language* (Technical Report). School of Computer Science and Engineering, University of New South Wales, Sydney, Australia.

Meyer-Delius, D., Pfaff, P., & Tipaldi, G. D. (Eds.). (2012). *Proceedings of the RSS Workshop on Long-term Operation of Autonomous Robotic Systems in Changing Environments*. Sydney, Australia.

Milstein, A., McGill, M., Wiley, T., Salleh, R., & Sammut, C. (2011). A method for fast encoder-free mapping in unstructured environments. *Journal of Field Robotics*, *28*, 817–831.

Minsky, M. (1988). *The society of mind*. New York: Simon & Schuster.

Mitchell, T., Banerji, R., & Utgoff, P. E. (1983). Learning by experimentation: Acquiring and refining problem-solving heuristics. In R. S. Michalski, J. G. Carbonell, & T. Mitchell (Eds.), *Machine learning: An artificial intelligence approach*. New York: Springer Verlag.

Nilsson, N. (1984). *Shakey the robot* (Technical Report 323). AI Center, SRI International, Menlo Park, CA.

Nilsson, N. (2001). Teleo-reactive programs and the triple-tower architecture. *Electronic Transactions on Artificial Intelligence*, *5*, 99–110.

Quigley, M., Conley, K., & Gerkey, B. (2009). ROS: An open-source robot operating system. *Proceedings of the 2009 ICRA Workshop on Open Source Software*.

Roberts, R. B., & Goldstein, I. P. (1977). *The FRL primer* (Technical Report AIM 408). Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA.

Ryan, M. R. K. (2002). Using abstract models of behaviours to automatically generate reinforcement learning hierarchies. *Proceedings of the Nineteenth International Conference on Machine Learning* (pp. 522–529). Sydney, Australia.

Sammut, C. (2012). When do robots have to think? *Advances in Cognitive Systems*, *1*, 73–81.

Sammut, C., & Yik, T. F. (2010). Multistrategy learning for robot behaviours. In J. Koronacki, Z. W. Ras, & T. Slawomir (Eds.), *Advances in machine learning I*. Heidelberg, Germany: Springer.

Scheutz, M. (2006). ADE: Steps toward a distributed development and runtime environment for complex robotic agent architectures. *Applied Artificial Intelligence*, *20*, 275–304.

Scheutz, M., Schermerhorn, P., & Kramer, J. (2007). First steps toward natural human-like HRI. *Autonomous Robots*, *22*, 411–423.

Sheh, R., Milstein, A., McGill, M., Salleh, R., & Hengst, B. (2009). Semi-autonomous robots for robocup rescue. *Proceedings of the 2009 Australasian Conference on Robotics and Automation*. Sydney, Australia.

Thrun, S. (2002). Probabilistic robotics. *Communications of the ACM*, *45*, 52–57.

Vainer, K. (2013). jMonkeyEngine 3.0 — Java OpenGL game engine. http://jmonkeyengine.com/.