An Architecture Schema for Embodied Cognitive Systems

Nick Hawes, Jeremy Wyatt and Aaron Sloman

November 24, 2006

DRAFT: Please do not quote without permission; comments welcome.

Contents

1 Introduction

The study of architectures to support intelligent behaviour is certainly the broadest, and arguably one of the most ill-defined enterprises in AI and Cognitive Science. In the CoSy project one of our goals is to develop and understand cognitive architectures suitable for the control of robots. This is not the same as developing an architecture for robot control, nor is it the same as developing a purely cognitive architecture unconnected to real sensors or actuators. We argue that work on architectures traditionally falls into two camps. First there are cognitive architectures which attempt to provide unified theories of cognition such as SOAR [Laird et al., 1987] and ACT-R [Anderson et al., 2004]. Their value is typically measured in terms of an ability to reproduce some of the characteristics of human like information processing. ACT-R for example is used extensively for creating models of cognition that are then evaluated against data from humans. In other words, they are evaluated as psychological theories. On the other hand, roboticists have been engaged with issues of how to enable robots to act reliably and robustly in a rapidly changing world when faced with limited computational power, uncertain sensing, and uncertain action. Architectures for robot control such as 3T [Bonasso et al., 1997] are therefore largely concerned issues of real-time control, uncertainty, sensory fusion (or the lack of it), and error recovery. They are evaluated in terms of the performance of the resulting robotic systems on a variety of tasks.

In CoSy we have interests in both cognitive science and engineering science, and consequently our work is related to both of the aforementioned camps whilst also looking at closely related issues. Our work is neither concerned with trying to model humans or any other specific type of animal, nor with trying to compete on practical design tasks. Rather it is concerned with trying to understand the possibilities and trade-offs involved in different designs in relation to different sets of requirements. In this paper we will describe an architecture schema which inherits some of the ambitions of classic cognitive architectures, and those of robot control architectures, whilst allowing us to explore these additional issues. It is important to note now that we don't present an empirical evaluation of an implementation of a *scenario-specific instantiation* of the architecture schema in this paper – although we do describe two possible instantiations based on the CoSy demonstrator scenarios. It is also worth stating at this stage that our intention is not to perform extensive evaluation of the architecture schema against human behaviour. Instead we intend to evaluate it by profiling the behaviour of scenario-specific instantiations of it under varying conditions, such as internal failures and varying types of change in the world.

The document is organised roughly as follows. In the next section (Section 2) we describe some of the background to our research on architectures and describe our methodology. In Section 3 we give a high-level description of the scenario that motivates our work, and the requirements that we have drawn from this scenario. In Sections 4 & 5 we then outline the key features of our architecture schema, and the key design principles. These include concurrent, modular processing; structured management of knowledge; and dynamic context sensitive processing. The architecture is composed of a set of subarchitectures coordinated by a central reasoning system. We describe the subarchitectures in Section 6, and the central coordination mechanisms in Section 7. In Section 8 we describe the various roles subarchitectures can play in an instantiation of the architecture schema. In Section 9 we give an example of processing

with the architecture design, and in Section 10 we compare the architectural choices we have made, and the issues we are addressing, with those of other prominent architectures from both the cognitive science and robot control communities. Finally we describe two possible scenario-specific instantiations of the architecture schema, and the flow of information in a particular instantiation of a subarchitecture.

2 Architectures and the Science of Cognitive Systems

To provide some motivation and context for this document we start by clarifying the role of "architecture" in the science of understanding and building artificial cognitive systems. To understand its role we must first go back to basics. AI is a field of science that attempts to *understand* intelligent systems partly through the process of *building* them. To date a great deal of work has been done on the building part of this, and much less on the understanding aspect.

If we are to attempt to understand the artificially intelligent systems built by ourselves and others, we must be able to objectively compare them to other such systems. To look at this in a different way, we require something that is orthogonal to the RoboCup competitions [Kitano et al., 1997, Kitano et al., 1999]: where they ask "how good is a system at doing X", we want to ask "*why* is a system good at doing X" and "why is system X better than system Y when they both make use of Z". It is hard to ask such questions of designs for systems when they bear no relation to each other, it is harder still to do so for implementations of the designs when the implementations can have an unclear relationship to the original designs. It is because of this issue that our research group has previously argued for researchers to make the distinction between design space and niche space [Sloman, 1998, Hawes et al., 2006], and proposed the CogAff schema as (an incomplete first draft of) an ontology for comparing architectures [Sloman, 2001].

Currently we are interested in applying this methodology to meet the main goal of the CoSy project, namely "to advance the science of cognitive systems through a multi-disciplinary investigation of requirements, design options and trade-offs for human-like, autonomous, integrated, physical (e.g., robot) systems"[1]. Given our previous statement about the nature of AI, this means that not only must we study and design such systems, but we must also implement them in order to further our understanding of our designs.

We are engaged in trying to understand principles for designing robotic systems and actually building one or more in order to test ideas and demonstrate the theoretical claims. In this process the study of architectures plays a dual role: On the one hand studying and comparing requirements and architectures that have previously been proposed provides insights leading to a general schema characterising possible structures for integrated intelligent systems. On the other hand, a specific architecture, once chosen for implementation, provides practical guidance and support during the implementation process, especially if tools are available that are designed to support the requirements of such architectures.

This is not a novel idea. Some of the most prominent architectures in AI & Cognitive Science provide software toolkits which can be used to implement systems based

---

[1]Taken from the CoSy website at cognitivesystems.org.

on them [Laird et al., 1987, Jones et al., 2006]. Unfortunately the use of such toolkits is problematic when the scientific aims of a project involve studying a variety of different architectures that include many software components from various sources that are written in different languages. Instead we must produce concrete designs based on our scientific principles, then constrain our implementations to meet the restrictions of these designs.

To satisfy the scientific principles mentioned above, we must do the following when faced with the task of implementing a robotic system:

1. Study the *requirements* of each of the designs that we eventually want to be able to understand and compare by building examples of them. This study can be done partly through the analysis of *scenarios* in which the systems demonstrate their competences [Sloman, 2006].

2. Use these requirements to produce an *architecture schema*. An architecture schema is a task and implementation independent set of rules for structuring processing components and information, and controlling information flow.

3. Using a scenario that includes considerable generality in the form of multiple tasks and multiple behaviour sequences, produce one or more instantiations of the schema so that implementations can be tested and demonstrated, and their strengths and weaknesses compared.

The separation of steps 2 and 3 is crucial because when evaluations are carried out across multiple implementations of instantiations of the abstract schema, or across multiple schemata with similar features, it must be clear as to which elements are part of the abstract schema (and therefore comparable across instantiations) and which are not (and therefore not directly comparable).

This document addresses the second and third of these above points. It presents an architecture schema for a certain (restricted) class of cognitive systems based on requirements we have identified previously. It also presents two scenario-specific instantiations of this schema in the form of designs for architectures suited to the CoSy test scenarios (which allow for a variety of types of tasks and behaviours). Ideally alternative architectures should be implemented and compared, but that is beyond our resources. Instead of that we can make comparisons with related implementations produced by other researchers.

The schema presented in this document should not be confused with the CogAff schema which we have presented elsewhere [Sloman, 2001][2]. The CogAff schema can be used to describe any architecture schema or instantiation, whilst the schema presented here is only able to describe a subset of designs.

3   Scenario & Requirements

In order to start the process described in Section 2 we must have a scenario or set of scenarios to work from. In a previous project deliverable (see CoSy deliverable DR.1.1)

---

[2]For a more in-depth discussion of the CogAff schema, see CoSy Deliverable DR-2-2.

we analysed a scenario in which a hypothetical future robotic assistant interacts with a family and its environment, changes its world through physical action and linguistic discourse, performs household tasks for its owners, and learns about its world. This analysis gave rise to a list of behaviours that such a robot must be able to enact. From this list of required behaviours it is possible abstract a fairly small set of general architectural requirements which we can use both to restrict our architecture schema, and to guide the design of specific instances of that schema. These requirements were selected because they appear to be essential to the performance of many of the tasks in the scenario, and also because they are central to some of the simpler scenarios we have derived from this more complex scenario (cf. the Explorer and Playmate scenarios). In short, any architecture schema we produce must support:

1. **Interaction in a Dynamic World**: Because the real world changes frequently and unpredictably, and because some behaviours are tied to the dynamics of the real world (e.g. grasping a moving object, or even comprehending a sentence as it is spoken), the architecture must allow processes within the robot to interact with each other and with processes in the external world in a manner appropriate for their purpose.

