EU FP7 CogX
ICT-215181
May 1 2008 (52months)

DR 1.1:
Motive Management

Nick Hawes[1], Hendrik Zender[2], Kristoffer Sjöö[3], Michael Brenner[4], Geert-Jan M. Kruijff[2] and Patric Jensfelt[3]

[1] *Intelligent Robotics Lab, School of Computer Science, University of Birmingham*

[2] *Language Technology Lab, German Research Center for Artificial Intelligence (DFKI), Saarbrücken, Germany*

[3] *Centre for Autonomous Systems, Royal Institute of Technology (KTH), Stockholm, Sweden*

[4] *Institute for Computer Science, Albert-Ludwigs-Universität, Freiburg, Germany*

⟨n.a.hawes@cs.bham.ac.uk⟩

Due date of deliverable:	July 31 2009
Actual submission date:	July 26 2009
Lead partner:	BHAM
Revision:	final
Dissemination level:	PU

Goal-directed behaviour is a defining characteristic of intelligent behaviour. In CogX we are committed to using planning (coupled with suitable reactive mechanisms) to determine how goals are achieved (see WP4), but how can an intelligent system create and manage its own goals? This document presents our work on the problem of *motive management*, where "motive" is an umbrella term for representations that dispose a system towards achieving future states. Our work is presented in the context of both the state-of-the-art and previous work by the consortium, and relates the problem of motive management to both our scenario-driven integration work, and the aims of the project as a whole.

Executive Summary

This report presents research carried out in WP1 on architectural designs for intelligent robots in general, and on mechanisms and representations for generating and managing a robot's goals in particular. The architectural work has distilled previously developed principles into a high-level system design called **PECAS**. This name refers to the architecture's foundations in the CoSy Architecture Schema (**CAS**), and its initial application in the **P**layMate and **E**xplorer scenarios of that recently completed project. Whilst PECAS provides coarse guidelines for system design, it currently provides limited guidance about how goals for a system should be generated, and how multiple, possibly conflicting goals should be managed. This problem area is the subject of Task 1.2 (Architectures for desire generation and management), with PECAS providing the architectural context within which the work has been performed. To date, the work done for Task 1.2 has been to analyse the requirements on motive management systems for intelligent robots, survey how these requirements are addressed by existing work in various sub-fields of AI and robotics, and use the knowledge gained from these exercises to synthesize an outline design for a novel architecture for motive management. In this context we use the term "motive" as an umbrella term encompassing goals, drives and other dispositional states which might cause an intelligent system to act (states we address separately in the annex described in §2.3). Managing motives means choosing what to do next in a way that is informed both by the possibility of action and by the desirability of the outcome. In a CogX robot, the motive management system will ultimately be responsible for choosing whether to self-extend (based on self-understanding), and which particular route of extension to follow (from possibilities suggested by other systems)[1]. It will also manage the non-reflective tasks the robot will tackle, and the trade-offs inherent in choosing between different tasks and different opportunities for self-extension.

Role of motive management in CogX

In CogX we are interested in designing, building and understanding artifacts which are capable of a *wide range* of intelligent behaviours, rather than a single type of narrow intelligence. In general, when we refer to intelligent behaviour we are, either implicitly or explicitly, referring to behaviour that is *goal-directed*, i.e. behaviour that serves to achieve a pre-identified (whether at run-time or design-time) state [5]. Even a cursory analysis of our integration scenarios demonstrate that it is possible for a system to have multiple goals at the same time. Moreover it is possible for a system to have

[1] This does not rule out the possibility of our systems having forms of non-planned, i.e. reactive, self-extension.

multiple *conflicting* goals, i.e. goals which cannot be achieved through a single course of action (in the immediate future at least). A hypothetical CogX artifact can have two distinct classes of goals: task goals (fetching and finding things, answering questions, generally being useful to humans etc.) and self-extension goals (learning new things, filling gaps in knowledge etc.). Not only can goals conflict within these classes (should the robot perform the new task requested by the last person it talked to, or finish the task it is half-way through; should the robot learn what a particular object looks like or what names humans typically use for it), but they can also conflict across these classes. Mechanisms able to resolve this latter type of conflict, the conflict between a goal to perform a task and a goal to self-extend, will be central to the unified theory developed by the consortium. In order to investigate instances of these conflicts we must first build an architectural framework within which goals, and conflicts, can be represented and processed. This report contributes towards the design of this framework, and thus the broader aims of the project.

We can place the work in a more concrete context by considering an example from the "Dora" integration scenario. This scenario features a mobile robot (called Dora) which is able to learn about the spatial properties of its environment, where this learning can be partly triggered by an understanding of what it knows it doesn't know (a limited form of self-extension through introspection). The learning is done to provide knowledge to help Dora perform tasks for its human controllers (a precursor of the hypothetical gopher robot described in the project technical annex). At the start of the scenario Dora is given a tour of an office environment. The human tour-guide labels particular rooms of interest (e.g. "we are now in the kitchen", "over there is the robot lab") and indicates objects that might be relevant to future tasks (e.g. "that is the cooker", "the cornflakes are in the top cupboard"). Given that the human's indications are only approximate, and Dora's sensors are noisy, at the end of the tour Dora has a number of gaps in its understanding of its environment. These will all give rise to self-extension goals; goals which can be achieved through knowledge gathering and information-processing activities. How these goals arise in an architecture, and how particular goals are selected for further expansion (planning and execution) are questions which we must address in this WP and in this report. If Dora is asked to perform a task such as bringing the human some cornflakes, how does it choose whether it should achieve this goal (the goal of the human having the cornflakes) rather than filling the gaps in its own knowledge? What if the gaps in its knowledge are directly related to the location of the cornflakes box or its ability to manipulate it? This indicates not only the need for a study of motive management in CogX, but also the need for close integration between WP1 and the workpackage on the planning of action, sensing and learning (WP4).

Contribution to the CogX scenarios and prototypes

As the above scenario sketch demonstrates, the work on motive management will be central to the integration of different capabilities in a single system. In this first year our demonstrator systems will only have a small range of different capabilities, thus reducing the need for management mechanisms. However, we will employ the same prototype motive management architecture in both the Dora and George scenarios to arbitrate between the various goals they will have. The exact requirements on the software systems developed and deployed will depend on the progress made in the other WPs and during the integration process. Our current aim is for at least Dora to feature both task and self-extension goals, and for the motive management architecture to control the selection and achievement of these. Although the need for motive management may be limited in this first year, it is essential that it is included (at least in prototype form) in our integrated systems at this early stage; other, dependent, behaviours should be developed within the managed framework (i.e. the goals and behaviours of the system should be deliberatively selected and planned) so that they can be more easily integrated with other behaviours when the capabilities of our systems are extended.

1 Tasks, objectives, results

1.1 Planned work

Work reported in this deliverable concerns Task 1.2:

> *Task 1.2: Develop representations for motives, and an architecture to generate them and contain processes that can manage them.*[2]

The plan to address this task was to review both the requirements on such a system in CogX and the current state-of-the-art techniques related to these requirements. The results of these studies were to be synthesised into a design for an extension to the PECAS architectural schema, which would then be implemented and tested in an integrated system.

1.2 Actual work performed

To date we have completed the survey elements of the planned work, and proposed an initial design for a motive management framework which goes beyond the state-of-the-art in a number of ways. The results of this process are presented in the article attached as an annex and described in §2.3. In summary, we started the requirements analysis by looking at a scenario involving a hypothetical robot helper in a near-future family home. By inspecting this, along with the in-depth analysis of motivation systems provided previously by Beaudoin et al. [1, 13], we identified the problems of *encoding drives* (how the needs of the system are represented), *goal generation* (how particular goals are generated from the drives with reference to the current state), and *goal selection* (how the system determines which goals to act on) as important aspects of any motive management framework. We then reviewed the existing literature on intelligent systems which manage their own goals to explore previous approaches to these aspects of the problem. The review encompassed belief-desire-intention (BDI) approaches (e.g. [2, 6]), reactive and behaviour-based systems (e.g. [9, 3]), reactive planners with goal management extensions (which represents perhaps the largest body of work on this subject) (e.g. [7, 11, 12]), and also reactive-deliberative hybrid systems and related planning literature (e.g. [4, 10]). The review article groups the previous work into categories based on the approaches they take to the three problems listed above. From an analysis of strengths and weaknesses of these categories we synthesised an early design sketch for a motive management framework. This includes reactive goal generators and alarms, homeostatic drives modelled as resource constraints in a planning domain (as this overcomes weaknesses in reactive goal generation

[2] In the technical annex we used the term "desire instance" not "motive". Our work since writing the annex has driven us to change terminology.

