Towards an Integrated Robot with Multiple Cognitive Functions

Nick Hawes[*] and **Aaron Sloman** and **Jeremy Wyatt** and **Michael Zillich**
School of Computer Science, University of Birmingham, UK

Henrik Jacobsson and **Geert-Jan M. Kruijff**
Language Technology Lab, DFKI GmbH, Germany

Michael Brenner
Albert-Ludwigs-University, Germany

Gregor Berginc and **Danijel Skočaj**
University of Ljubljana, Slovenia

Abstract

We present integration mechanisms for combining heterogeneous components in a situated information processing system, illustrated by a cognitive robot able to collaborate with a human and display some understanding of its surroundings. These mechanisms include an *architectural schema* that encourages parallel and incremental information processing, and a method for *binding information* from distinct representations that when faced with rapid change in the world can maintain a coherent, though distributed, view of it. Provisional results are demonstrated in a robot combining vision, manipulation, language, planning and reasoning capabilities interacting with a human and manipulable objects.

Introduction

We aim to understand how to build 'cognitive' robots, partly in order to eventually test theories about how humans work. Such robots should understand their environment and collaborate with humans in varied situations, employing many competences that are recombined as needed. Resulting behaviour must be extensible, purposive and flexible, i.e. achieving goals despite changing constraints. It must be robust in the face of imperfect sensors and effectors, able to modulate behaviour dynamically, and to learn without reprogramming, among other things. To meet this formidable combination of challenges we must answer many difficult questions about how systems can be synthesised from interacting, changeable, components. This paper reports on progress towards meeting such requirements, within a range of architectures for which we have implemented tools.

From analysis of detailed *scenarios* we derive *requirements* that lead to *design principles* for architectures that can be expressed in terms of *architectural schemata*. Schemata define a (large) *design space* containing many specific designs: the *architectural instantiations*. A schema provides a set of constraints on this space. The next section describes a scenario, and the requirements and design principles that arise from it, which are then expressed in a

---
[*]Corresponding author: n.a.hawes@cs.bham.ac.uk

schema, the CoSy Architecture Schema (CAS), and an associated software toolkit (CAST, the CAS Toolkit) that allows rapid prototyping of CAS instantiations. Some of the issues that arise are described in terms of a pair of related systems that perform tasks requiring human-robot collaboration: cross-modal learning of object qualities, and linguistically initiated manipulation. The systems integrate: vision, language understanding and generation, spatial reasoning, learning, planning and manipulation – using tens of individual task-specific modules running concurrently in real time. Problems addressed concern incremental processing, flow of information, linking of representations across components, cross-modal processing and processing control.

Background and Approach

We start from film script-like *scenario* descriptions of a collaborative table-top manipulation scenario in which the robot asks and answers questions about the scene, performs requested actions, and learns names and qualities of objects. Behavioural competences include recognising, locating and manipulating objects; planning action sequences; interpreting and generating natural language; and understanding various (multi-modal) properties such as size, spatial relations and colour. These behaviours must be performed in real-time in a contextually-appropriate manner. The robot should behave sensibly when interrupted, when its actions fail, when it lacks sufficient information and when it must coordinate different actions (e.g. speaking and moving).

Requirements arising from this task apply to many similar robot scenarios, including the ability to *cope with external change* (objects move, utterances occur during action); and the need to *integrate information from multiple sources* (speech, vision, touch) where related information from sources arrives asynchronously. For this, the architecture must support many components running concurrently. Besides these *run time* requirements there are also *design time* requirements. Architectures should make cognitive systems (relatively) easy to design, using current technologies. The best available components will often use specialised representations to facilitate and speed up processing. This makes integration harder as they may be incompatible with other representations. So the architecture needs to allow
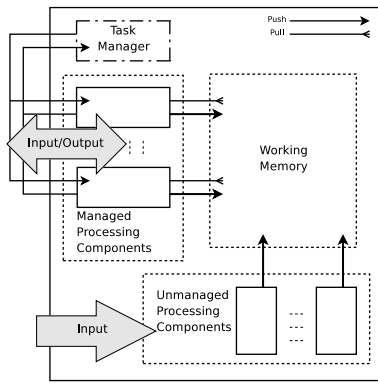
Figure 1: The CAS Subarchitecture Design Schema.

easy methods for linking modules using different forms of representation, without excessive run-time overhead.