2. **Information Integration from Multiple Sources**: To be able to act and interact both linguistically and through various forms of physical behaviour (e.g. grasping, locomotion) many different forms of information must be considered. Allowing communication between different subsystems and consistent decision making and conflict resolution means that the architecture must provide mechanisms for information transfer, for relating the content of information from different sources, possibly using different formats, and for detecting and dealing with conflicts or inconsistencies both within and between sub-processes.

3. **Goal-Directed Behaviour**: In a system with multiple resources and multiple goals (as in our scenario) it is crucial that it can manage its long term behaviour in a way that allows it to act to achieve the goals. This means that the architecture must provide structures that allow global motives to influence processing throughout the system.

It is important to note that although these requirements describe generally desirable properties of intelligent systems, they are not intended to describe the complete space of requirements for a general intelligent system. Instead they are intended to reflect the most important requirements from our study of a set of limited scenarios. As such, the above observations, and all subsequent proposals relating to the architecture schema, are all related to this limited set of scenarios. We are not attempting to propose a general theory of mind, cognition (human-level or otherwise) or of information processing, but rather study the design options and trade-offs apparent in the space of possible information-processing architectures for robots that can display the competences required for our limited set of scenarios.

The scenarios we have chosen involve a robot with multiple capabilities: language understanding, multiple types of vision, planning, navigation and manipulation, all in interaction with a human. This means that a number of architectural issues not commonly addressed become prominent. Perhaps the most important is that because we want to incorporate multiple specialised representations for each sub-system (as opposed to the general representations employed in SOAR or ACT-R) we are driven to

consider many processing modules, each of which updates asynchronously. In contrast to some robotic architectures these multiple modules may not be vying for control of actuators, but may be generating new knowledge that flows around the system. This means that the problem of how the information flow should be managed looms large. In particular we have hard choices to make about the degree of knowledge sharing between modules; how they should be grouped, if at all, into sub-systems; whether a particular type of information should be pushed or pulled through the system; how varying latency in communication between modules can be accommodated; how incremental processing between some sub-systems can be implemented; whether coordination should be achieved centrally; and in what ways execution monitoring and error recovery need to be incorporated. Some of these issues arise in existing robotic architectures, but no robot systems that we know of, employing principled architectures, exist able to cope with such a variety of input channels or such a variety of types of internal processes collaborating in producing overall intelligent behaviour. The architecture schema we give doesn't make final decisions about all of these issues, as many are issues relevant to particular scenario-specific instantiations of the schema.

Although all of our scenarios feature language we do not see linguistic competence as essential to models of intelligence. It is clear both from the existence of highly competent pre-linguistic human children, and the existence of intelligent non-human animals that do not use human language (including hunting animals, nest building animals, tree climbing animals, animals with complex social systems, etc.) that use of a human language to communicate is not a requirement for rich and flexible behavioural competence.

It is therefore important to understand what sorts of mechanisms, forms of representations and architectures can support such rich competences without the use of human language. Moreover, it is arguable that unless human language is added to a system that already has goals, percepts, intentions, plans, for which a variety of competences can be combined, there will be no basis for the language to have a semantic content for the animal or robot. (It will have nothing to communicate.) So we need to be able to investigate what sort of architectural scheme can incorporate a robot that exhibits substantial competence without the use of language, and then extends its competence by the addition of language, where the language subsystems interface in principled ways to the other components.

4   Key Features

The following list provides a quick summary of the key features of the architecture schema. More detail is provided in the subsequent sections.

1. The schema contains *subarchitectures* for separate competences. These subarchitectures are loosely coupled to reduce complex inter-dependencies. Systems could contain subarchitectures for motor control, vision, action, planning, language etc. The structure is recursive. Some subarchitectures can be further decomposed, for instance in the multi-window visual and motor sub-systems mentioned in the representations deliverable.

2. Each subarchitecture contains a number of processing components which share information via a subarchitecture-specific working memory.

6

3. Each of these subarchitecture working memories is only writable by processes within its subarchitecture, and by a single global process (the *goal manager*).

4. Information flow is controlled by goals generated within the architecture at run time, allowing it to deal with new problems and opportunities. This allows the schema to support different approaches to processing (e.g. incremental processing, forward chaining, backward chaining etc.). The architecture schema distinguishes between two classes of goal: *global goals* (goals of various kinds that require coordination across subarchitectures), and *local goals* (goals of various kinds that originate from a processing component within a single subarchitecture).

5. The knowledge that can be used within a subarchitecture is defined by a set of ontologies for that subarchitecture. Relationships (not necessarily equivalence) between the entries in the ontologies in different subarchitectures are defined by a set of general ontologies. These ontologies can also be used to define knowledge at an architecture-general level.

5   Design Principles

From the explicit architectural requirements presented in Section 3 we have generated three related design principles which represent the realisation of these requirements in our architecture schema.

5.1   Concurrent Modular Processing

Two design features will help with the requirement of supporting action in a dynamic world in the scenarios we are considering.

First, our architecture design is composed of concurrently active processing components. The need for this, as many other researchers have observed, arises from the fact that action may require concurrent sub-tasks, including controlling detailed arm and finger movements at the same time as using visual information to guide the process. We also require the robot to be able to cope with a question or piece of advice from a human while performing the task. In principle a very fast single process using a 'big switch' could deal with multiple-competence tasks, but a more modular design based on concurrent processes aids development and testing and allows for the possibility of distribution across separate processors. It also allows the architecture go grow, with new components being added while existing ones are not completely redesigned to provide the new functionality.

Second, the architecture combines processing components into *subarchitectures* for various competences (e.g. language, vision, manipulation, action planning etc.) connected by a small number of competence-independent components. Subarchitectures give the architecture coarse internal structure, and importantly (for the requirement of supporting action in a dynamic world) allow processes that commonly work together to easily share information without interacting with other (subarchitecture external) processes. Additionally, this structure is useful for managing information exchange across

competences (cf. Section 5.2), controlling processing (cf. Section 5.3), and allowing the architecture (and its functionality) to be extended relatively independently of existing components.

To reduce the dependence of components in one subarchitecture on the presence of other components within the same or other subarchitectures (for flexibility and ease of alteration), our design encourages subarchitectures to interact via local working memories. Our current design only allows access to a subarchitecture's working memory by components within the subarchitecture and architecture-general processes. This places the burden of control on the global processes and prevents subarchitecture processes from having a view of the entire architecture. Instead, subarchitecture processes can access particular global memories (which could be subdivided into globally accessible memories for particular tasks) into which selected results of processing across the architecture are asynchronously added, cf. Sections 7.3 & 8.1.

5.2   Structured Management of Knowledge

The requirement that our architecture is able to integrate information from multiple external and internal sources represents a key problem in designing (and implementing) cognitive systems. Many designs focus on structuring processing components but ignore the information that flows between them. This makes the whole system more about different forms of processing, rather than a gradually evolving information processing context. To overcome this, we propose a rough ontology-based structure for the managing information across the architecture.

The information structure is defined by two levels of ontologies: *subarchitecture ontologies* and *general ontologies*[3]. Each subarchitecture has its own set of subarchitecture-specific ontologies which define the information it can process. In addition to this it is evident from previous work that the knowledge described by subarchitecture ontologies will require its own representations, so for the remainder of this discussion when we use the term "ontology" we are also implying an attendant formalism that is used to represent information described by the ontologies.

Subarchitecture ontologies will contain knowledge specific to the competences that the subarchitecture supports. It is assumed that a lot of this knowledge will only be understood by the processes within the subarchitecture (e.g. if it represents the intermediate results of a process), or that this knowledge may have an expected useful life-time that may make sharing it with other subarchitectures pointless (e.g. precise metrical information about the world which will change unpredictably). The general ontologies are not linked into any particular subarchitecture, but are competence independent and represent knowledge that is relevant across both the collection of subarchitectures and the general architecture mechanisms (such as global goals, coarse conceptions of space and time, plan structures etc.). The general ontologies can be used for general-purpose planning and reasoning, motive generation and other tasks that must cut across subarchitectures.