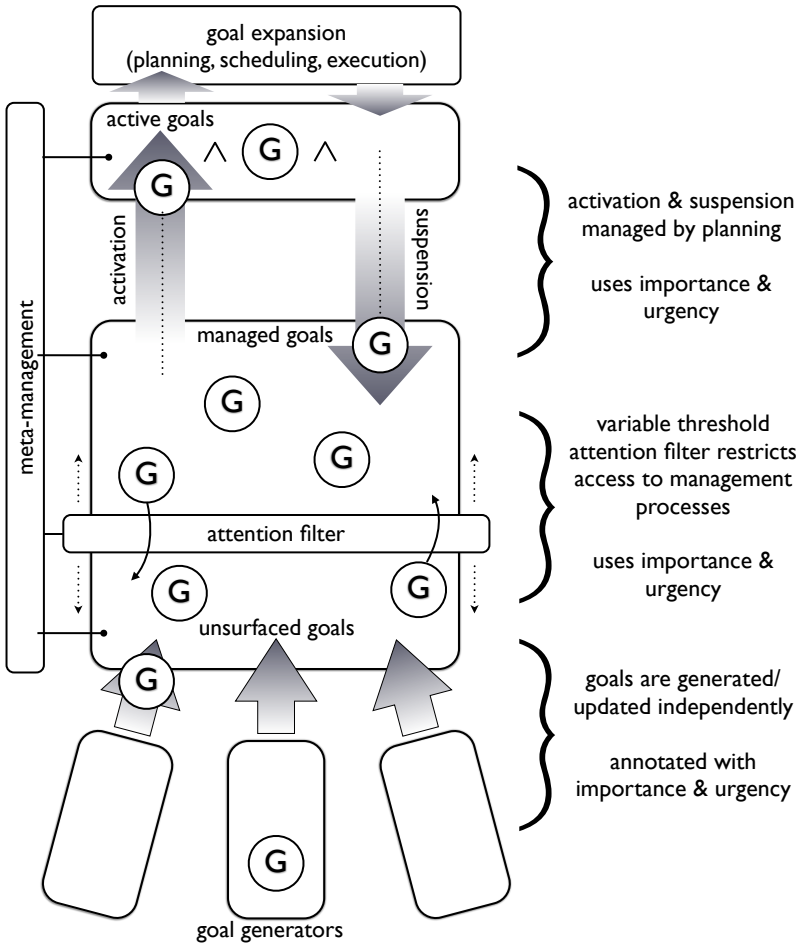


Figure 1: A visualisation of the proposed design for a motive management framework.

for some problems) and a goal management process which includes *active* and *suspended* goals plus an *attention filter* which restricts the management process's inputs based on resource constraints. Central to this management process will be a continual planning architecture able to process resources (as is being developed in WP4), and deal with oversubscribed goal states. Oversubscription planning allows the deliberative system to reason about which goals it can achieve given costs (which can be derived from resource usage, or from importance and urgency values provided by goal generators), rather than have goal selection as an external process to planning. The proposed design is pictured in Figure 1.

We are just beginning the development of early prototypes based on this design sketch. These prototypes will be integrated into the PECAS archi-

tecture (presented in the annex described in §2.1), which currently has only limited, ad-hoc goal generation and management facilities. We will evaluate the resulting systems in the Dora and George integration scenarios, and also in a simulated domain (allowing for greater variety of test conditions).

1.3 Relation to the state-of-the-art

Motive management has only been tackled explicitly by handful of researchers, although many systems have been designed and built which solve small parts of the range of problems involved. Part of the reason for this lack of research is due to difficulties of producing an intelligent system capable of demonstrating even a narrow range of goal-directed behaviours: there has just been no need for researchers to worry about how to arbitrate between different behaviours as their systems can typically only do one of a small, closely-related, range of things. However the recent advances in both component intelligence (i.e. single behaviours) and integrated intelligent systems (partly as a result of the EU Cognitive Systems programme) has brought AI and robotics to the point where these issues must now be considered. As such, the work planned and performed in this WP is timely and important, particularly given the dearth of related literature on motive management in robotics.

It is too soon to relate our design and implementation work to the state of the art (as it is still at the early prototype stage). However, based on our survey, our design is for a system that will go beyond current systems in both the range of drives and goals it can handle, and the power of the mechanisms it uses to manage them. When the motive management framework is fully developed we shall relate it to some of the approaches identified in the attached annex: the MADbot motivated planning system [4]; the Goal and Resource Using architecturE (GRUE) [7]; and systems from BDI community (e.g. [6]) including goal-management extensions to existing architectures (e.g. the BDI abstraction layer recently added to Soar [8]);

2 Annexes

2.1 Hawes et al. Planning as an Architectural Control Mechanism (HRI'09)

Bibliography Nick Hawes, Michael Brenner and Kristoffer Sjöö. Planning as an Architectural Control Mechanism. In *Proceedings of the 4th ACM/IEEE international conference on Human robot interaction (HRI '09)*, pp. 229-230. La Jolla, California, USA. 2009.

Abstract We describe recent work on PECAS, an architecture for intelligent robotics that supports multi-modal interaction.[3]

Relation to WP This short paper describes the architectural foundations upon which some of the theoretical and practical contributions of CogX will initially build (particularly those in the context of the integrated systems). Whilst not related directly to motive management, this paper can be considered a very brief primer on the shared history of the consortium's architectural work. In the context of WP1, the work on PECAS defines the context within which we will place the representations and mechanisms under investigation (i.e. representations of beliefs, and architectures for motive management). This paper was significantly expanded to produce the paper described in §2.2. For readers with limited time we suggest reading the short version and skipping the expanded version. For readers with a greater interest in architectural issues, we suggest skipping the shorter version and only reading the expanded version.

2.2 Hawes et al. Planning and Acting with an Integrated Sense of Space (HYACS@IJCAI'09)

Bibliography Nick Hawes, Hendrik Zender, Kristoffer Sjöö, Michael Brenner, Geert-Jan M. Kruijff and Patric Jensfelt. Planning as an Architectural Control Mechanism. In *International Workshop on Hybrid Control of Autonomous Systems (HYCAS) at IJCAI '09.* Pasadena, California, USA. 2009.

Abstract The paper describes PECAS, an architecture for intelligent systems, and its application in the Explorer, an interactive mobile robot. PECAS is a new architectural combination of information fusion and continual planning. PECAS plans, integrates and monitors the asynchronous flow of information between multiple concurrent systems. Information fusion provides a suitable intermediary to robustly couple the various reactive and deliberative forms of processing used concurrently in the Explorer. The Explorer instantiates PECAS around a hybrid spatial model combining SLAM, visual search, and conceptual inference. This paper describes the elements of this model, and demonstrates on an implemented scenario how PECAS provides means for flexible control.

Relation to WP As the expanded version of the paper presented in §2.2, this paper has much the same relation to the WP. This paper contains a more detailed exposition of the design of the PECAS architecture and the

[3]This was a short paper accepted as late-breaking abstract. The 2-page paper limit resulted in this reduced abstract.

consequences of this design. The paper includes an example from a mobile robot system which is able to find an object for a human instructor.

2.3 Hawes. A Survey of Motivation Systems for Intelligent Robots (report)

Bibliography Draft article. Intended for submission the Elsevier's journal Artificial Intelligence.

Abstract The ability to achieve one's goals is a defining characteristic of intelligent behaviour. A great many existing theories, systems and research programmes address the problems associated with generating behaviour to achieve a goal; much fewer address the related problems of how and why goals should be generated in an intelligent artifact, and how a subset of all possible goals are selected as the focus of behaviour. It is research into these problems of *motivation*, which this article aims to stimulate. Building from the analysis of scenario involving a futuristic household robot, we extend an existing account of motivation in intelligent systems to provide a framework for surveying relevant literature in AI and robotics. This framework guides us to look at the problems of *encoding drives* (how the needs of the system are represented), *goal generation* (how particular instances of goals are generated from the drives with reference to the current state), and *goal selection* (how the system determines which goal instances to act on). After surveying a variety of existing approaches in these terms, we build on the results of the survey to sketch a design for a new motive management framework which goes beyond the current state of the art.

Relation to WP This article presents the results of the requirements analysis, literature survey and early design work described in §1.2. If short of time, Section 5 "Summary of Approaches" can be read on its own to provide an overview of the results of the survey, and Section 6 "A Design for a Motive Management Framework" can be read for more information on the proposed framework.

References

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

[2] M. E. Bratman. *Intention, Plans, and Practical Reason*. Harvard University Press, Cambridge, MA, 1987.

[3] Joanna J. Bryson. The Behavior-Oriented Design of modular agent intelligence. In R. Kowalszyk, Jörg P. Müller, H. Tianfield, and R. Un-

land, editors, *Agent Technologies, Infrastructures, Tools, and Applications for e-Services*, pages 61–76. Springer, Berlin, 2003.

[4] A. M. Coddington. Integrating motivations with planning. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS07)*, pages 850–852, 2007.

[5] Stan Franklin and Art Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *ECAI '96: Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages*, pages 21–35, London, UK, 1997. Springer-Verlag.

[6] Michael P. Georgeff and François Felix Ingrand. Decision-making in an embedded reasoning system. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pages 972–978, 1989.

[7] Elizabeth Gordon and Brian Logan. Managing goals and resources in dynamic environments. In Darryl N. Davis, editor, *Visions of Mind: Architectures for Cognition and Affect*, chapter 11, pages 225–253. Idea Group, 2005.

[8] Sean A. Lisse, Robert E. Wray, and Marcus J. Huber. Beyond the ad-hoc and the impractically formal: Lessons from the implementation of formalisms of intention. In *Working Notes of the AAAI Spring Symposium on Intentions in Intelligent Agents*, March 2007.

[9] Pattie Maes. The agent network architecture (ana). *SIGART Bull.*, 2(4):115–120, 1991.

[10] Romeo Sanchez Nigenda Menkes van den Briel and Subbarao Kambhampati. Over-subscription in planning: A partial satisfaction problem. In *ICAPS 2004 Workshop on Integrating Planning into Scheduling*, 2004.

[11] F. Michaud, C. Côté, D. Létourneau, Y. Brosseau, J. M. Valin, É. Beaudry, C. Raïevsky, A. Ponchon, P. Moisan, P. Lepage, Y. Morin, F. Gagnon, P. Giguère, M. A. Roux, S. Caron, P. Frenette, and F. Kabanza. Spartacus attending the 2005 aaai conference. *Auton. Robots*, 22(4):369–383, 2007.