The Architecture Schema

Analysis of scenarios led to three design requirements: support for concurrent modular processing, structured management of knowledge, and dynamic control of processing. These are met by the CoSy Architecture Schema (Hawes, Wyatt, & Sloman 2006). The schema allows a collection of loosely coupled *subarchitectures* (SAs). As shown in Figure 1, each contains a number of processing components which share information via a working memory (WM), and a control component called a task manager. Some processing components within an SA are *unmanaged* and some *managed*. Unmanaged components perform relatively simple processing on data, and thus run constantly, pushing their results onto the working memory. Managed processes, by contrast, monitor the changing working memory contents, and suggest possible processing tasks using the data in the working memory. As these tasks are typically expensive, and computational power is limited, tasks are selected on the basis of current needs of the whole system. The task manager is essentially a set of rules for making such allocations. Each SA working memory is *readable* by any component in any other SA, but is *writable* only by processes within its own SA, and by a limited number of other *privileged* SAs. Components within privileged SAs can post instructions to any other SA, allowing top-down goal creation.

If there are several goals in an SA they are mediated by its task manager. This mixes top-down and data driven processing and allows goals to be handled that require coordination within one SA or across multiple SAs. At one extreme the number of privileged SAs can be limited to one (centralised coordination), and at the other all SAs can be privileged (completely decentralised coordination). In our scenario the preference seems to be for a small number of specialised, privileged coordination SAs.

An overriding principle in CAS is that processing components work concurrently to build up shared representations. SAs work concurrently on different sub-tasks, and components of an SA work on parts of a sub-task. Instantiations of CAS function as distributed blackboard systems as used, for example, in Hearsay-II (Erman *et al.* 1988).

We have designed and built instantiations of CAS that have a small number of coordinating SAs, and a large number of components that concurrently refine common representations. Control of individual motor systems is not distributed in our implemented systems although this is perfectly possible using our approach. To support our work, the architecture schema is realised in a software toolkit (CAST) (Hawes, Zillich, & Wyatt 2007). CAST allows components written in different languages to be combined into CAS instantiations. Different instantiations can be created without recompilation or changes to components, allowing functionally different systems to be generated quickly, making it possible to evaluate the influence of the architecture schema on various instantiations without altering the components of the architecture.

Cross-Subarchitecture Binding

In order to combine information from different subarchitectures we need to provide a means for accessing information from their working memories. When subarchitectures are interpreting data from different sensors across an integrated system, they will generate various representations that reflect the surrounding environment in different ways. For example, spatial, visual, planning and communication SAs may each have their own representations of an object. Allowing each to influence others (e.g. in resolving uncertainty) and allowing information from different SAs to be combined (e.g. information about things that are seen and touched) requires the different representations to be bound together somehow. For example, if a visual SA notices a green object of unknown type that is within reach, then a "grab the green book" command processed by a language SA may be informative since a likely category for the object can be inferred ("book"). Alternatively, information from a visual SA's working memory might be used by the linguistic SA to determine the referent of the phrase "the green book".

We have been experimenting with a CAS instantiation that uses a *binding* SA which contains a working memory that mirrors the contents of all SAs that contain information about objects[1]. The binding SA features a component called a binder that maintains the contents of binding WM by joining information from different SAs. The binder receives candidates for binding from other SAs. These candidates must be filtered and abstracted with respect to their source domains to contain information that could be useful to other SAs (such as colour and type in the previous example). The filtering and abstraction is task-dependent and therefore each SA requires a corresponding binding *monitor*. This is a process which is triggered by updates to the WM and which translates these into binding *candidates* before writing them to the binding WM where they will be bound together by the binder to form *instance bindings* (representations that should roughly correspond to the objects in this case). Monitors should ideally perform *intra-modular bind-*

---

[1]Although we focus on objects in this discussion, binding could occur between other classes of things, including processes.

ing (Kruijff, Kelleher, & Hawes 2006) in order for the binder to be relieved of domain-specific reasoning.