To embed the knowledge of the subarchitectures within the knowledge framework of the whole architecture, concepts within the subarchitecture ontologies that are re-

---

[3]We use the term "ontology" to mean a (structured) description of the kinds of information that the cognitive system can process from either internal or external sources. This is not the same as an ontology that could be used by the *designer* of such systems, cf. CoSy Deliverables 2.1 & 2.2.

lated in some way to concepts within the general ontologies can be interconnected in various ways (e.g. action recognition models could be linked to action verbs and to planning actions via common activation). This allows general mechanisms and subarchitectures to interact with other subarchitectures via the mediating presence of the general ontology. In the previous work we have investigated particular ways in which such interconnections could be made (e.g. [Kruijff et al., 2006c]), but for our current architecture schema we prefer to leave the mechanisms unspecified as they are likely to vary a great deal across scenario-specific instantiations of the abstract schema.

In terms of engineering a working system, ontologies provide a useful mechanism for separating interface and implementation. They allow us to specify the kinds of information the architecture must be able to obtain and process, without initially specifying exactly how these components are actually implemented. This provides the flexibility necessary to collaboratively develop integrated systems in which some part of the ontology may be learnt, or developed at a different rate from the rest of the system (provided suitable short-term replacements can be implemented). In terms of a scientific study of a cognitive system, we can point to the ontologies as a way of demonstrating the scope of the system's competences. This will allow us to describe the generality and limitations of particular instantiations of the schema.

5.3   Dynamic Contextual Processing

In order to satisfy our requirement of the architecture supporting goal-directed behaviour, and given our previous design decision to use multiple concurrent processing components, we must provide mechanisms to control which components can influence processing and when. We propose the following way in which a component in our architecture can be controlled. Components must announce their intent to perform some processing by posting a goal stating what they want to do (and possibly why). An additional control mechanism can then monitor the current state of the architecture and determine whether the goal should be adopted or not. A side-effect of this approach to control is that there can be no fixed path of information flow through the architecture as components may not always be operational, or permitted to process when they can.

To support this dynamic approach to information processing, our architecture distinguishes between two types of goal:

1. **Global Goals**: These are the goals that control the global information processing behaviour of the architecture. Once a global goal has been adopted it can be considered as the ultimate justification of the system's subsequent behaviour. Typical global goals may be to explore the world (i.e. gather as much information as possible) or to respond to an utterance (either just linguistically or through action or using a combination of both). Global goals will typically involve coordination across multiple subarchitectures.

2. **Local Goals**: The presence in a subarchitecture's working memory of a particular piece of information will trigger the proposal of local goals by the subarchitecture components that can process the information. For example, a parser will propose a goal of extracting a logical form from an utterance, or an edge detector will propose a goal of extracting all the edges from a region of interest.

Within the architecture, it is the role of the *subarchitecture task managers* to determine whether local goals should be adopted or not. This decision can also be made by the *global goal manager* which must also make decisions about global goals for the entire system. These decisions will depend on the current global goals, and what other demands are currently being made on the architecture. This will require these components to have a knowledge of what the architecture is currently doing, which things can happen in parallel, and, more importantly, how the results of particular processes contribute towards the satisfaction of global goals. This final type of knowledge could either be statically defined, or derived by a deliberative system such as a planner.

6   Subarchitecture Design

The core of the architecture design is the manner in which local processes in a subarchitecture interact via a local working memory. This section describes the general template for this interaction.

As mentioned in Section 5.3 to make the behaviour of subarchitectures as flexible as possible whilst being controllable (e.g. to allow attentional mechanisms to have an effect) we propose that every time a processing step can be undertaken by a subarchitecture component, it must propose this processing step as a *local goal*. For example, if a locomotion subarchitecture contains a description of a point that must be reached, and a component within the subarchitecture can move the system to that point, then this component must propose a goal to that effect.

Depending on the current control profile of the architecture these local goals may be automatically adopted or rejected, or they may get stored until the necessary resources are available (or some other condition is met). This level of indirection also allows different components to process the same information in different ways with an appropriate mechanism selecting which one should be active (or if they all should). Such an approach may appear to add a large amount of overhead to the processing of the architecture, but in the long term it will allow us to scale architecture implementations to bigger problems, and to investigate various methods of control, without having to manage the complex dependencies that develop when processing components are directly, and possibly arbitrarily, connected.

We propose that subarchitectures should be designed along the lines of Figure 1 (the rest of the global components are not shown), with each subarchitecture composed of the following components:

- **Subarchitecture Working Memory**: The subarchitecture specific working memory holds the results of the processes contained within the subarchitecture. Depending on the specific scenario, subarchitecture working memories could be augmented to maintain a local history, or assign levels of activation to processing results and related information. Entries in the working memory should conform to the specified ontology of the subarchitecture.

- **Subarchitecture Task Manager**: The subarchitecture task manager is the component that manages the subarchitecture's local goals to control which of the processing components are active. Processing components within the subarchitecture must post local goals to this task manager when they want to act. These
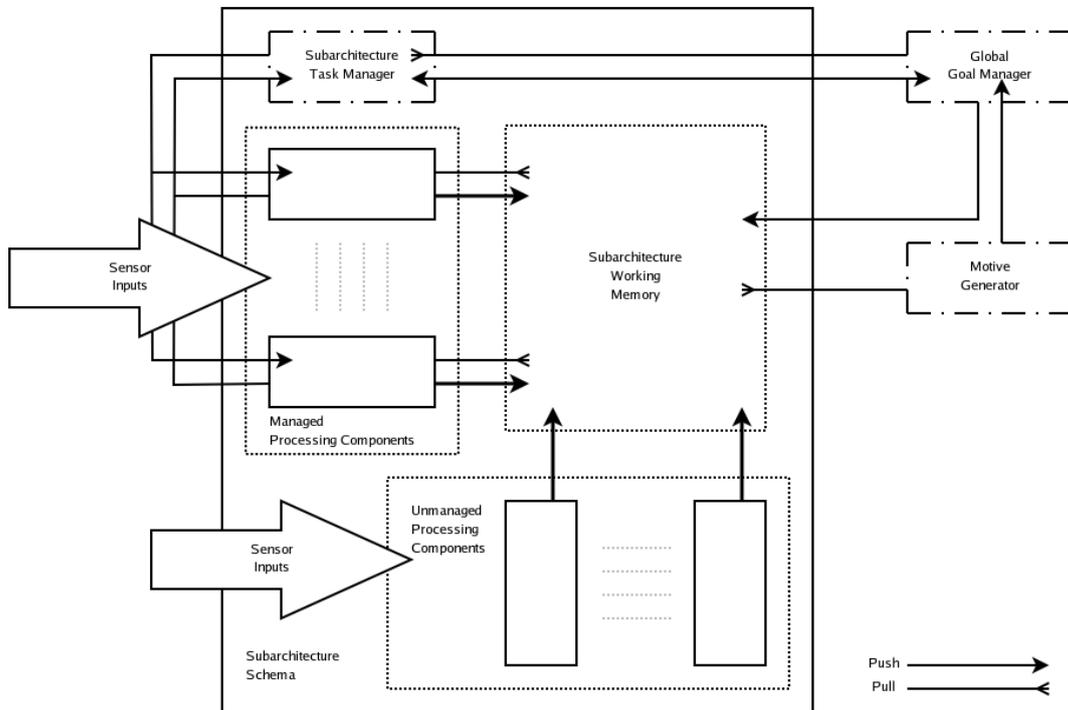
Figure 1: Subarchitecture Schema

goals are then either adopted, rejected, or stored and ordered in some way (which can be scenario dependent). This is described in more detail in Section 7.2.

- **Unmanaged Processing Components**: Each subarchitecture can have a set of zero or more unmanaged processing components which monitor data external to the subarchitecture for simple, easily determined cues for further processing within the subarchitecture. These will typically be linked to a system's sensors. These components can add information to the working memory *without* proposing a local goal. Examples of such components may be a motion detector in a visual subarchitecture or a speech-detector in a linguistic subarchitecture. unmanaged processing components will sit withing the reactive tier of the CogAff architecture schema [Sloman, 2001].

- **Managed Processing Components**: Each subarchitecture can also have a set of zero or more managed processing components. Each of these must monitor the subarchitecture working memory for information that it can process. When it recognises such information it must generate a local goal that indicates what it intends to do. This goal is posted to the subarchitecture task manager, and the component must then wait for it to be adopted.

One part of the subarchitecture design which currently needs further development is how motor processes, and related behaviour-based control styles, can be integrated in a principled manner. Currently the route of information into the subarchitecture and onwards into the wider architecture can be explicitly discussed in terms of the schema

components, but such a discussion is not yet possible in terms of subarchitecture outputs into a robotic platform. It is possible that managed components could enclose motor controllers and related methods of generating physical output from the system. In this case the overall control of output signals would rest with the control mechanisms for the managed components (principally the subarchitecture task manager).