[12] Matthias Scheutz and Paul Schermerhorn. Affective goal and task selection for social robots. In Jordi Vallverdú and David Casacuberta, editors, *The Handbook of Research on Synthetic Emotions and Sociable Robotics*. Information Science Reference, 2009.

[13] Ian Wright. *Emotional Agents*. PhD thesis, School of Computer Science, The University of Birmingham, 1997.

Planning as an Architectural Control Mechanism

Nick Hawes
School of Computer Science
University of Birmingham, UK
n.a.hawes@cs.bham.ac.uk

Michael Brenner
Institut für Informatik
Albert-Ludwigs-Universität
Freiburg, Germany
brenner@informatik.uni-freiburg.de

Kristoffer Sjöö
Centre for Autonomous Systems
Royal Institute of Technology (KTH), Stockholm, Sweden
krsj@csc.kth.se

ABSTRACT

We describe recent work on PECAS, an architecture for intelligent robotics that supports multi-modal interaction.

Categories and Subject Descriptors
I.2.8 [Computing Methodologies]: ARTIFICIAL INTELLIGENCE: *Problem Solving, Control Methods, and Search*

General Terms: Design, Theory

Keywords: architecture, planning, robotics, integration

1. INTRODUCTION

An information-processing (IP) architecture designed to enable autonomous robots to interact intelligently with humans in multiple contexts must solve a number of problems. One of these is to unify the processing of multiple, concurrently active, heterogeneous subsystems into a single stream of intelligent behaviour. To do this the architecture must mediate between both the different *representations* used throughout the system and the *processes* in its multiple subsystems. Over the last couple of years we have been exploring designs for IP architectures for intelligent robots. In this work we have already addressed mediation between representations [1, 3]. In this paper we present recent work on the problem of mediating between processes.

2. BACKGROUND AND MOTIVATION

The following summarises some of the assumptions that underlie our work. We have been developing the **PECAS** architecture to fulfill the requirements of scenarios featuring situated dialogue coupled with table-top manipulation (our **P**layMate scenario) or semantic mapping (our **E**xplorer scenario). Our architecture is based on the CoSy Architecture Schema (**CAS**), which structures systems into *subarchitectures* (SAs) which cluster *processing components* around *working memories* [2]. From this schema we have created a number of SAs (vision, communication, navigation, manipulation etc.) which can be selectively grouped into a single architecture for a particular scenario. All these SAs are active in parallel, and all operate on SA-specific representations (as is necessary for robust and efficient task-specific processing). These disparate representations are unified by a *binding SA*, which performs abstraction and cross-modal information fusion on the information from the other SAs [3]. This gives us a way of mediating between heterogeneous content in our systems, but it does not say anything about how we can use this content in the generation of behaviour.

We have built a number of systems using PECAS, including interactive robots for table-top manipulation and for semantic mapping of indoor environments. These robots have multiple capabilities that can be used to perform many different user-specified tasks. In order to provide the robots with a generic and extensible way to deal with such tasks, we treat the computation and coordination of overall system behaviour as a *planning* problem. The use of planning gives the robot a high degree of autonomy: complex goal-driven behaviours need not be hard-coded into the system, but are planned by the robot itself. Likewise, the robot can autonomously adapt its plans to changing situations using *continual planning* and is therefore well suited to dynamic environments. However, relying on automated planning means that all tasks for the robot need to be posed as goals for a planner, and all behaviour to achieve these goals must be encoded as actions that the planner can process.

3. BEHAVIOUR AS PLANNING

Planning systems are given goals as logical formulae. In our architecture, such goals are typically generated when intentional content is processed by the communication SA (i. e. given by a human in natural language), then made available to the motivation SA via binding. For example, if the user wants the robot to bring them a certain book, this might lead the robot to form a goal such as `(holds human1 book1)`. Note that while goals are most commonly provided by a human, they can also arise from internal processes.

While the traditional use of planning is achieving goals in the physical world using physical actions, such direct interpretations of behaviour are the exception rather than the rule in human-robot interaction. Here, where information is incomplete, uncertain, and distributed over several agents and throughout subsystems, much of the actions to be performed by the system are to do with processing *information*. Whilst some IP may be performed continuously by the system (e. g. listening for sounds to recognise, SLAM) much IP is too costly to be performed routinely and should instead be performed only when relevant to the task at hand, i. e. it should be planned based on context.

4. PLANNING FOR IP

Underlying our approach to IP is the functionally decomposed, concurrently active, structure of PECAS. As each SA is effectively a self-contained processing unit, our design

HRI'09, March 11–13, 2009, La Jolla, California, USA.
ACM 978-1-60558-404-1/09/03.

leads naturally to an integration strategy: each SA is treated as a separate agent in a multi-agent planning problem. Although this separation has many features which we do not have space to discuss, a crucial one is that each SA's knowledge is separate within the planning state, and can only be reasoned about using epistemic operators (e. g. `(K vision.sa colour(obj1))`, meaning that the vision SA knows the colour of an object). Likewise, goals are often epistemic in nature, e. g. when a human or a SA wants to query the vision SA for the colour of an object.

To realise internal and external information exchange each SA can use two special actions: *tell-value* and *ask-value*. These provide and request information, respectively, and have epistemic effects. Interaction with humans or other external agents can also use (but is not limited to) these actions. As a result, planning of IP becomes a matter of planning for epistemic goals in a multiagent system. For example, if a human teacher tells our robot that "the ball is blue", this gives rise to the motivation that all SAs dealing with colours (e. g. vision) should know the colour of the ball in question. This may lead to a plan in which the communication SA uses a tell-value action to give the vision SA this information. Note that the factual information provided by the teacher is not directly entered into the robot's knowledge base. Instead it gives rise to a more complex motivation which enables the planner to initate more complex IP as necessary, e.g. triggering a colour learning process.

This design gives the robot more autonomy in deciding on the task-specific information flow through its subsystems. But there is also another assumption underlying this design: whilst the binding SA is used to share information throughout the architecture, not all information in the system can or should be shared this way. Some information is unavailable because it is modality specific, and even cross-modal knowledge is often irrelevant to the task at hand. If all information was shared this would overwhelm the system with (currently) irrelevant information (e.g. lists of all the people, rooms, objects, object categories etc. that parts of the system know about). Thus, in order to restrict the knowledge the planner gives "attention" to without losing important information, it needs to be able to *extend* its planning state on-the-fly, i. e. during the continual planning process. We call this process *task-driven state generation*.

To support this the planner makes use of meta-level information, so-called *produce* and *consume* facts. They describe which SAs can produce which predicates (i. e. where certain types of information can come from) and which SAs can consume which predicates (i. e. where certain types of information should go). This enables more general formalisations for, e. g., teaching goals (all SAs which consume a particular predicate should be told the value of any new instances of that predicate), and requesting information (if a SA needs the value of a state variable, then it should ask a SA that can produce it). Produce and consume facts provide the planner with enough information to use tell-value and ask-value in its IP planning. We assume that the SA-specific details for asking and telling can be left opaque to the planner and will be filled in at execution time by the executing SA. It is not obvious whether this assumption will be valid in all cases, but it provides a useful starting point.

5. EXAMPLES

In this section we will provide examples of our approach in action. In our mobile robotic scenario, where a robot interactively explores an office environment and runs simple errands for human users, task-driven state generation is used in several ways to help the robot deal with its necessarily incomplete knowledge. When the robot is given a command such as "Bring me the Borland book" the planner realises that in order to achieve this task it first needs to satisfy the epistemic subgoal of knowing where the book is. Thus, in the initial phases of the continual planning process, it will query SAs who (as specified by appropriate *produce* facts) can provide information about the location of the book. In this case, it is the conceptual mapping SA which can provide default knowledge about the locations of objects, e. g. the library in the case of books. Having updated the state with the information from conceptual mapping, more detailed planning becomes possible, allowing the robot to plan to move to the library to search for the book.

Essentially the same process is used for planning human-robot interaction. If the agent who is believed to be able to "produce" facts about the book location is not an internal SA but a human, the robot plans to *ask-value* the human. However, the preconditions for ask-value may vary for different addressees. In particular, external agents must first be approached and engaged in a conversation. Since the same planning approach is used for both physical actions as well as internal IP, the planner can directly initiate a situated dialogue including the physical movement of the robot.

In our table-top scenario the robot is capable of learning and recognising visual features such as colour and shape. If the confidence of a recognition result falls within a particular window of uncertainty (e.g likely but not certain), the robot can generate clarification behaviour. Clarification is represented as a goal in which the requesting SA should know the value of a particular predicate (e.g. the colour of an object). The plan created for this goal consists of the requesting SA (e.g vision) asking SAs which produce this predicate for its value (e.g. communication). If the communication SA is asked, this may result in the robot asking a nearby human for the information, and if an answer is provided to the robot, the information is made available via the binder. This highlights how our design allows the planner to operate without knowing the details of how each SA implements its responses to ask-value and tell-value actions.

6. CONCLUSION

We have presented work on using planning to allow a robot to coordinate multiple processes in PECAS, an architecture for intelligent robots. Our approach has been applied to different HRI scenarios, demonstrating its generality.

Acknowledgments

Supported by the EU integrated projects CoSy and CogX.

7. REFERENCES

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

[2] N. Hawes, A. Sloman, J. Wyatt, M. Zillich, H. Jacobsson, G.-J. Kruijff, M. Brenner, G. Berginc, and D. Skočaj. Towards an integrated robot with multiple cognitive functions. In *AAAI '07*, 2007.

[3] H. Jacobsson, N. Hawes, G.-J. Kruijff, and J. Wyatt. Crossmodal content binding in information-processing architectures. In *HRI '08*, 2008.