To enable other SAs to communicate with the binding SA, both to generate binding candidates and to use the information represented by instance bindings, a shared language is needed. In our instantiations this language is oriented around *features* and *relations*. The features describe properties of objects that can be provided by SAs, e.g. colour, shape, position, etc. The relations describe states that can exist between two or more objects, e.g. spatial relations. Some features can be compared with (or be in common with) features of other modalities in which case a common ground for evaluating the binding is established. New features and comparison functions between features can be added if new SAs are added that support, or require, new features. Binding candidates and instance bindings are represented in the binding WM as sets of these features. The feature set can contain zero, one or more instantiations of values for each feature. This means that the object description conveyed by the binding candidates and instance bindings are open-ended in terms of how much they specify about their referent: either the feature is not represented at all, or it is instantiated with one or more values (e.g. an object may have more than one colour). To maintain instance bindings, the binder compares the features of newly generated candidates to those of the existing bindings (using a scoring function). If a clear match emerges from this process then the new candidate is bound into the matching instance binding. If no matching instance binding is found, a new one is created. If more than one instance binding matches equally well with a new candidate, this creates a disambiguation problem. Investigating this is part of our ongoing work.

Once an instance binding has been created, any component in the architecture can use this to access all of the candidates bound into it. Moreover, these binding candidates may in turn refer back to more detailed representations in their source SAs. The binding SA WM thereby serves as a provider for *proxies* between all SAs in the system. This proxy service, based on a shared feature language, greatly simplifies the problem of exchanging information between SAs that use heterogeneous representations.

Instantiations and Empirical Work

Our integration mechanisms have been tested by incrementally implementing two systems using CAST: a cross modal learning instantiation (CLI), and a linguistically-driven manipulator instantiation (LMI). The CLI (Figure 2, dark lines) combines SAs for cross-modal language interpretation and generation (Kruijff, Kelleher, & Hawes 2006) and visual property learning (Skočaj *et al.* 2007) to produce a system that can learn and describe object properties in dialogue with a tutor. The LMI (Figure 2, light and dark lines) extends the CLI with SAs for planning, spatial reasoning and manipulation (Brenner *et al.* 2007) to produce a system that executes manipulation commands that refer to objects using the previously learnt visual properties. In these instantiations information exchange between the subarchitectures is mediated by the binding mechanism discussed previously.
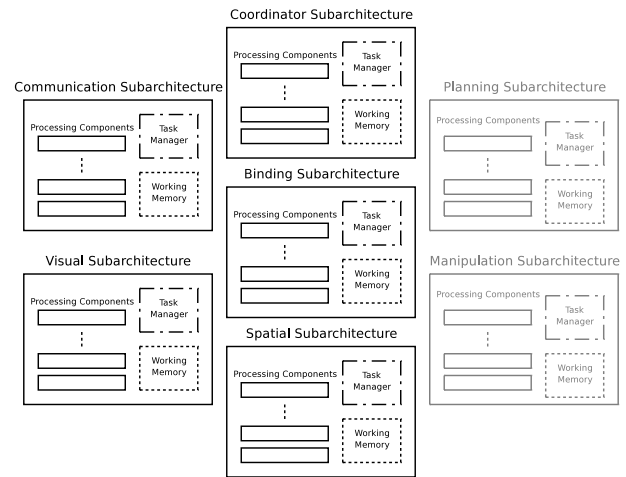

Figure 2: The instantiation designs.

A	Red object placed on table.
B	Tutor (T): "This is a red thing."
C	Red object replaced with blue object.
D	Robot (R): "Is that red?"
E	T: "No, this is a blue thing."
F	Blue object replaced with red object.
G	Blue object placed to right of red object.
H	Blue object placed to left of red object.
I	T: "Put the blue things to the left of the red thing."
J	R moves right hand blue object to left of red object.