7   Goals, Control and Memory

The previous section described how processing occurs within a single subarchitecture. A crucial aspect of any architecture is how it integrates processing across the entire architecture. In this case this involves how the results of processing in one subarchitecture can trigger processing in a different subarchitecture. This requires three additional components that are connected to all subarchitectures: the motive generator, the global goal-manager, and general memory.

7.1   The Motive Generator

The motive generator is connected to the working memory in every subarchitecture. Its role in the architecture is to monitor these working memories for information that may be able to trigger processing in a different subarchitecture, or subarchitectures. When the motive generator notices such information, it uses it to generate a new *global goal* for the whole architecture. New goals are pushed into the global goal manager. For example, the motive generator may notice that a linguistic subarchitecture has fully interpreted an utterance as a command and may then generate a global goal to follow this command, or it may notice that a planning subarchitecture has completed a plan and may then generate a global goal to execute this plan.

In previous work we have argued that a mechanism for quickly generating new goals is important if an agent must keep up with a dynamic, rapidly-changing world [Beaudoin and Sloman, 1993]. This is because it is essential that decision making processes within the architecture are aware of the myriad possibilities for future behaviour as soon as they can be predicted, even if these possibilities are not acted on (either immediately or ever). Because of this need for speed, we propose that the motive generator be a reactive component in terms of the CogAff schema. This requires that it should make use of fast, forward-chaining rules or pattern-matching, rather than slower deliberative capabilities. Crucially, this means that the motive generator will not be able to consider the implications of the architecture's current state in much detail and should therefore propose goals regardless of much of the architecture's current state (i.e. its current set of goals, beliefs, processes etc.).

7.2   The Global Goal Manager and Subarchitecture Task Managers

Because the motive generator must necessarily be ignorant of the state of the architecture beyond the aspects of information required to generate new global goals, an additional component is required to manage the goals it proposes. This is the role of the goal manager. Our architecture schema does not specify the exact nature of this

management process, but the minimum required abilities will be to accept or reject proposed global goals. In addition it will be useful if the goal manager can store goals for later adoption, perhaps when the current goal is no longer valid or other conditions are met. For a detailed analysis of a broad range of goal management tasks see [Beaudoin, 1994].

Once a global goal has been adopted, the goal manager must then trigger the behaviour required to achieve this goal. To do this it must write some suitable data into the working memory of the appropriate subarchitecture. This will then produce a local goal from a processing component in the subarchitecture and start a chain of processing in that subarchitecture. This is a similar behaviour to an unmanaged component writing information into it subarchitecture working memory, except that in this case the source of the information is the result of a process in another subarchitecture.

To be able to manage goals successfully, the global goal manager must have access to information on various aspects of the architecture's current state. Exactly how the goal manager operates should be scenario-specific (although perhaps general enough to be readily reusable), but the minimum of information it should have access to are the local goals currently being pursued by the subarchitectures, and the goals it has adopted in the past (which may still be being pursued). The latter can be recorded internally to the goal manager, but the former must be obtained by connecting the global goal manager to each of the subarchitecture task managers. Each subarchitecture task manager can then inform the global goal manager when it adopts a local goal and when the subsequent processing is completed.

To allow the goal manager to exert more control over the processing in the subarchitectures we also propose that the global goal manager can use this connection to the subarchitecture task managers to make decisions on the management of local goals. This could be done on a goal-by-goal basis where appropriate (where the global goal manager has knowledge from the wider architecture that is relevant, for example), or it could be done by setting management profiles in the subarchitecture task managers that control how they operate. Examples of such profiles could be setting the subarchitecture task managers to reject all goals, reject all goals that involve physical action, immediately accept all goals that involve a particular object, or queue all goals until a particular state is reached.

The final necessary interaction between the global goal manager and the subarchitecture task managers is for the subarchitecture task managers to signal when a chain of processing initiated by the global goal manager is complete. This behaviour will allow the global goal manager to move on to a subsequent processing step, perhaps by adopting another goal or trigger behaviour in another subarchitecture to pursue the current goal. As discussed previously, a chain of processing will be started in a subarchitecture by the global goal manager writing appropriate information into its working memory as a result of adopting a global goal. To be able to signal the completion of processing, the subarchitecture task manager must be able to determine when all the necessary processing on this information has been done. A useful heuristic for this will be when it cannot adopt any further local goals relating to the initial information. The signal sent back to the global goal manager when this state has been reached may indicate success or failure, or it may just indicate that no further processing can be done.

7.3 The General Memory

The general memory in the architecture stores long-term knowledge about concepts, types, processes, competences, beliefs, goals and anything else that may be of general use to processing components in an instantiation of the architecture schema. In this current iteration of the architecture schema we are not specifying any structure for the general memory. This means that it could be split into many structured, special-purpose memories for different types of information, or it could be one large repository for everything the system considers worth storing in a generally accessible place. The main restriction should be that the contents of general memory are relatively static when compared to the contents of subarchitecture working memories. For more a discussion of more dynamic memories see Section 8.1.

8   Interactions Between Subarchitectures

The arrangement of processes into subarchitectures is a fairly general way of viewing groups of related processes. In terms of complete architecture instantiations there may be various ways in which subarchitectures may need to interact. In addition to operating in isolation, subarchitectures may abstract across the information produced by combinations of other subarchitectures (cf. Section 8.1), or take part in tightly-coupled task-specific interactions with other subarchitectures (cf. Section 8.2).

The most basic use of the subarchitecture design approach is to group processes that only directly interact with each other. These components will not interact with other components in the rest of the architecture instantiation (except the general ones such as general memory). This will give rise to closely controlled processing loops that are the typically required for modality-specific information processing (e.g. extracting information from visual or linguistic input). Within our architecture schema, these kinds of subarchitectures can be created by directly instantiating the subarchitecture design described in Section 6.

8.1   Subarchitectures for Active Memories

One way of viewing information-processing architectures for intelligent systems is as a two-level collection of subarchitectures with the division based on the primary sources of information the subarchitectures process. On the first level we can group subarchitectures that process information closely tied to the system's inputs and outputs. On the second level we can group subarchitectures that process information produced by other subarchitectures and provide information for other subarchitectures to work with. Because of the role these second type of subarchitectures typically play in instantiations of the schema, we will refer to them as *active memory* subarchitectures. This name reflects that their purpose is to provide structured information to the rest of the architecture (the "memory" part), but that this information is the result of "active" processing, rather than retrieval from a static store of knowledge.

Although this division is an over-simplification of the reality of both natural and artificial intelligent systems, it allows us develop a view of how our architecture schema can be used to characterise the processes that result in a system having views of the

world, and itself, that are not tied directly to a single input modality. An example of this is the approach taken to grounding language in other input modalities by the CoSy project [Kruijff et al., 2006b]. In this work modal information is produced by subarchitectures directly concerned with a single input modality, and then an additional, cross-modal, subarchitecture binds information across these modalities using knowledge about the cross-modal correspondences. These bindings can then be used within other subarchitectures to influence processing. Another example of an active memory subarchitecture is the spatial working memory we are currently researching for use in the CoSy demonstrator systems. In this proposed subarchitecture high-level representations of space are generated from detailed modality-specific spatial information obtained from input and output subarchitectures (primarily vision and manipulation).

Within our architecture schema, active memory subarchitectures can be created by directly instantiating an extension of the subarchitecture design described in Section 6. The extensions required to this design is that processing components within the subarchitecture can read from the working memories of other subarchitectures, and that components in other subarchitectures can read from the working memory in the active memory subarchitecture. Although this violates one of the primary tenets of our design, it is justifiable because of the expected level of detail and rate of change of the information involved.

The key difference between the information in the working memories of active memory subarchitectures and standard subarchitectures is how it is accessed and by which processes. The standard subarchitecture working memories are designed to be changing rapidly and therefore constantly monitored by subarchitecture components for changes. The active memory subarchitecture working memories are not intended to change so rapidly (although we do not specify any limit on this). Because of this it is not intended that they be explicitly monitored, but rather specific information should be pulled from them when required.

Although the contents of working memory in active memory subarchitectures will mostly be determined by the details of the scenario-specific architecture and the scenario itself, the schema should present some guidelines for this because of the privileges granted (in information-processing terms) to these subarchitectures.