Planning and Acting with an Integrated Sense of Space

N. Hawes[1] and **H. Zender**[2] and **K. Sjöö**[3] and **M. Brenner**[4] and **G.J.M. Kruijff**[2] and **P. Jensfelt**[3]

[1]Intelligent Robotics Lab, School of Computer Science, University of Birmingham, UK

[2]Language Technology Lab, German Research Center for Artificial Intelligence (DFKI), Saarbrücken, Germany

[3]Centre for Autonomous Systems, Royal Institute of Technology (KTH), Stockholm, Sweden

[4]Institute for Computer Science, Albert-Ludwigs-Universität, Freiburg, Germany

[1]nah@cs.bham.ac.uk, [2]{zender, gj}@dfki.de, [3]{krsj,patric}@csc.kth.se, [4]brenner@informatik.uni-freiburg.de

Abstract

The paper describes PECAS, an architecture for intelligent systems, and its application in the Explorer, an interactive mobile robot. PECAS is a new architectural combination of information fusion and continual planning. PECAS plans, integrates and monitors the asynchronous flow of information between multiple concurrent systems. Information fusion provides a suitable intermediary to robustly couple the various reactive and deliberative forms of processing used concurrently in the Explorer. The Explorer instantiates PECAS around a hybrid spatial model combining SLAM, visual search, and conceptual inference. This paper describes the elements of this model, and demonstrates on an implemented scenario how PECAS provides means for flexible control.

1 Introduction

Recently there has been an enormous increase in R&D for domestic robot assistants. Moving beyond the Roomba, more complex robot "gophers" are envisioned, to assist in performing more demanding tasks in human environments. To achieve this vision, the study of integrated robotic systems that fulfill many different requirements is necessary.

Research on individual aspects of such systems has yielded impressive robots, e.g. the museum guides Rhino [Burgard *et al.*, 2000] and Robox [Siegwart and et al., 2003], or the in-store assistant ShopBot [Gross *et al.*, 2008]. Other robots, like RoboVie [Ishiguro *et al.*, 2001], Mel [Sidner *et al.*, 2004], BIRON [Peltason *et al.*, 2009], or our CoSy systems [Hawes *et al.*, 2007; Kruijff *et al.*, 2007] provide capabilities for the robot to interact with a human using spoken dialogue. As impressive as they are, all these systems lack the wide range of capabilities needed by a versatile robotic assistant. Producing such a system by integrating the results of specialized subfields such as control, perception, reasoning, and dialogue remains a major challenge to AI and robotics.

If we wish to build a mobile robotic system that is able to act in a real environment and interact with human users we must overcome several challenges. From a system perspective, one of the major challenges lies in producing a single intelligent system from a combination of heterogeneous specialized modules, e.g. vision, natural language processing, hardware control etc. Ideally this must be done in a general-purpose, extensible and flexible way, with the absolute minimum of hardwired behaviors. This both allows solutions to be reused in different systems (allowing an understanding of the design trade-offs to be obtained), and for the same system to be altered over time as requirements change. Additionally, taking account of the "human in the loop" poses the challenge of relating robot-centric representations to human-centric conceptualizations, such as the understanding of large-scale space [Kuipers, 1977].

In this paper we present PECAS (see Section 2), our novel approach to integrating multiple competences into a single robotic system. PECAS allows us to address many of the previously described problems in an architectural way, providing an approach that is ultimately resuable in other robots and domains. For a general-purpose architecture to be deployed it must be *instantiated* with task-specific content. Section 3 presents the Explorer system, our instantiation of PECAS in an interactive mobile robot. Following this we use the Explorer instantiation to present examples of PECAS as a control system (in a general sense). Section 4 presents a complete system run from our implementation, demonstrating how the flow of information and control passes between low and high levels in our system. Section 5 discusses control in PECAS in general, and the strengths and weaknesses of our approach.

2 The PECAS Architecture

Our recent work on intelligent robotics has led to the development of the PlayMate/Explorer CoSy Architecture Sub-Schema (**PECAS**). PECAS is an information-processing architecture suitable for situated intelligent behavior [Hawes *et al.*, 2009]. The architecture is designed to meet the requirements of scenarios featuring situated dialogue coupled with table-top manipulation (the **P**layMate focus [Hawes *et al.*, 2007]) or mobility in large-scale space (the **E**xplorer focus [Zender *et al.*, 2008]). It is based on the CoSy Architecture Schema (**CAS**), which structures systems into *subarchitectures* (SAs) which cluster *processing components* around *shared working memories* [Hawes *et al.*, 2007]. In PECAS, SAs group components by function (e.g., vision, communication, or navigation). All these SAs are active in parallel, typically combining reactive and deliberative forms of processing, and all operating on SA-specific representations (as

is necessary for robust and efficient task-specific processing). These disparate representations are unified, or *bound*, by a *subarchitecture for binding* (binding SA), which performs abstraction and cross-modal information fusion on the information from the other SAs [Jacobsson *et al.*, 2008]. PECAS makes it possible to use the multiple capabilities provided by a system's SAs to perform many different user-specified tasks. In order to give the robots a generic and extensible way to deal with such tasks, we treat the computation and coordination of overall (intentional) system behavior as a *planning* problem. The use of planning gives the robot a high degree of autonomy: complex goal-driven behaviors need not be hard-coded, but can be flexibly planned and executed by the robot at run-time. The robot can autonomously adapt its plans to changing situations using *continual planning* and is therefore well suited to dynamic environments. Relying on automated planning means that tasks for the robot need to be posed as goals for a planner, and behavior to achieve these goals must be encoded as actions that the planner can process. The following sections expand upon these ideas.

2.1 Cross-Modal Binding

Cross-modal binding is an essential process in information-processing architectures which allow multiple task-specialized (i.e., *modal*) representations to exist in parallel. Although many behaviors can be supported within individual modalities, two cases require representations to be shared across the system via binding. First, the system requires a single, unified view of its knowledge in order to plan a behavior that involves more than one modality (e.g., following a command to do something relative to the object or area). Second, binding is required when a subsystem needs information from another one to help it solve a problem (e.g., using visual scene information to guide speech recognition [Lison and Kruijff, 2008]).

Our approach to binding underlies much of the design and implementation of our systems, and so we will reiterate it here (for more details see [Jacobsson *et al.*, 2008]). Each PECAS SA that wishes to contribute information to the shared knowledge of the system must implement a *binding monitor*. This is a specialized processing component which is able to translate from an arbitrary modal representation (e.g., one used for spatial modeling or language processing) into a fixed *amodal* (i.e., behavior neutral) representation. Across a PECAS system the binding monitors provide a parallel abstraction process mapping from multiple, different representations to a single, predicate logic-like representation. Binding monitors deliver their abstracted representations into the binding SA as binding *proxies* and *features*. Features describe the actual abstract content (e.g., color, category, or location) in our amodal language, whilst proxies group multiple features into a single description for a piece of content (such as an object, room, or person), or for relationships between two or more pieces of content. The binding SA collects proxies and then attempts to fuse them into binding *unions*, structures which group multiples proxies into a single, cross-system representation of the same thing. Groupings are determined by feature matching. Figure 1 illustrates this: the SA for navigation (nav SA) and the SA for conceptual mapping and reasoning (coma SA),

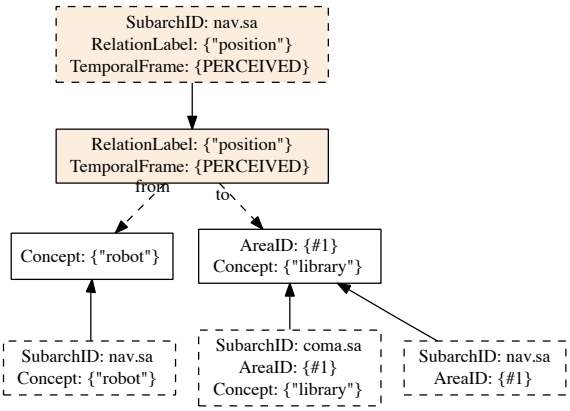


Figure 1: Binding localization and conceptual information: "the robot is in the library." Proxies have dashed borders, unions solid borders. Relation proxies, -unions are colored.

provide their information to the binding SA. Throughout this process links are maintained between all levels of this hierarchy: from modal content, to features and proxies, and then on to unions. These links act like pointers in a programming language, facilitating access to information content regardless of location. Binding thus supports the two identified cases for cross-modal binding: the collection of unions provide a single unified view of system knowledge, and cross-subsystem information exchange is facilitated by linking similarly referring proxies into single union.

2.2 Planning for Action and Processing

For PECAS we assume that we can treat the computation and coordination of overall system behavior as a planning problem. This places the following requirements on PECAS: it must be able to generate a state description to plan with; system-global tasks for the robot need to be posed as goals for a planner; and behavior to achieve these goals must be encoded as actions which can be processed by the planner. In our implementation we use the MAPSIM continual planner and its Multi-Agent Planning Language (MAPL) [Brenner and Nebel, 2009]. In MAPL, we can model beliefs and mutual beliefs of agents as well as operators affecting these, i.e., perceptual and communicative actions. The continual planner actively switches between planning, execution, and monitoring in order to gather missing goal-relevant information as early as possible.

To provide a planning state, the planning SA automatically translates from the unions in the binding SA into MAPL. The planner thus automatically receives a unified view of the system's current knowledge. As we maintain links from unions back to modal content, our planning state, and therefore our plans, remain grounded in representations close to sensors and effectors. In PECAS, planning goals arise as modal *intentional content* which is then abstracted via binding monitors and placed in the planning SA's working memory. From here we use the same translation method as is used on the planning state to produce MAPL goals for the planner.