Figure 3: Events from the instantiation run

In the CLI the subarchitectures are as follows: the communication SA (CSA) containing components for speech recognition, dialogue interpretation and production, and speech synthesis; the vision SA (VSA) containing components for change detection, segmentation, and visual property learning (three components); the binding SA (BSA) containing components for generating visual and linguistic binding candidates, and the binder; the spatial SA (SSA) containing components for monitoring instance bindings and representing the current scene; and the control subarchitecture containing components for motive generation and management. In the LMI the SSA is extended with components for adding spatial relationships to the current scene, and with these additional subarchitectures: the planning subarchitecture containing components for planning, problem generation, and execution monitoring; and the manipulation subarchitecture containing a component for translating planned actions into behaviour via visual servoing.

The architecture schema and binding mechanisms can be illustrated with an extended example from the combined CLI and LMI system. In this example the tutor teaches the colours of objects one by one, before giving a manipulation command (see Figure 3). The implemented system is capable of discriminating many more colours and other visual properties, and giving verbal descriptions of the scene,
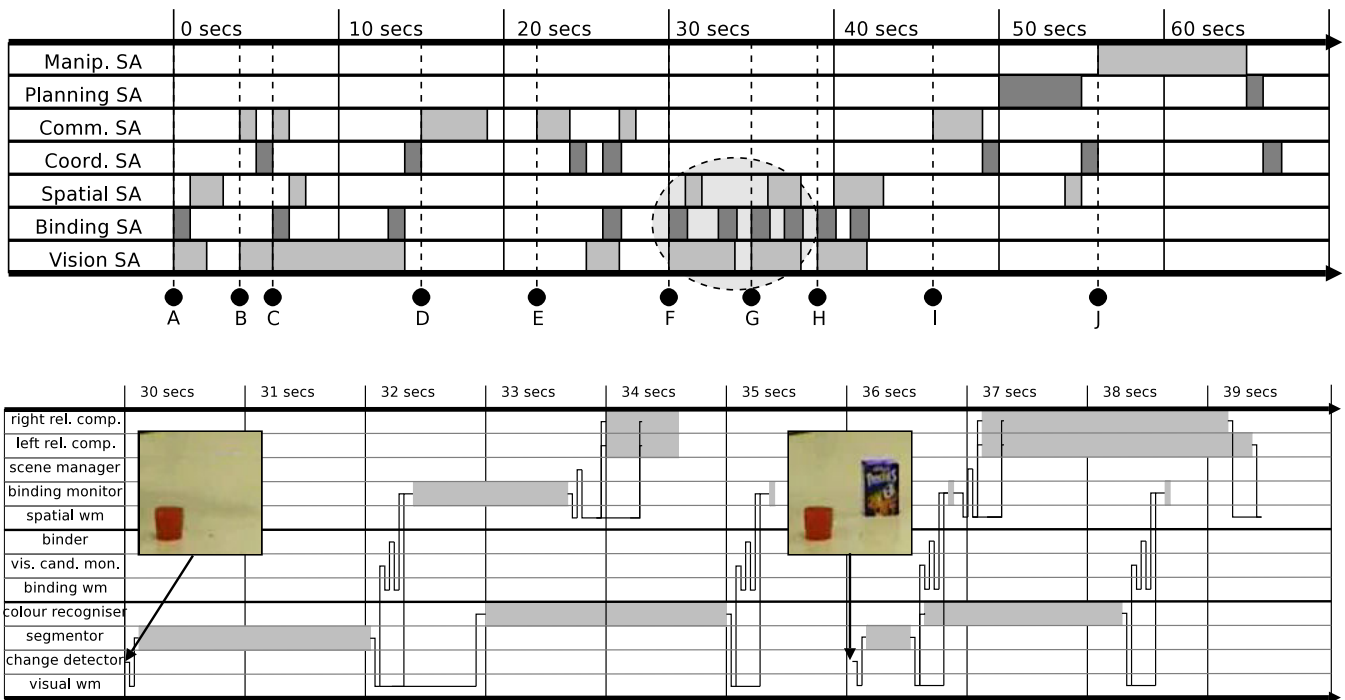
Figure 4: How processing occurs across SAs in the example. The lower diagram contains a detailed view of the circled area from the upper one. Grey areas represent processing. Black lines on the lower diagram represent data exchanged via WMs.