One key point is that the contents of the working memory should be potentially useful to more than one subarchitecture in an instantiated architecture. For example, a high-level description of the current location of the system may be useful to many subarchitectures, but the current laser range profile of the location may be much less so. Related to this is the level of description of the contents of working memory in an active memory subarchitecture. Information should be presented in a form that is understandable to other subarchitectures, not just a single subarchitecture. For example, an object description abstracted from information in the visual subarchitecture and represented in terms of coarse properties and shape may be generally understandable as these facts may have close analogues in other subarchitectures, but a description in terms of a point cloud and HSV values may be much less generally understandable due to the specific nature of the representation.

In addition to this, we believe that the contents of working memory in active memory subarchitectures should change at a frequency that makes the information usable for high-level deliberative reasoning. This necessitates a choice of a representation that either abstracts away frequently changing details, or one that only repre-

sent relatively stable information. This observation is again closely tied to the level of detail of the chosen information and representation (as mentioned above). For a discussion related to representing changeable information in a dynamic world see [Wood, 1993, Hawes, 2004].

Elsewhere we have argued that complex embodied cognitive systems require such amodal representations which these constraints give rise to [Sloman et al., 2006]. They are necessary to ease the complexity and overhead involved in integrating the results of heterogeneous processes, whether they abstract across different types of information, represent the same thing in different ways, or are used for sending one item of information to various processes.

8.2   Specialised Subarchitecture Connections

In addition to the mechanisms for information processing described previously, we are aware that we may need to make allowances for specialised connections between subarchitectures that either circumvent some of the communication mechanisms of the schema, or are unaffected by parts of them. For example, if an architecture for an agent that must grasp objects separates its visual processes and its manipulative processes into separate subarchitectures, how does the latter subarchitecture get access to the specialised metrical information about object surface patches and position that it needs to carry out the grasping, when we've already specified that such detailed, potentially changeable information should not be placed into a generally accessible memory. Also, in incremental utterance understanding the behaviour of a number of components need to be tightly coupled. If all of these components do not share a subarchitecture (perhaps because they do not always work together), should they still share the partial results of processing via a general memory?

In general we are interested in ways that subarchitectures can quickly exchange detailed, specialised information that may change rapidly. This exchange should not rely completely on the central management mechanisms (as it may overwhelm them), but it also should not be done in a completely ad-hoc manner. We currently view this as an open question, but one of crucial importance to the overall design (as such behaviour is central to many operations of cognitive systems). In the following paragraphs we discuss a number of possible solutions that exist to this problem.

One approach to direct information exchange between subarchitectures that does not deviate too far from the previously described architecture is to allow subarchitectures to pass pointers via general memory to information within their working memories. Other subarchitectures could use these pointers to access the information they required directly, without invoking any other mechanisms. This appears to be a powerful mechanism that maintains all of the other structural constraints previously imposed. The downside is that information that may need to be processed together still remains separate in this approach, and it does not seem an efficient solution for two-way information sharing between two subarchitectures.

A similar approach that allows related information to be collected together involves having an additional process that copies the relevant information between the working memories. This would deal with the previously identified issue of information separation, but it would result in the duplication of the copied information. This could potentially cause problems with copies of information being changed in different ways

simultaneously in different subarchitectures. This would result in the architecture being put into an inconsistent state.

An approach that alters the structure of the architecture, but does not face the separation and consistency problems of the previously described approaches, is for subarchitectures that occasionally perform tightly-coupled processing to share a working-memory specifically for these processes. In the previous example, this would mean that a visual subarchitecture and a manipulation subarchitecture would share a working memory specifically for visual servoing. These shared working memories would be treated like standard working-memories by the managed components within the attached subarchitectures. If data is processed from a shared working memory, then the results are placed back into the same memory.

For the current iteration of the schema we will use the approach of allowing shared working memories for specialised connections between subarchitectures. Future implementations based on the architecture schema will be used to evaluate this approach.

9 Examples of Information Processing

In a subarchitecture based on our schema, a typical processing chain will look something like the list below. In this example, the format and other presentation details are unimportant to the content, and the names of goals and types of data are defined by ontologies.

1. An unmanaged component extracts some raw data from a sensor, and adds it to `subarchitecture1`'s working memory: `[raw_data data]`. Where `raw_data` is be the name of an entry from the subarchitecture ontology, and `data` is the actual data.

2. A managed component in `subarchitecture1` notices `raw_data` and posts a goal to the subarchitecture task manager to process it:
`[lcl_goal lcl_goal_1 process_raw_data [raw_data data]]`.
Where `lcl_goal_1` is the id of the goal and `process_raw_data` is the name of the information processing goal.

3. Through some decision process the subarchitecture task manager adopts this information processing goal. To signify this it sends the following back to the component: `[adopt lcl_goal_1]`.

4. The processing component then performs the processing. The result of this processing is that `[processed_data pdata]` is added to the working memory.

It is in this manner that subarchitectures can go through multiple processing steps for a single piece of data. Depending on the nature of the problem, data and components, this approach allows components to run in parallel if required by the subarchitecture task manager.

The following list gives an example of how the global components interact with the subarchitectures to move information around and trigger processing in different subarchitectures.

1. The motive generator notices `[processed_data pdata]` in `subarchitecture1`'s working memory and posts a global goal to the global goal manager to process it:
   `[gblgoal gblgoal_1 act_on [processed_data pdata]]`

2. At the same time the motive generator moves the `[processed_data pdata]` into the general memory, so that it is accessible to other subarchitectures.

3. Through some decision process the global goal manager adopts the proposed global goal. To act on the goal it must trigger some behaviour in another subarchitecture.

4. To trigger behaviour in `subarchitecture2` the goal manager writes `[processed_data pdata]]` into `subarchitecture2`'s working memory.

5. At this point the managed components in `subarchitecture2` notice the change and can propose local goals if appropriate.

The above is the simplest information processing strategy for the whole architecture. Additional complexity could occur with processing happening in parallel, multi-step global goals, specialised connections between working memories etc.

10 Related Work

When comparing the work presented in this document to the work in the literature there are a number of places to start and a number of different ways comparisons can be drawn. Because our work is difficult to compare directly to much of the other work that potential readers may judge as relevant, we will eschew this approach and instead discuss how our work relates to the broad trends evident in the literature.

The term "architecture" has been defined many times in relation to systems developed to demonstrate capabilities that could be judged as cognitive or intelligent. Two recent surveys cover most of the relevant ways in which the term has been used in the past [Langley et al., 2006, Vernon et al., In Press]. What most of these uses of the term agree on is that "architecture" refers to some kind of underlying structure or organisation in a system. We also use the term is this sense. Where groups of researchers start to differ is on the additional meanings of the word. Cognitivists use the term to refer to unifying theory of mind, including specifications of all relevant aspects of cognition (learning, attention, memory etc.). This includes two of the most well-known cognitive architectures: ACT-R [Anderson et al., 2004] and SOAR [Laird et al., 1987]. Although other uses of the term "architecture" are not so all-inclusive, they do usually refer to those aspects of a cognitive or information-processing system that remain *constant* over time. Whilst this definition is closer to the definition we use than the "unified theory of mind" one, we believe that the structural elements that make up the architecture of an information-processing system do not have to remain constant over time, but can develop as the system itself develops. To quote Vernon et. al who use the following to describe the architecture of emergent systems, the resources represented by the architecture are "the initial point of departure for the cognitive system and they provide

the basis and mechanism for its subsequent autonomous development, a development that may impact directly on the architecture itself" [Vernon et al., In Press].

Disagreements over what should be covered by an architecture lead us to discuss the distinction we drew in Section 2 between an architecture schema, and scenario-specific instantiations of such a schema. Because our main scientific objective is the study of design options and trade-offs between architectures, we restrict our comparison to related work to those works that are explicit in presenting an architecture independent of any particular scenario or implementation. Myriad other architectures exist that are implicit in such implementations, including some that offer behaviour that demonstrates the integration of capabilities perhaps in advance of those offered by the architectures discussed below (e.g. Kismet [Breazeal, 2003]). Unfortunately, comparisons between such architectures and those that are explicitly designed as abstract templates for information-processing systems are not easy to make or are potentially inaccurate because it is impossible to draw a line between the scenario-specific and non-scenario-specific elements without input from the original authors.