While the traditional use of planning is achieving goals in the world using physical actions, such direct interpretations of behavior are the exception rather than the rule in cognitive robotics (cf. [Shanahan, 2002]). Here, where infor-

mation is incomplete, uncertain, and distributed throughout subsystems, much of the actions to be performed by the system are to do with processing or moving *information*. Whilst some information processing may be performed continually (e.g., SLAM), much of it is too costly to be performed routinely and should instead be performed only when relevant to the task at hand, i.e., it should be planned based on context.

Underlying our approach to information-processing is the functionally decomposed, concurrently active, structure of PECAS. As each SA is effectively a self-contained processing unit, our design leads naturally to an integration strategy: each SA is treated as a separate agent in a multi-agent planning problem. A crucial feature of this strategy is that each SA's knowledge is separate within the planning state, and can only be reasoned about using epistemic operators (i.e., operators concerned with knowledge). Likewise, goals are often epistemic in nature, e.g., when a human or a SA wants to query the navigation SA for the location of an object.

To realize internal and external information exchange each SA can use two epistemic actions, *tell-value* and *ask-value*, coupled with two facts about SAs, *produce* and *consume*. The actions provide and request information respectively. The facts describe which SAs can produce and consume which predicates (i.e., where certain types of information can come from and should go). For example, if a human teacher tells our robot that "this is the kitchen," this gives rise to the motivation that all SAs which consume room knowledge (e.g., coma SA described in the next section) should know the type of the room in question. This may lead to a plan in which the SA for situated dialogue (comsys SA) uses a tell-value action to give the coma SA this information.

Using this design, planning of information-processing becomes a matter of planning for epistemic goals in a multi-agent system. This gives the robot more autonomy in deciding on the task-specific information flow through its subsystems. But there is another assumption underlying this design: whilst the binding SA is used to share information throughout the architecture, not all information in the system can or should be shared this way. Some of it is unavailable because it is modality specific, and even cross-modal knowledge is often irrelevant to the task at hand. If all information was shared this would overwhelm the system with (currently) irrelevant information (e.g., lists of all the people, rooms, objects, object categories etc. that parts of the system know about). Thus, in order to restrict the knowledge the planner gives "attention" to without losing important information, it needs to be able to *extend* its planning state on-the-fly, i.e., during the continual planning process. In PECAS state extension can be done using ask-value and tell-value actions, and results in a process we call *task-driven state generation*.

3 The Explorer Instantiation

The binding and planning SAs described above are system and scenario independent. We now discuss the Explorer-specific SAs to describe concrete functionality and how this relates to system control. All SAs have been implemented in CAST (an open-source toolkit implementing the CAS schema) and tested on an ActivMedia PeopleBot. Figure 2 shows all SAs used in the Explorer PECAS instantiation. Most components used in the different SAs have been discussed in detail in earlier work (references provided below).

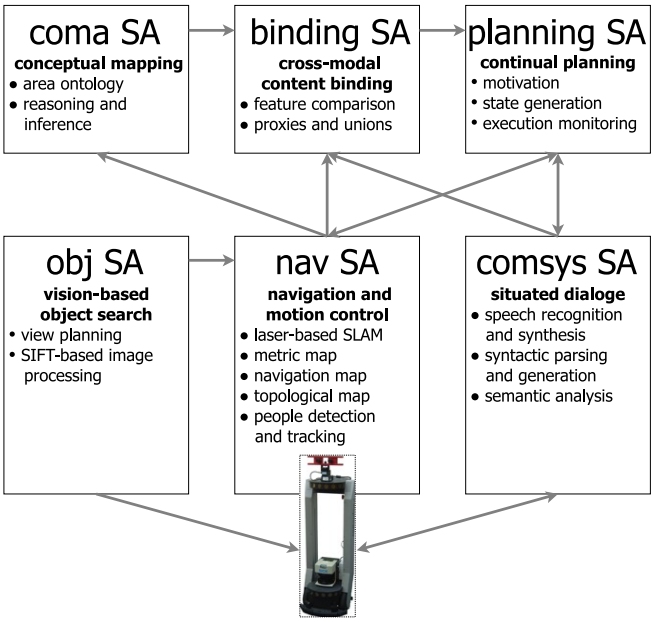


Figure 2: The Explorer Architecture

For a mobile robotic system that is supposed to act and interact in large-scale space, an appropriate spatial model is key. The Explorer maintains a multi-layered conceptual spatial map of its environment [Zender *et al.*, 2008]. It serves as a long-term spatial memory of large-scale space. Its individual layers represent large-scale space at different levels of abstraction, including low-level metric maps for robot motion control, a navigation graph and a topological abstraction used for high-level path planning, and a conceptual representation suitable for symbolic reasoning and situated dialogue with a human. In the Explorer, different SAs represent the individual map layers. For the details on human-augmented map acquisition see [Kruijff *et al.*, 2007].

nav SA The SA for navigation and spatial mapping hosts the three lowest levels of the spatial model (metric map, navigation map, and topological layer). For low-level, metric mapping and localization the nav SA contains a module for laser-based SLAM. The nodes and edges of the *navigation map* represent the connectivity of visited places, anchored in the metric map through x-y-coordinates. Topological areas, corresponding roughly to rooms in human terms, are sets of navigation nodes. This level of abstraction in turn feeds into the conceptual map layer that is part of the coma SA.

The nav SA contains a module for laser-based people detection and tracking [Zender *et al.*, 2007]. The nav SA binding monitor maintains the robot's current spatial position and all detected people, as proxies and relations on the binding SA. The smallest spatial units thus represented are areas. This provides the planner with a sufficiently stable and continuous description of the robot's state. The planning SA can pose *move* commands to the nav SA. The target location is defined based on the current task which might be to follow a person, move to a specific point in space, etc. Move commands are executed by a navigation control module, which

performs path planning on the level of the navigation graph, but automatically handles low-level obstacle avoidance and local motion control.

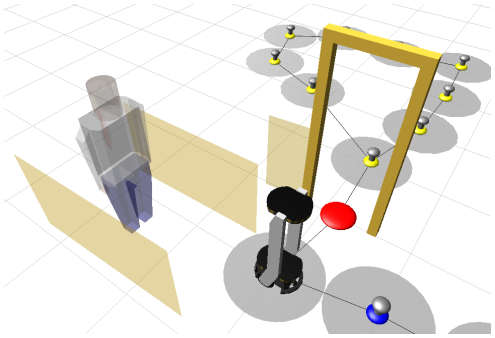
obj SA The SA for vision-based object search contains the components for finding objects using vision. It consists of a module for view planning and one for visual search. The view planning component creates a plan for which navigation nodes to visit, in what order and in what directions to look. Details of the process can be found in [Gálvez López *et al.*, 2008]. The visual search consists of SIFT feature matching directly on acquired images. Objects that are found are published on the obj SA working memory. The nav SA detects this and in turn extends the spatial model with the new objects. This then propagates the information to the coma SA and, if and when necessary, to the binding SA.

coma SA The SA for conceptual mapping and reasoning maintains an abstract symbolic representation of space suitable for situated action and interaction. It represents spatial areas (nav SA), objects in the environment (obj SA), and abstract properties of persons (e.g., ownership relations) in a combined A-Box and T-Box reasoning framework based on an OWL-DL reasoner, which can infer more specific concepts for the area instances [Zender *et al.*, 2008]. The coma SA makes its information available to the binding SA on demand, i.e., whenever planning SA sends an *ask-val* command to the coma SA, it will add its knowledge about spatial entities, especially their most specific concepts. In our system the explicit definitions of area concepts through occurrences of certain objects are also used to raise expectations about typical occurrences of certain objects. If the planning SA needs to know the location of an object that has not been encountered before, it can query the coma SA, which will then provide a *typical* location of the object in question. This is done via special T-Box queries involving the OWL-DL definitions of concepts. An example of this will be discussed in Section 4.

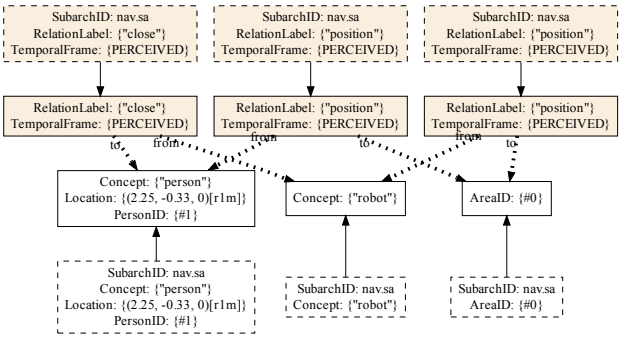
comsys SA The subarchitecture for situated dialogue processing has a number of components concerned with understanding and generation of natural language utterances [Kruijff *et al.*, 2009]. Speech recognition converts audio to possible text strings, which are subsequently parsed. Parsing produces a packed representation of logical forms (LFs) that correspond to possible semantic interpretations of an utterance. Finally, the semantics are interpreted against a model of the dialogue context. Content is connected to discourse referents, being objects and events talked about over the course of an interaction. In the dialogue context model, both the content of the utterance and its intent are modeled. All of this information is communicated to the planning SA and the binding SA through proxies representing the indexical and intentional content of the utterances. In rough terms the indexical content (information about entities in the world) is used by the binding SA to link with information from other modalities. Meanwhile the intentional content (information about the purpose of the utterance) is used by the planning SA to raise goals for activity elsewhere in the system [Kruijff *et al.*, 2009].

4 Example: Finding a book

This section presents a scenario in which a human asks the Explorer to perform a task. It shows how PECAS controls system behavior and information-processing. The example is taken directly from our implemented system, showing system visualizations (with minor post-processing).