but because of space we consider only this example. The account is organised by the integration mechanisms employed. These mechanisms are *the collaborative refinement of common representations*, *cross-modal binding* based on known mappings between ontologies, *top down and bottom up goal raising*, *learning mappings between ontologies*, *lazy binding* and *mediation between qualitative and quantitative representations*. None of these is enforced by the schema, but they arise naturally from the decision to employ a variety of representations in separate subarchitectures. All are general, none rely on properties of the example. To aid the reader we annotate the text with markers for the events A to J described in Figure 3. How subarchitecture activity varies with these events can be seen in Figure 4. Although the instantiation is usually distributed across a number of machines, this data was generated by running it on a single machine. This decision was taken to ease the process of collecting data. With a distributed setup processes complete quicker, but the pattern of information processing remains the same.

Collaborative Refinement of Representations

A subarchitecture couples the results of processing via the types of objects can that can reside in the working memory. For example, in the visual WM (VWM) there are two main types of data object, corresponding to regions-of-interest (ROIs) and proto-objects (POs). Each is structured and typed. Visual components refine knowledge about either by overwriting their fields. Components can refine different fields of the same data-object in parallel. If the data-objects are well chosen they provide a degree of bind-

ing of the results of processing. When, for example, an object is placed in front of the robot (events A, C, F, G, H) the visual change detection component is triggered, causing a scene changed fact to be added to the VWM. The appearance of this fact causes the segmentor component to run on the changed scene, which results in a new ROI being generated along with a related PO. Once these data structures are in working memory other components start to process them in parallel (as seen in Figure 4). In the visual SA the property learning components extract features from the ROI and these are added into the ROI data structure. Their presence in turn triggers the recogniser, which adds any recognised property data — such as the colour and shape. When objects are added in quick succession (events F,G,H) each triggers a similar sequence. New objects are therefore processed concurrently with old ones. The only current restriction is that each component processes one object at a time.

Cross Modal Binding

Previously we described the cross-subarchitecture binding mechanism we use in the instantiations. This mechanism does not bind directly between pairs of subarchitectures, but via a shared feature-based language. The important point here is that it relies on mappings between the ontologies for different subarchitectures, e.g. between the binding features used for the visual entity red and the name red. This shared language has a mapping to each different SA representation. Typically, this mapping would be defined by the system designer, an approach taken in our system for most of the mappings. We would like, however, for the system to be able to

acquire such mappings, and this is what the learning part of the example interaction demonstrates. Later on, when references are made to objects (e.g. in event I), the binder works using the mappings learnt previously between the ontologies (in events A to E) to carry out the binding.

Learning Mappings Between Ontologies

In the first learning case (events A and B) the tutor puts down a red object. As no colours have been learnt yet no visual property information is added to the ROI. When the tutor says "this is a red thing" the speech recognition component adds the speech string into the CSA where it is processed by the parser and the dialogue interpretation component. The result is a structure in the communication WM describing the teaching instruction. This triggers the coordination SA to generate and act on a motive to learn the visual properties of the thing referred to by "this". Since there is no mapping between ontologies to guide binding, the robot assumes that the object the human is teaching it about is the most recent one to appear. The motive manager in the coordination SA uses this binding to get a reference to the associated PO and ROI. It now pushes a goal down to the VSA by posting a visual learning instruction together with information about the properties of the object. This is an example of top down goal creation. A description of an object for learning may contain several different types of properties ("a small round red thing"). When the learning goal is written to the VWM the visual property learning component is triggered and updates its internal representations. The learning algorithm we use is able to learn quite general correlations between elements in two ontologies (Skočaj *et al.* 2007). The colour information is translated from the linguistic representation (as a structured logical form) into the visual representation (a colour index) using the binding feature language.

When the red object is used for learning (event C) the visual system now guesses that the object is blue, due to generalisation from the previous case. It doesn't regard the match as reliable, so a query is raised by writing a structure containing references to the PO and the colour blue into the VWM. In a reverse of the previous interaction this causes a *clarification dialogue* in which the robot says "is this blue?", to which the tutor replies "no, this is red". This demonstrates how the mix of top down and bottom up control gives the robot the ability to take initiative in the dialogue.