Although most existing work on architectures does not make explicit the distinction presented in Section 2, it is often possible to infer the set of requirements the work is based on. It is possible to divide the majority of the work into two camps based on requirements: human performance and robot performance. Within the human performance camp are cognitive architectures largely aimed at enabling an often disembodied system to produce human, or human-like, performance on a particular set of tasks, e.g. CLARION [Sun et al., 2001], the Global Workspace Architecture [Shanahan, 2005], ACT-R and SOAR. This means that these architecture schemata include all of the mechanisms and assumptions that the authors view as necessary for instantiations of the architectures to demonstrate human-like performance. Within the robot performance camp are cognitive architectures largely aimed at enabling a robotic system to complete a usually well-specified task in the physical work such as collecting cans, or moving around a building, e.g. 3T [Bonasso et al., 1997, Gat, 1997], ICARUS [Langley and Choi, 2006], and the Subsumption Architecture [Brooks, 1986]. These architectures are typically focused around producing robust actions in the world, and therefore mechanisms to enable this are usually prominent in their designs (e.g. RAPs [Firby, 1987]).

The preceding discussion is not intended as a criticism of the referenced architectures, as each one is more-or-less explicit about their target scenarios and the effects this has on their architecture designs. What we wish to highlight is that all of these architectures were designed with a particular purpose in mind, and this should be considered before they are used for any other purpose. None of the architectures mentioned above are general-purpose approaches for creating information-processing systems, just as neither is the architecture we present in this document. As such researchers should not just blindly pick an existing architecture for their system without first considering their requirements.

A high-level description of the desired performance of our system was described in Section 3 and from this the requirements for the underlying architecture were distilled into three design principles. These were presented in Section 5. The following paragraphs examine existing architectures with reference to these principles. It is also worth noting before this that the aims of our research, and consequently the requirements for our systems, sit between the two camps discussed above: we want to be able to produce robust robotic behaviour, but we are also interested in this behaviour

demonstrating properties that could be described as human-like (although we are not aiming to replicate human behaviour).

The design principle that has perhaps the biggest mismatch with existing architectures is that of concurrent modular processing (Section 5.1). We view it as critical that elements of an architecture can be active in parallel (so that a system can handle asynchronous inputs from multiple sensors whilst solving a problem, managing resources and coordinating action across different modalities, for example), but the significant majority of existing architectures are essentially monolithic in their styles of processing. By this we mean that the architectures only support a single process occurring at a time. This is true for most of the architectures mentioned so far including ACT-R, SOAR, ICARUS, 3T, the Subsumption Architecture and CLARION. It is also true of many other architectures that have a similar coarse structure to these, such as Prodigy [Veloso et al., 1995]. One main exception to this list is the Global Workspace Architecture, where concurrently active processes are central to its overall processing style.

Conversely, concurrently actively modules are a common feature of implemented robotic and software systems where abstract architecture research has not been made a prime concern. This includes examples such as Kismet [Breazeal, 2003], BIRON [Haasch et al., 2004], and many other systems that attempt to achieve tasks whilst interacting with a dynamic and unpredictable world. This goes some way to (perhaps anecdotally) demonstrating the importance of the design principle of concurrently active processes.

Our second design principle is the structured management of knowledge (Section 5.2). For this we require that processes within an architecture are able to use problem specific representations as required, and then any knowledge that is globally useful to the architecture must be translated into a single central representation as necessary.

Many existing architectures postulate a single central representation, but they vary greatly on the other representations catered for. For example, ACT-R and SOAR can again be considered monolithic, but this time in terms of representation. In both systems all knowledge must be encoded in their preferred forms of representation, with external processes translating information to and from this representation to support interaction external to the system. There are a number of issues that can be raised with this, but crucially for us this means that these external processes are not considered within the architecture and therefore their nature, and their interactions with the rest of the system, cannot be studied through the study of architecture. This perhaps reflects these architectures' cognitivist stance which posits cognition as a central process only indirectly coupled to input-output processes [Vernon et al., In Press].

Perhaps the most common approach to including more than one representation in architectures is to couple a high-level logical representation that can be used for planning and reasoning to a behaviour-based system with problem-specific representations that are used for action tasks. This is an approach taken by many architectures including 3T, ICARUS and CEREBUS [Horswill, 2001]. This express reliance on behaviour-based systems, and the use of problem-specific representations almost exclusively for action, reflect the motivation of such architectures as methods for generating robust behaviour from robotic systems.

Our architecture schema is in essence a generalisation of the approach taken by the robotics-motivated architectures.

Some of the researchers behind these systems realised that a specific task (affecting change in an unpredictable world through physical action) might require different specific representations within concurrently operating different subsystems performing different information-processing sub-tasks required for the single external task, such as combinations of visual servoing along with condition checking in plan execution. Visual perception and speech understanding tasks also obviously require different kinds of representations for different sub-processes. The consequence of this is that to meet our requirements, our architecture schema must make allowances for processes, or groups of processes, to use representations that are not generally understood by the rest of the architecture (this could include distributed or sub-symbolic representations). We also view the translation of parts of these representations into a generally understandable format as a problem that is central to understanding and developing integrated cognitive systems. This is why we require central representations (i.e. the representations used by generally accessible memories) and mechanisms for translating to and from then to be part of our architecture schema, rather than placing such processes outside of the architecture (as happens in architectures with monolithic representations)[4].

As with the previous design principle, multiple representations are common in implemented systems such as the work done previously by CoSy [Kruijff et al., 2006b, Hawes and Wyatt, 2006]. As such our work can be viewed as an attempt to formalise in an architecture schema what is already viewed as common practice in some research fields.

Our final design principle is that the concurrently active processes within the architecture must be controlled in a way that is subject to both problem-specific constraints and global constraints. Due to the fact that the majority of the other architectures do not have concurrently active processes, it is hard to relate their control methods directly to our requirement as specified. In the majority of cases, a central control loop selects a single process for activity or a single rule for firing. This is usually based on the evaluation of some logically specified preconditions, although in the case of ACT-R this is also based on the activation of information in memory [Anderson et al., 2004]. In the Global Workspace Architecture, which is one of the few architectures to specify concurrently active processes, a central controlling process is actively rejected [Shanahan, 2005]. Instead processes compete for access to the Global Workspace, where they get a chance to broadcast information to the rest of the architecture. This could in effect act like a control signal, but it could also be used to pass information around the architecture. This idea of competition between processes is similar to the mechanism we have specified for managed subarchitecture processes (in the case that there are multiple processes proposing the same goal), although we still have an separate arbitration mechanism (which could arbitrate through competition of some sort). This idea of multiple competing processes is also included in other architectures, for example the Subsumption Architecture [Brooks, 1986].

---

[4]We do not wish to be drawn into arguments about the nature of symbols, so for this discussion we have focused purely on the engineering consequences of allowing various representations and a single central representation to co-exist in a single architecture.

11 Scenario-Specific Instantiations of the Architecture Schema

This section of the document presents subarchitecture-level descriptions of examples of possible scenario-specific instantiations of the previously presented architecture schema. Two instantiations are presented, one each for simplified versions of the CoSy demonstrator scenarios in which a human can order the robot to perform a simple task. Following this a more in-depth look at the design is provided by a description of the information flow within a single subarchitecture.
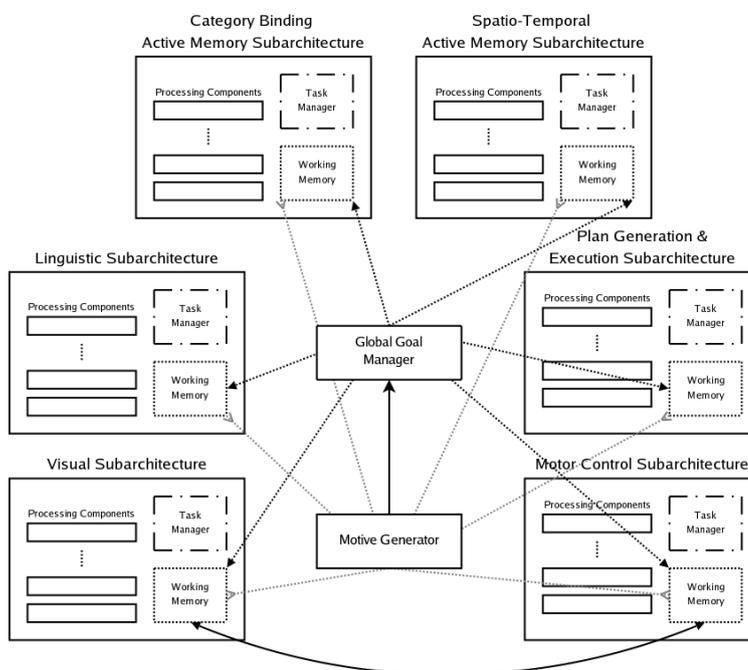
11.1 A PlayMate Instantiation

Figure 2: A PlayMate Instantiation of the Architecture Schema

Figure 2 presents a schematic overview of an instantiation of the previously presented architecture schema. The instantiation represents an architecture for a robot to take part in a much simplified version of the CoSy PlayMate scenario. In this version the only allowable interaction is for a human to command the PlayMate robot to move objects around on a table. The general roles of the subarchitectures and general components in the instantiation are described in the following paragraphs.