(a) Screenshot of the visualization tool



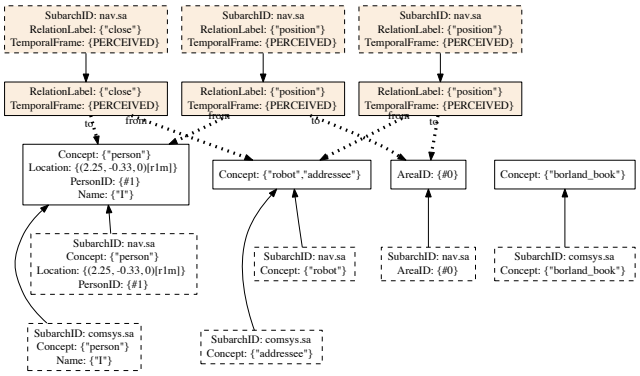
(b) Contents of binding working memory

Figure 3: Initial situation: the user approaches the robot

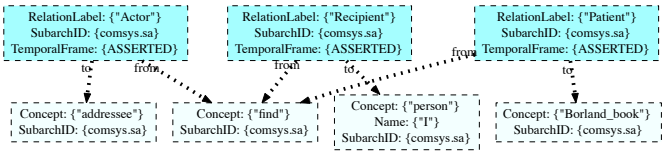
The system starts in the spatial context and binding state visualized in Figure 3: the robot and person are occupying the same area, and the person is close to the robot. The *robot proxy* is provided by the nav SA which it abstracts from its representation of the robot pose. The *person proxy* is provided by the nav SA because a person is being tracked. In addition to these, the nav SA makes available a proxy for the *area* in which one of these proxies occurs, linking them with a *position* relation proxy. Finally, the *close* relation proxy connects the robot proxy to the proxy of the person because the person is geometrically close to the robot. Note that no objects are present, nor are other areas except the current area.

Next, the human approaches the robot and says "find me the Borland book". The comsys SA interprets this utterance, presenting the elements of its interpretation to the rest of the system as proxies. Figure 4a shows the results. The Explorer itself (the recipient of the order) is represented by a proxy with Concept `addressee`, which binds to the robot proxy already present. The word "me" refers to the speaker, and generates a "person" proxy identified by the Name feature `I`. The expression referring to the book is given by a "Borland_book" proxy, not yet bound to any other proxies.

The comsys SA can determine the intention of this utterance, and separates the intentional elements of the interpretation from the aforementioned descriptive proxies. This intentional content is written to planning SA as a proxy structure with links back to the binder. The structure of this *motive* can be seen in Figure 4b. Planning SA, detecting a new mo-



(a) State of binding SA



(b) Representation of intentional content (as a *motive*)

```
Objects:                    Facts:
(area_id_0 - area-id)       (area-id gensym1 : area_id_0)
(gensym0 - robot)           (area-name area_id_0 : gensym1)
(gensym1 - area-name)       (perceived-pos gensym0 : area_id_0)
(gensym4 - person)          (perceived-pos gensym4 : area_id_0)
(gensym6 - movable)         (close gensym4 gensym0 : true)
```

(c) Planning state after processing the intentional content

Figure 4: State after the user has uttered the command "Find me the Borland Book."

tive, begins the process of creating a plan to fulfill it. First, it converts the information on the binder (Figure 4a) to the MAPL representation in Figure 4c. In this process unions become objects and predicates in the planning state. E.g., as the person union is related by a position relation union to an area union, this will be expressed to the planner as `(perceived-pos gensym4 : area_0)`, where `gensym4` is an auto-generated planning symbol referring to the person, and `area_0` refers to the area. The planner similarly converts the motive from Figure 4b into a MAPL goal `(K gensym4 (perceived-pos gensym6))`. This can be read as the Explorer having the goal of the the person knowing the position of the book. We use this interpretation of the command "Find me...", as the robot does not have the ability to grasp objects.

Given this state and goal, the planner creates a plan:

```
L1: (negotiate_plan gensym0 coma_sa)
L2: (tell_val_asserted-pos
        coma_sa gensym0 gensym6)
L3: (find_a gensym0 gensym6 gensym0)
L4: (tell_val_perceived-pos
        gensym0 gensym4 gensym6)
```

This plan states that the Explorer must find the location of the book (`L3`), then report this location to the person (`L4`). Before it does this it must negotiate with the coma SA (as each subarchitecture is treated as a separate agent) to provide a location where it might be able to find the book (`L1`,`L2`). The reasoning behind this plan is that Explorer must provide the person with a perceived location for the book (as is specified in the goal), and, having not seen it recently, the only way to obtain a perceived location is via its object search functionality. To perform an object search the system must have both an object to search for (the book in this case) and an area to search. *Typical* positions of objects (as opposed to their *perceived* positions) are stored in the ontology in coma SA. Rather than make all of this knowledge available via binding by default (a choice which would add many extra and redundant facts to the planning state), typical positions are offered by coma SA using a `produce` fact (see Section 2.2). This allows the planner to query coma SA for typical positions when it requires them. One advantage of this on-demand state generation is that the comsys SA could also be used to provide the same knowledge (and would be if the book was not found initially). In the above plan, the planner makes use of this by

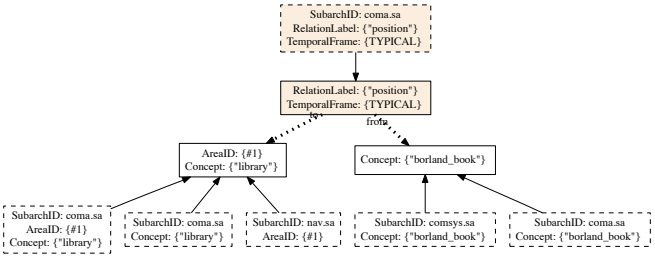


Figure 5: Hypothetical position of the Borland book.

getting the coma SA to `tell-val` the typical position of the Borland book to the binding SA. This yields the information that, as it is a book, it would typically be found in a library. Along with this, the coma SA also volunteers the specific information it has on libraries: an area exists in its map that is a library. This is illustrated in Figure 5.

Given this hypothesis for the book's location, MAPSIM uses a replanning step to expand the initial plan to include steps to move the robot to the library, search there for the book, then move back and report to the user. The updated plan and planning state are now as follows:

```
Objects:
(area_id_0 - area-id)   (area_id_1 - area-id)
(gensym0 - robot)       (gensym1 - area-name)
(gensym4 - person)   (gensym6 - borland_book)
(gensym6 - movable)     (gensym7 - area-name)

Facts:
(area-id gensym1 : area_id_0)
(area-id gensym6 : area_id_1)
(area-name area_id_0 : gensym1)
(area-name area_id_1 : gensym7)
(asserted-pos gensym6 : gensym7)
(perceived-pos gensym0 : area_id_0)
(remembered-pos gensym4 : gensym1)

Plan:
L1: (move gensym0 area_id_1 area_id_0)
L2: (object-search-in-room
        gensym0 gensym6 area_id_1)
L3: (approach-person
        gensym0 gensym4 area_id_0)
L4: (tell_val_perceived-pos gensym0
        gensym4 gensym6)
```

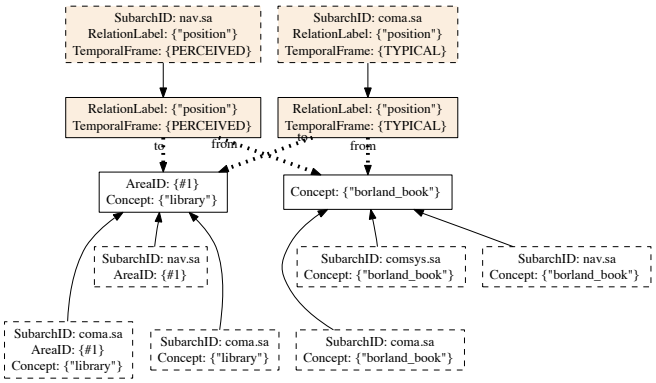


Figure 6: Perceived location of the book

In the above, `gensym7` is the binding union of the library. Using the **AreaID** feature from the this union, the planner issues a command to the nav SA which moves the robot to the library (fulfilling step `L1`). As with all other steps in the plan (including the information-processing ones), the results of this action are checked by MAPSIM to determine whether it has completed successfully or whether replanning is required. This check is performed by inspecting the planning state and comparing it to the expected state. This means that all actions must have effects that are visible on the binding SA (for subsequent translation). Once the check has passed for `L1` (confirming the robot has arrived in the library), the planner issues an object search command to the object SA. The Explorer searches the room as described previously. Once the object is found, the nav SA adds it to the navigation graph. Since it is part of the current spatial context, it is also exported to the binder in the form of an object proxy, which is connected to the room's proxy by a new position proxy. This position proxy has a `PERCEIVED` temporal frame.

The new proxies generated by object search bind to the existing complex (pictured in Figure 5), resulting in the structure in Figure 6. This binding provides the original comsys SA book proxy with a perceived position (in addition to its typical one). With this knowledge in the planning state (i.e., the effect of `L2` is verified, which satisfies one precondition of `L4`), the planner is able to trigger the remaining steps in the plan: moving to the user and reporting the perceived position. A move command is send to the nav SA referencing the **Location** feature from the person proxy. Once close to this location, a `tell-val` is sent to the comsys SA to communicate the book's location to the person. A content generation component in the comsys SA uses the contents of the binder (see Figure 6) to generate the utterance "the Borland book is in the library", thus completing the plan and satisfying the original goal (that the person knows the position of the book).