Lazy Binding and Mediation

The last stages of the example concern the planning and execution of a manipulation. This demonstrates our approach to binding through planning, and the need to mediate between qualitative and quantitative representations of space. After the first blue object is put down (event G), a second blue object is placed to the left of the red object (event H). Following this the human commands the robot to "put the blue things to the left of the red thing" (event I). The CSA interprets the utterance as a command and, consequently, the coordination SA sends it to the planning SA. The command is translated to a planning *goal formula* (Brenner *et al.* 2007). One particularly important aspect of this approach is that referential constraints used in the command can be

kept unbound in the goal formula and will be resolved by the planner in a context-dependent manner. This is particularly important for resolving referential expressions involving plurals and binding constraints, e.g. "the blue things near the green thing", that may not be bound completely before starting the execution of a plan. If, for example, during plan execution a new blue object is perceived, the monitoring component of the planning SA will trigger re-planning in which the planner will resolve the same goal formula differently, this time moving the newly detected object too. We refer to this process as *lazy binding*.

To convert the world state into an *initial state* for planning the problem generation component in the planning SA pulls a spatial description of the current scene from the spatial SA WM. Similar information about visual features is pulled from the visual SA (via the binding SA) and included in the initial state. These facts about objects and their relations are then used by the planner to generate a plan that satisfies the goal formula, resolving the referential constraints on-the-fly. In our example, the planner correctly detects which objects are to be moved and which target positions satisfy the description "to the left of the red thing".

This interaction demonstrates how the binder mediates the exchange of information between components. In this case, it provides the planning SA with the necessary qualitative representation of objects and their relations, but allows it to ignore the underlying quantitative representations used by the SAs the information comes from. This is also illustrated by the information flow when a plan is executed. In the implemented system when the plan is successfully created the coordination SA passes it on to the manipulation subarchitecture. Each object in the plan is represented as a reference to an instance binding, which can be used to access the binding candidate from the visual SA which in turn can be used to access detailed information about the object's pose in the world. The robot must then pick a particular location in the world that satisfies the qualitative goal position "left of". This is achieved by inverting the mapping from a quantitative to qualitative spatial representation. We refer to this as the process of *mediating between qualitative and quantitative representations* of space. It's this quantitative information that is used by the manipulation SA when plans are executed, but it's precisely this kind of detailed, volatile information that the planning system must be isolated from when planning about object positions.

Discussion and Conclusion

We can compare the presented schema to related work on architectures for modelling human cognition and to architectures for robotic systems. It is also beneficial to consider what features are commonly required by robotic integrated systems such as the ones we ultimately wish to produce.

One of the crucial differences between the CoSy Architecture Schema and architectures for modelling human cognition such as ACT-R (Anderson *et al.* 2004), Soar (Laird, Newell, & Rosenbloom 1987) and ICARUS (Langley & Choi 2006) is that these commit to unified representations for all information, whereas CAS permits the specialised representations typically required by robotic systems. This

is related to the fact that processing components in cognitive modelling architectures are typically logical rule systems, whereas CAS components can be arbitrary processing modules. Architectures for modelling human cognition also typically include mechanisms for learning (e.g. chunking in Soar and spreading activation in ACT-R). Such mechanisms are not present in CAS at the schema level, but could be designed into an instantiation.

A key feature of CAS is that it allows components to be active in parallel. An architecture that ran the components from the lower half of Figure 4 in serial would require approximately four seconds longer to process the data. This is simply a result of parallelism, but an architecture must support parallelism in terms of control and the concurrent access to information. Concurrently active components are common in existing robotic systems (e.g. (Mavridis & Roy 2006)), but support for them is missing from the cognitive modelling architectures mentioned previously. Parallelism is usually present at a low level in robotic architectures designed to operate reactively on their world (e.g. the Subsumption architecture (Brooks 1986)). On top of this many robotic tier-based architectures build scheduling apparatus to generate goal directed behaviour (Gat 1997). Such scheduling systems are not explicitly required by our schema, but the task manager plays a similar role by providing control over processing components. Cognitive modelling architectures typically do not support parallel components, but rather feature serial rule execution (although Minsky's work is an exception (Minsky 1987)).