Input to the system is handled by the two modal-specific subarchitectures: the visual subarchitecture and the linguistic subarchitecture. The role of the visual subarchitecture is to extract information from the world that can be used for identifying objects that the human may refer to in a command, locate them in space to allow the human to use spatial prepositions in their action commands, and identify surfaces that are gras-

pable to support manipulation. The role of the linguistic subarchitecture is to interpret the human's utterances as part of a situated dialogue about the current scene, and also to allow the PlayMate to make utterances about its world. To allow the PlayMate to understand utterances in terms of the visual world (i.e. to support the grounding of language in vision), the category binding active memory subarchitecture binds object references from both of the modal subarchitectures together into an amodal representation of the referred to entity [Kruijff et al., 2006c]. In parallel with this, the spatio-temporal active memory subarchitecture abstracts symbolic spatial relationships that describe the current scene, allowing the linguistic subarchitecture to ground utterances with respect to spatial prepositions as well as object references [Kelleher and Kruijff, 2005]. It is the role of the plan generation and execution subarchitecture to use information from both of the active memories to generate a problem description suitable for planning which is then either fed into a planner to generate a manipulation plan to satisfy the human's commands [Brenner et al., 2007], or used as input to a plan execution monitoring component that examines whether the preconditions and effects of the plan steps are satisfied in the world as appropriate. The motor control subarchitecture uses information pulled from the visual subarchitecture via a specialised connection to perform the manipulation tasks specified by the plans produced in response to the human's utterances.

In this instantiation, the role of the motive generator and global goal manager is to transfer information and control around the subarchitectures to allow them to perform the particular tasks required to respond to an action command. First, an action command in the linguistic subarchitecture must be noticed and a global goal to act on this command must be proposed and accepted. Second, this acceptance must be translated into the necessary input to the plan generation and execution subarchitecture to trigger a plan to be formed. Finally the resulting plan must be sent to the motor control subarchitecture for execution, whilst this execution is monitored in the plan generation and execution subarchitecture.

11.2 An Explorer Instantiation

Figure 3 presents a schematic overview of an instantiation of the previously presented architecture schema. The instantiation represents an architecture for a robot to take part in a much simplified version of the CoSy Explorer scenario. In this version the only allowable interaction is for a human to command the Explorer robot to move to a particular room in the current building.

This Explorer instantiation of the schema is similar to the previously described PlayMate instantiation in many ways (given the simplified nature of the scenario). The role of the linguistic subarchitecture is identical across both instantiations (i.e. interpreting action commands), and where the PlayMate instantiation receives additional input from just vision, the Explorer receives input from vision and a localisation and mapping subarchitecture. This subarchitecture builds maps from range data available from laser scans of its environment and uses this to localise itself in the world. Given this new input modality subarchitecture the roles of the two active memory subarchitectures can be extended. The category binding active memory subarchitecture must now also bind linguistic references to spatial areas to information within the mapping and localisation subarchitecture [Kruijff et al., 2006a]. The spatio-temporal active memory subarchitecture must now use the mapping and localisation subarchitecture as an addi-
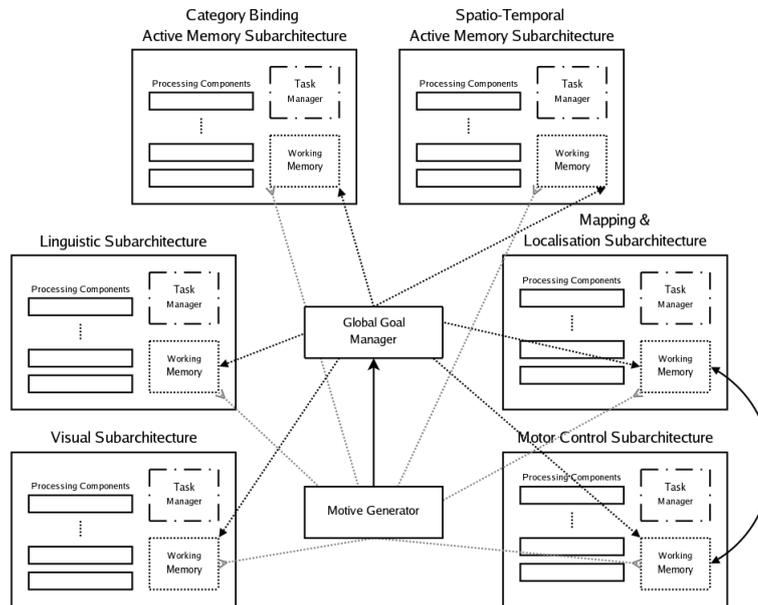
Figure 3: An Explorer Instantiation of the Architecture Schema

tional source to extract spatial relationships from. In this Explorer instantiation the role of the motor control subarchitecture is to move the robot from its current position to target the position given by the human and bound into the mapping subarchitecture via the category binding subarchitecture. This behaviour is closely coupled with the mapping and localisation subarchitecture (as the Explorer robot must relate its movements to its position in the world), so, much like the PlayMate, a specialised communication connection is required between these two subarchitectures.

As with the PlayMate instantiation the role of the central components is to ensure that information and control is passed to the appropriate subarchitectures for the processing steps required to act on the human's command. First this involves the motive generator proposing a global goal to follow the human's command. Then this must result in the necessary information being written to the localisation and mapping subarchitecture in order for a new route to be generated to the target point. Finally a global goal must be proposed and accepted to transfer this route to the motor control subarchitecture to generate the necessary behaviour.

11.3   A Plan Generation Subarchitecture in Detail

To get a more detailed view of how a subarchitecture processes information we can examine a hypothetical plan generation subarchitecture based on some previous CoSy work [Brenner et al., 2007]. Given an interpreted action command as input from a linguistic subarchitecture a planning subarchitecture based on this work would need to extract a planning goal from the command, then generate an initial state for the planning problem, and finally use all this generated information as input to a planning process to produce a plan for achieving the goal. To achieve this behaviour our planning subarchitecture will contain three managed components: a planning goal generator com-

ponent, and planning state generator component and a planner component. In addition the subarchitecture will contain a subarchitecture task manager and the subarchitecture working memory.

Assuming a surrounding architecture instantiation much like the one described in Section 11.1, processing in the subarchitecture is triggered by the global goal manager writing an interpreted linguistic action command into the subarchitecture working memory. This change to working memory is detected by the planning goal generator component which subsequently proposes a local goal describing its intention to extract a planning goal from the command to the subarchitecture task manager. The task manager accepts this and the component subsequently extracts a planning goal from the command and writes it to subarchitecture working memory. This change triggers the planning state generator to propose a local goal to generate an initial state for this planning goal. This is accepted by the task manager (perhaps after confirming that the planning goal generator has been run previously), and the component is allowed to run. To generate the initial state for the planning problem the component must obtain information from the architecture about the types and attributes of the objects that are to be included in the initial state. Again assuming a surrounding architecture instantiation like the one described in Section 11.1, this can be done by pulling a representation of the current scene from the spatio-temporal active memory subarchitecture and categorical information about the objects in this representation from the category binding subarchitecture. Using this information the component generates the initial state and writes it to working memory. Finally the planning component notices this change to working memory and proposes a local goal to produce a plan for the most recent planning goal using the most recent initial state. This is accepted by the subarchitecture task manager and the planner runs and adds the resulting plan to the subarchitecture working memory. This working memory change could then be noticed by the architecture's motive generator which could propose a global goal for the system to execute the plan.

12   Conclusion & Future Work

In this paper we described an architecture schema which can be used to generate architecture instantiations for a limited set of scenarios (i.e. it is not a general model or intelligence or a completely general purpose architecture schema). We described both the design requirements that the schema is based on and how these requirements have been realised in our schema. In Section 10 we described how the presented schema is related to the current state-of-the-art in architectures research, positioning our working as distinct to both the monolithic architectures produced by cognitive scientists and the behaviour-focused architectures produced by roboticists.