5 Discussion

The preceding sections illustrate how our architectural ideas come together to create a control system for intelligent behavior. Although the surface form of the scenario does not present much in the way of novel interactions, the PECAS architecture and the multi-level spatial representation provide a novel system-level approach with a number of important features. Including the binding SA in the architecture allows multiple modalities to collaborate on problems that a single modality in isolation would not be able to solve. E.g., in the Explorer the comsys SA initially provides a description of an object based on natural language input, conceptual mapping then extends this description, and vision finally completes it. Whilst other systems may include elements of cross-modal fusion, we have taken the additional, novel step of using the results of fusion to provide input to a continual planner. This allows the behavior of multiple modalities to be marshalled in pursuit of system goals in a general, extensible manner. Using continual planning PECAS achieves this in a way that is responsive to external change and certain types of failure. In PECAS all this is true both of actions that have physical effects, and of internal, information-processing actions.

Both the theory and implementation of the Explorer system and PECAS are works in progress, so we can identify many areas that need further study, or currently limit our approach. E.g., while the the use of MAPSIM provides many of the strengths of our work, planning occurs at quite a high level of abstraction. This consequently also applies to interactions between subarchitectures, and between the Explorer and the world. Whilst this has some advantages (e.g., subsystems are free to interpret commands in modality specific ways, something discussed in more detail below), it may be a hindrance to more closely coupled interactions between behaviors, such as positioning the robot to see an object that it is trying to pick up. Also, actions used in the PECAS architecture must have effects that are visible at the level of binding proxies, which may not hold for actions that have effects in a single SA. Our system also relies on many translations between formalisms. Whilst our structured support for this (via binding) is a clear strength, in practice the translations can become somewhat arbitrary and hard to maintain.

5.1 Approaches to Control

The HYCAS workshop aims to investigate issues of hybrid control in autonomous systems, so what lessons can we learn from the work presented here? In all PECAS systems we have a number of control patterns working in parallel. At the lowest level we can reasonably discuss in terms of architecture, components typically run in one of two modes. Either they perform continuous processing which provides a stream of data to working memory, or they wait for a particular event which triggers some processing (which may or may not result in a change to working memory). In this case events may either be external to the system (e.g., a sensor, such as a microphone, being triggered), or internal (where an event describes a change to working memory contents). Within a SA these types of processing behaviors happen concurrently (with SAs also working in parallel to each other). Control of SA-level processing is typically constrained at design-time, when components are set to listen to particular types of events. At run-time these events, and mediated access to the information they describe, provide implicit synchronization during processing. Thus PECAS does not provide explicit control strategies within SAs (although a few common control strategies tend to be reused).

As described in the preceding sections, the path to high-level control in PECAS comes via SAs exposing modal con-

tent to the rest of the system via the binding system. The process by which this occurs plays a major role in system control. Binding monitors provide abstracted representations of SA-local content. They typically do this based on three different triggers: SA-internal events, SA-external events, and on-demand. The first of these is the most basic case: the generation of a new local representation triggers the SA's binding monitor to generate a proxy. This happens in the Explorer for discourse referents in the comsys SA. The second case, SA-external events, typically provides a way for the existing binding state to influence the generation of further proxies (one of the limited, distributed, forms of attention in PECAS): the monitor listens for both SA-internal and -external events, then, when some particular events co-occur, it generates a proxy from some local content. This happens in the Explorer when the conceptual mapping SA provides proxies in response to proxies generated by other SAs (e.g., when the comsys SA generates a proxy for an object, the coma SA provides additional proxies to bind with it). The final case, on-demand monitor operation, occurs when a binding monitor is explicitly asked (rather than implicitly triggered) to provide information about a particular entity already represented in the binding SA. This approach is used by the system to deliberatively add information to the binding system. This is typically done during the planning process (as an element of on-demand state generation).

The first two of these binding monitor triggers represent additional design-time control decisions within PECAS systems. The designer explicitly chooses which SA and system events should cause information to be shared via binding (and thus added to the planning state for system control). The underlying assumption is that the system will need high-level access to this information regardless of context, and therefore this hard-wired approach is acceptable. The latter case, on-demand triggering, provides a system with explicit control over the information shared between all SAs and used for planning. We expect this approach, whether driven by planning or other mechanisms, to become the dominant approach in future PECAS systems. The alternative (implicit control over the contents of the binding SA) would place the system entirely at the mercy of reactive control, potentially flooding the binding SA with irrelevant or redundant information.

Binding monitors typically provide two types of abstraction: level-of-detail abstraction and temporal abstraction. The former has been taken for the implicit meaning of "abstraction" in the preceding sections: translation of a complex modal representation into a less complex amodal representation. Temporal abstraction is often implicit in level-of-detail abstraction, but it is important to make its presence explicit as it influences our control approach. Changes within SAs typically occur at a rate linked to the rate of change of sensors used for that SA's modality or the processing schemes used to interpret the results of those sensors. E.g., in the nav SA the pose of the robot is updated by SLAM at 5Hz, in the comsys SA elements of the discourse references are incrementally updated during an utterance interpretation (and across multiple utterances if they are reused), and in the obj SA object positions are updated as close to framerate as the system can manage. If the planner, or any other deliberative system, had to take control decisions using information at this level of description from multiple SAs, its decisions would only be valid for that length of time all of these representations remained unchanged (a number limited by the most volatile item of information). This would make system-wide control rather difficult. Binding monitors ameliorate this problem by only propagating *relevant* changes from the SA level to the binding SA. What constitutes a relevant change is both SA- and task-specific, but often relevance is coupled to the potential of the change to significantly alter the global state of the system. Temporal abstraction occurs because significant changes typically do not occur at the same rate as all changes; they often happen much less frequently. This highlights the close coupling between temporal abstraction and level-of-detail abstraction, as the latter defines our global state. This fact is often relied upon in systems which operate on multiple levels of abstraction. In this sense the role binding plays in PECAS can be meaningfully compared with the definition of an *interface layer* in the work of Wood (e.g., [Wood, 1994]). Interface layers are where a designer identifies critical points in the representations used by a system. These are points at which the representations become suitable for particular types of reasoning tasks. The identification of these layers is crucial for system control; they provide a way to match up representations with decision making approaches, e.g., detailed, dynamic representations for reactive control, and more abstract, stable representations for deliberative control. So, to reiterate an important point, unions and proxies (and to some extent the actions used by the planner) represent a stable point in the space of representation used by PECAS systems. Without them we would not be able to use planning (which requires such stability) to control system behavior.

From a control perspective there are two interesting aspects to our use of planning. First, as mentioned previously, we use continual planning: we integrate execution monitoring and replanning into our high-level control system. This provides a form of closed-loop high-level control for our system, where the effects of actions are monitored relative to expectations established by their definitions, and replanning is triggered if these expectations are violated. Second, the planner only has an opaque interface to the actions themselves. Rather than being concerned with how each action is implemented, PECAS only requires that the implementing SA abides by the contract provided by the action definition; otherwise planning and monitoring will fail. This is in contrast to other systems (e.g., 3T [Bonasso *et al.*, 1997]) where high-level control is used to schedule behaviors all the way down to the lowest-level (e.g., skills) too. By adopting a less exacting approach to action execution we allow each SA to interpret the action in a contextually appropriate way. SAs may choose to use one or many components to execute an action and may go through as many intermediate steps as required. This allows a single high-level control action to become a multiple step lower-level action, e.g., when an action results in a dialogue, or a visual search behavior. Of course, this means the planner is unable to directly influence the creation or scheduling of these lower level tasks. This is not a problem in our current domains where actions do not compete for resources across SAs, but in future this could become a problem. Possible

solutions include making the actions available to the planner less coarse but still not providing a one-to-one mapping to SA-internal actions (i.e., giving it tighter control over SA behavior), or annotating actions with resource constraints.

In summary, the overall behavior of a PECAS system, including the Explorer instantiation described in this paper, emerges from the interaction of reactive and deliberative control systems at multiple levels of abstraction. Multiple concurrent components within SAs are controlled implicitly by design-time event-subscription rules, and use CAST's event mechanisms and working memories to synchronism their processing at run-time. Across the system a collection of binding monitors provide an interface at which representations become abstract and stable. This allows a single deliberative control process to interact with the multiple concurrent SAs. It is this interface level which allows a PECAS instantiation to solve some problems with deliberative approaches (e.g., cross-SA coordination) and others with reactive approaches (e.g., within-SA coordination and sensor and effector control) whilst remaining contextually appropriate and responsive to its environment (i.e., no single control strategy ever exclusively takes charge of the entire system). However, this approach currently relies on an external designer fixing the representations either side of the interface level. Whilst this is not necessarily a problem in the short-term, in the future we would like to investigate what properties define a good interface level so that new system designers will not have to make uninformed design decisions.

6 Conclusion

We described PECAS, an architecture for intelligent systems. PECAS is a new architectural combination of information fusion and continual planning. Its purpose is to plan, integrate and monitor the asynchronous flow of information between multiple concurrent systems to achieve a task-specific system-wide goal. We used the Explorer instantiation to show how this works out in practice. The Explorer instantiates PECAS around a hybrid spatial model combining SLAM, visual search, and conceptual inference, with the possibility to use spoken dialogue to interact with a human user. We described the elements of this model, and demonstrated using a realistic (and implemented) scenario how PECAS provides a novel approach to control for autonomous systems.

Acknowledgements

Supported by the EU FP7 ICT Cognitive Systems Integrated Project "CogX" (FP7-ICT-215181-CogX) and in part by the Swedish Research Council, contract 621-2006-5420.
For more information see `http://cogx.eu`.