Although we are aware that our work overlaps and contrasts in various interesting ways with many other robotic and integrated system projects, some with primarily engineering goals, and some aiming towards modelling human and animal capabilities, space constraints have forced us to leave detailed comparisons to be addressed in future papers. Although our work can be superficially compared to that of Mavridis & Roy (2006), McGuire et. al (2002) and Bauckhage et. al (2001), the crucial difference is that whereas these systems have been created to perform in a limited task domain, our instantiations have been designed and built to support what we call "scaling out" (as opposed to "scaling up"): new components have access to data produced by existing components via working memories, the binding mechanism is able to accommodate new information and representations into its feature set, and additional functionality provided by new components allows new tasks and problems to be dealt with, using the same architectural framework.

In summary we have presented an architectural schema and a number of novel integration mechanisms, and shown how they can support flexible interactions between a human and robot in a tabletop domain with objects. We have shown how the integrated system is able to scale itself out (the ontology learning works for tens of object properties) and argued that it is the innovative integration mechanisms and the schema as a whole that have enabled this flexibility.

Acknowledgements

References

Anderson, J. R.; Bothell, D.; Byrne, M. D.; Douglass, S.; Lebiere, C.; and Qin, Y. 2004. An integrated theory of the mind. *Psychological Review* 111(4):1036–1060.

Bauckhage, C.; Fink, G. A.; Fritsch, J.; Kummert, F.; Lömker, F.; Sagerer, G.; and Wachsmuth, S. 2001. An Integrated System for Cooperative Man-Machine Interaction. In *IEEE Int. Symp. on Comp. Int. in Robotics and Automation*, 328–333.

Brenner, M.; Hawes, N.; Kelleher, J.; and Wyatt, J. 2007. Mediating between qualitative and quantitative representations for task-orientated human-robot interaction. In *Proc. IJCAI '07*.

Brooks, R. A. 1986. A robust layered control system for a mobile robot. *IEEE J. of Robot. and Automation* 2:14–23.

Erman, L.; Hayes-Roth, F.; Lesser, V.; and Reddy, D. 1988. The HEARSAY-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty. *Blackboard Systems* 31–86.

Gat, E. 1997. On three-layer architectures. In Kortenkamp, D.; Bonnasso, R. P.; and Murphy, R., eds., *Artificial Intelligence and Mobile Robots*.

Hawes, N.; Wyatt, J.; and Sloman, A. 2006. An architecture schema for embodied cognitive systems. Technical Report CSR-06-12, Uni. of Birmingham, School of Computer Science.

Hawes, N.; Zillich, M.; and Wyatt, J. 2007. BALT & CAST: Middleware for cognitive robotics. Technical Report CSR-07-1, Uni. of Birmingham, School of Computer Science.

Kruijff, G.-J.; Kelleher, J.; and Hawes, N. 2006. Information fusion for visual reference resolution in dynamic situated dialogue. In Andre, E.; Dybkjaer, L.; Minker, W.; Neumann, H.; and Weber, M., eds., *Proc. PIT '06*, 117 – 128.

Laird, J. E.; Newell, A.; and Rosenbloom, P. S. 1987. Soar: An architecture for general intelligence. *Artificial Intelligence* 33(3):1–64.

Langley, P., and Choi, D. 2006. A unified cognitive architecture for physical agents. In *Proc. AAAI '06*.

Mavridis, N., and Roy, D. 2006. Grounded situation models for robots: Where words and percepts meet. In *Proc. IROS '06*.

Mcguire, P.; Fritsch, J.; Steil, J. J.; Rothling, F.; Fink, G. A.; Wachsmuth, S.; Sagerer, G.; and Ritter, H. 2002. Multi-modal human-machine communication for instructing robot grasping tasks. In *Proc. IROS '02*.

Minsky, M. L. 1987. *The Society of Mind*. London: William Heinemann Ltd.

Skočaj, D.; Berginc, G.; Ridge, B.; Štimec, A.; Jogan, M.; Vanek, O.; Leonardis, A.; Hutter, M.; and Hawes, N. 2007. A system for continuous learning of visual concepts. In *Proc. ICVS '07*.