We have yet to experimentally evaluate our schema, so this remains a major piece of future work. We intend to do this by profiling the behaviour of different scenario-specific instantiations of the schema under varying conditions, such as internal failures and varying types of change in the world. We are currently working towards this by developing a software toolkit that embodies the schema in software and simplifies the creation of scenario-specific instantiations of it on real robots using state-of-the-art processing components. In addition to this we also plan to further explore the integration of motor control and behaviour-based control methods into the schema. This

will allow us to extend the schema to handle a wider range of requirements based on physical behaviour. Finally, we intend to study the schema in both a theoretical way and a practical way (using the aforementioned toolkit) to generate examples of sub-architecture instantiation approaches that can be used to generate commonly required information-processing styles.

References

[Anderson et al., 2004] 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.

[Beaudoin, 1994] Beaudoin, L. P. (1994). *Goal Processing In Autonomous Agents*. PhD thesis, School of Computer Science, The University of Birmingham.

[Beaudoin and Sloman, 1993] Beaudoin, L. P. and Sloman, A. (1993). A study of motive processing and attention. In Sloman, A., Hogg, D., Humphreys, G., Ramsey, A., and Partridge, D., editors, *Prospects for Artificial Intelligence*, pages 229–238. IOS Press.

[Bonasso et al., 1997] Bonasso, R. P., Firby, R. J., Gat, E., Kortenkamp, D., Miller, D. P., and Slack, M. G. (1997). Experiences with an architecture for intelligent, reactive agents. *J. Exp. Theor. Artif. Intell.*, 9(2-3):237–256.

[Breazeal, 2003] Breazeal, C. (2003). Emotion and sociable humanoid robots. *Int. J. Hum.-Comput. Stud.*, 59(1-2):119–155.

[Brenner et al., 2007] Brenner, M., Hawes, N., Kelleher, J., and Wyatt, J. (2007). Mediating between qualitative and quantitative representations for task-orientated human-robot interaction. In *Proc. of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI)*, Hyderabad, India.

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

[Firby, 1987] Firby, R. J. (1987). An investigation into reactive planning in complex domains. In *Proceedings of The Sixth National Conference on Artificial Intelligence*, pages 202–206.

[Gat, 1997] Gat, E. (1997). On three-layer architectures. In Kortenkamp, D., Bonasso, R. P., and Murphy, R., editors, *Artificial Intelligence and Mobile Robots*. MIT/AAAI Press.

[Haasch et al., 2004] Haasch, A., Hohenner, S., Hüwel, S., Kleinehagenbrock, M., Lang, S., Toptsis, I., Fink, G. A., Fritsch, J., Wrede, B., and Sagerer, G. (2004). BIRON – The Bielefeld Robot Companion. In Prassler, E., Lawitzky, G., Fiorini, P., and Hägele, M., editors, *Proc. Int. Workshop on Advances in Service Robotics*, pages 27–32, Stuttgart, Germany. Fraunhofer IRB Verlag.

[Hawes, 2004] Hawes, N. (2004). *Anytime Deliberation for Computer Game Agents*. PhD thesis, School of Computer Science, University of Birmingham.

[Hawes et al., 2006] Hawes, N., Sloman, A., and Wyatt, J. (2006). Requirements & designs: Asking scientific questions about architectures. In *Proceedings of AISB '06: Adaptation in Artificial and Biological Systems*, volume 2, pages 52–55.

[Hawes and Wyatt, 2006] Hawes, N. and Wyatt, J. (2006). Towards context-sensitive visual attention. In *Proceedings of the Second International Cognitive Vision Workshop (ICVW06)*, Graz, Austria.

[Horswill, 2001] Horswill, I. (2001). Tagged behavior-based systems: Integrating cognition with embodied activity. *IEEE Intelligent Systems*, 16(5):30–37.

[Jones et al., 2006] Jones, R. M., Crossman, J. A., Lebiere, C., and Best, B. J. (2006). An abstract language for cognitive modeling. In *Proceedings of the Seventh International Conference on Cognitive Modeling*, pages 160–165, Trieste, Italy.

[Kelleher and Kruijff, 2005] Kelleher, J. D. and Kruijff, G.-J. M. (2005). A context-dependent model of proximity in physically situated environments. In *Proceedings of the Second ACL-SIGSEM workshop on the Linguistic Dimensions of Prepositions*, Colchester, Essex, UK.

[Kitano et al., 1999] Kitano, H., Tadokoro, S., Noda, I., Matsubara, H., Takahashi, T., Shinjou, A., and Shimada, S. (1999). Robocup rescue: Search and rescue in large-scale disasters as a domain for autonomous agents research. In *Proceedings of IEEE Conference on Man, Systems, and Cybernetics*.

[Kitano et al., 1997] Kitano, H., Tambe, M., Stone, P., Veloso, M., Coradeschi, S., Osawa, E., Matsubara, H., Noda, I., and Asada, M. (1997). The robocup synthetic agent challenge 97. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 24–30.

[Kruijff et al., 2006a] Kruijff, G., Zender, H., Jensfelt, P., and Christensen, H. (2006a). Clarification dialogues in human-augmented mapping. In *Human-Robot Interaction*, Salt lake City, UT. IEEE/ACM.

[Kruijff et al., 2006b] Kruijff, G.-J. M., Kelleher, J., Berginc, G., and Leonardis, A. (2006b). Structural descriptions in human-assisted robot visual learning. In *Proceedings of the 1st Annual Conference on Human-Robot Interaction (HRI'06)*, Salt Lake City, UT.

[Kruijff et al., 2006c] Kruijff, G.-J. M., Kelleher, J. D., and Hawes, N. (2006c). Information fusion for visual reference resolution in dynamic situated dialogue. In Andre, E., Dybkjaer, L., Minker, W., Neumann, H., and Weber, M., editors, *Perception and Interactive Technologies: International Tutorial and Research Workshop, PIT 2006*, volume 4021 of *Lecture Notes in Computer Science*, pages 117 – 128, Kloster Irsee, Germany. Springer Berlin / Heidelberg.

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

[Langley and Choi, 2006] Langley, P. and Choi, D. (2006). A unified cognitive architecture for physical agents. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, Boston. AAAI.

[Langley et al., 2006] Langley, P., Laird, J. E., and Rogers, S. (2006). Cognitive architectures: Research issues and challenges. Technical report, Computational Learning Laboratory, CSLI, Stanford University, CA.

[Shanahan, 2005] Shanahan, M. (2005). Consciousness, emotion, and imagination: A brain-inspired architecture for cognitive robotics. In *Proceedings of the AISB '05 Workshop: Next Generation Approaches to Machine Consciousness*, pages 26–35.

[Sloman, 1998] Sloman, A. (1998). The "semantics" of evolution: Trajectories and trade-offs in design space and niche space. In Coelho, H., editor, *Progress in Artificial Intelligence, 6th Iberoamerican Conference on AI (IBERAMIA)*, pages 27–38. Springer, Lecture Notes in Artificial Intelligence, Lisbon.

[Sloman, 2001] Sloman, A. (2001). Varieties of affect and the cogaff architecture schema. In *Proceedings of the AISB'01 Symposium on Emotion, Cognition and Affective Computing*, pages 1–10.

[Sloman, 2006] Sloman, A. (2006). Introduction to Symposium GC5: Architecture of Brain and Mind – Integrating high level cognitive processes with brain mechanisms and functions in a working robot. Technical Report COSY-TR-0602:, School of Computer Science, University of Birmingham, UK.

[Sloman et al., 2006] Sloman, A., Wyatt, J., Hawes, N., Chappell, J., and Kruijff, G.-J. M. (2006). Long Term Requirements for Cognitive Robotics. In *Cognitive Robotics: Papers from the 2006 AAAI Workshop: Technical Report WS-06-03, http://www.aaai.org/Library/Workshops/ws06-03.php*, pages 143–150, Menlo Park, CA. AAAI Press.

[Sun et al., 2001] Sun, R., Merrill, E., and Peterson, T. (2001). From implicit skills to explicit knowledge: a bottom-up model of skill learning. *Cognitive Science*, 25(2):203–244.

[Veloso et al., 1995] Veloso, M., Carbonell, J., Perez, A., Borrajo, D., Fink, E., and Blythe, J. (1995). Integrating planning and learning: The prodigy architecture. *Journal of Theoretical and Experimental Artificial Intelligence*, 7(1).

[Vernon et al., In Press] Vernon, D., Metta, G., and Sandini, G. (In Press). A survey of artificial cognitive systems: Implications for the autonomous development of mental capabilities in computational agents. *IEEE Transactions on Evolutionary Computation, Special Issue on Autonomous Mental Development*.

[Wood, 1993] Wood, S. (1993). *Planning and Decision Making in Dynamic Domains*. Ellis Horwood.