References

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

[Brenner and Nebel, 2009] M. Brenner and B. Nebel. Continual planning and acting in dynamic multiagent environments. *Aut. Agents and Multi-Agent Sys.*, 2009. accepted for publication.

[Burgard *et al.*, 2000] W. Burgard, A. B. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun. Experiences with an interactive museum tour-guide robot. *Artificial Intelligence*, 114(1–2), 2000.

[Gálvez López *et al.*, 2008] D. Gálvez López, K. Sjö, C. Paul, and P. Jensfelt. Hybrid laser and vision based object search and localization. In *ICRA '08*, pages 2636–2643, 2008.

[Gross *et al.*, 2008] H. M. Gross, H. J. Böhme, C. Schröder, S. Müller, A. König, C. Martin, M. Merten, and A. Bley. ShopBot: Progress in developing an interactive mobile shopping assistant for everyday use. In *SMC '08*, pages 3471–3478, 2008.

[Hawes *et al.*, 2007] N. Hawes, A. Sloman, J. Wyatt, M. Zillich, H. Jacobsson, G. J. Kruijff, M. Brenner, G. Berginc, and D. Skočaj. Towards an integrated robot with multiple cognitive functions. In *AAAI '07*, pages 1548–1553, 2007.

[Hawes *et al.*, 2009] N. Hawes, M. Brenner, and K. Sjöö. Planning as an architectural control mechanism. In *HRI '09*, pages 229–230, New York, NY, USA, 2009. ACM.

[Ishiguro *et al.*, 2001] H. Ishiguro, T. Ono, M. Imai, T. Maeda, T. Kanda, and R. Nakatsu. Robovie: an interactive humanoid robot. *Int. J. Industrial Robot*, 28(6):498–503, 2001.

[Jacobsson *et al.*, 2008] H. Jacobsson, N. Hawes, G. J. Kruijff, and J. Wyatt. Crossmodal content binding in information-processing architectures. In *HRI '08*, 2008.

[Kruijff *et al.*, 2007] G. J. Kruijff, H. Zender, P. Jensfelt, and H. I. Christensen. Situated dialogue and spatial organization: What, where... and why? *International Journal of Advanced Robotic Systems*, 4(1):125–138, 2007.

[Kruijff *et al.*, 2009] G. J. Kruijff, P. Lison, T. Benjamin, H. Jacobsson, H. Zender, I. Kruijff-Korbayová, and N. Hawes. Situated dialogue processing for human-robot interaction. In H. I. Christensen, G. J. Kruijff, and J. L. Wyatt, editors, *Cognitive Systems*. Springer Verlag, 2009. to appear.

[Kuipers, 1977] B. Kuipers. *Representing Knowledge of Large-scale Space*. PhD thesis, MIT, 1977.

[Lison and Kruijff, 2008] P. Lison and G. J. Kruijff. Salience-driven contextual priming of speech recognition for human-robot interaction. In *ECAI '08*, 2008.

[Peltason *et al.*, 2009] J. Peltason, F. H. K. Siepmann, T. P. Spexard, B. Wrede, M. Hanheide, and E. A. Topp. Mixed-initiative in human augmented mapping. In *ICRA '09*, 2009. to appear.

[Shanahan, 2002] M. Shanahan. A logical account of perception incorporating feedback and expectation. In *KR '02*, pages 3–13, 2002.

[Sidner *et al.*, 2004] C. L. Sidner, C. D. Kidd, C. H. Lee, and N. B. Lesh. Where to look: A study of human-robot engagement. In *IUI '04*, pages 78–84, 2004.

[Siegwart and et al., 2003] R. Siegwart and et al. Robox at expo.02: A large scale installation of personal robots. *Robotics and Autonomous Systems*, 42:203–222, 2003.

[Wood, 1994] S. Wood. *Planning and Decision Making in Dynamic Domains*. Ellis Horwood, Upper Saddle River, NJ, USA, 1994.

[Zender *et al.*, 2007] H. Zender, P. Jensfelt, and G. J. Kruijff. Human- and situation-aware people following. In *RO-MAN '07*, pages 1131–1136, 2007.

[Zender *et al.*, 2008] H. Zender, O. Martínez Mozos, P. Jensfelt, G. J. Kruijff, and W. Burgard. Conceptual spatial representations for indoor mobile robots. *Robotics and Autonomous Systems*, 56(6):493–502, 2008.

A Survey of Motivation Systems for Intelligent Robots

Nick Hawes

Intelligent Robotics Lab, School of Computer Science, University of Birmingham, UK

Abstract

The ability to achieve one's goals is a defining characteristic of intelligent behaviour. A great many existing theories, systems and research programmes address the problems associated with generating behaviour to achieve a goal; much fewer address the related problems of how and why goals should be generated in an intelligent artifact, and how a subset of all possible goals are selected as the focus of behaviour. It is research into these problems of *motivation*, which this article aims to stimulate. Building from the analysis of scenario involving a futuristic household robot, we extend an existing account of motivation in intelligent systems to provide a framework for surveying relevant literature in AI and robotics. This framework guides us to look at the problems of *encoding drives* (how the needs of the system are represented), *goal generation* (how particular instances of goals are generated from the drives with reference to the current state), and *goal selection* (how the system determines which goal instances to act on). After surveying a variety of existing approaches in these terms, we build on the results of the survey to sketch a design for a new motive management framework which goes beyond the current state of the art.

Key words: motivation, planning, goal-directed behaviour, agent architecture

1. Introduction

It is generally acknowledged that *goal-directed behaviour* is one defining feature of intelligent autonomous behaviour (Franklin and Graesser, 1997). This is usually taken to mean that an intelligent system should be able to perform actions to alter the current state of the world (where the world also includes the system itself) to satisfy some need. To date, researchers have succeeded in producing many systems that display this kind of goal-directed behaviour within a single task or domain, but have made little progress on more general systems able to satisfy their needs across many different domains. This is a function of the fragmented nature of AI as a field of study: sub-disciplines of AI study domains to which their technologies are directly applicable, with limited concern for their role in more general-purpose systems. One effect of this narrow focus is that surprisingly little attention has been given to the issue of where goals for an intelligent system come from; it is often assumed that they are produced in some other subsystem, external to the one being actively researched (Jennings et al., 2006). However, as we start to study designs for *integrated intelligent systems* which are able to use multiple interacting competences to solve a wider variety of problems than those addressed by a single sub-domain of AI, we can no longer ignore the problem of goal generation and a host of associated, cross-cutting, problems[1]. We ultimately require architectural solutions to many such interrelated problems if we want to make progress towards the kinds of general-purpose intelligent systems that have long been promised (but not delivered) by AI.

More specifically we are interested in studying mechanisms for *goal generation* which allow a system to generate its own goals to satisfy pre-determined needs, and *goal management* which select which of these goals are subsequently allowed to influence the system's behaviour. As these mechanisms encompass different states and process which *motivate* a system to behave in a particular way, we refer to then collectively as a *motive management framework*. This article supports our interest in such a framework with an example scenario (see Section 1.1), before building on previous work to identify requirements for goal generation and management. Armed with these requirements we then survey the existing literature (Sections 4.1 to 5) before sketching a design for a motive management framework (Section 6).

1.1. A Motivating Scenario

We can illustrate the problem of motive management with an example scenario from an application domain which is currently a firm favourite in integrated systems projects: the home help robot. Let us imagine a futuristic world in which the fields of vision, manipulation, dialogue, navigation etc. have yielded technology which is powerful and robust enough to be deployed on robot which is sold as "Alfred", a butler for a modern age. When Alfred is first unpacked by its new owners it is given a tour of their family home. This tour (as is becoming traditional in HRI, e.g. (Peltason et al., 2009; Zender et al., 2007)) is an opportunity for Alfred's owners to indicate and label important rooms and features of the house ("this is the kitchen", "through

[1] We use the term "cross-cutting" to refer to those problems which *cut across* the concerns of other subsystems in a typical intelligent system. Such issues may include problems such as goal generation and management as discussed in this article; attention and resource allocation; cross-modal inference; execution monitoring; and the generation and use of amodal representations.

there is the bathroom"); point out objects ("the dishwasher is under the sink", "that's my priceless Ming vase"); and perhaps provide Alfred with some knowledge about the day-to-day running of the home ("we like to have breakfast at 7 am", "try not to leave tyre marks on the wooden floor"). Given what we know about typical human behaviour, state-of-the-art AI techniques and the tasks such a robot might be required to do in the home, we can add further detail to this scenario. First, we can assume the tour given to Alfred only covered a subset of all the information about the home that it will need in order to carry out tasks. For example, different members of the family may use different names for rooms and objects; Alfred may not have been taken into or shown all the rooms in the house; and the tour will certainly have not included indications of the locations of all the objects (both stationary and dynamic) that Alfred might have to know about for all future tasks. Second, given the difficulty of operating in natural, previously unseen, environments, we have to assume that Alfred cannot be completely certain about all the information provided to it during the tour. Perhaps it didn't manage to reliably segment all the things indicated to it from the background, or successfully segment the large-scale space of the building into chunks against which it can resolve the room names provided during the tour. Finally, we will assume that Alfred should not just wait for instructions to perform a particular chore: it should seek out opportunities to assist the family whenever it can. For example, it might clear dirty mugs from a side-table, pick up toys from the floor, or empty the washing basket into the washing machine when the basket is full, all without being asked.

With this scenario in place, we can then consider what Alfred should do when the initial tour concludes and it is left to its own devices (literally). It will have a number of gaps in its knowledge about its environment, and it should assume that the humans which will interact with Alfred will take all these limitations for granted when giving it tasks. It will also have a list of tasks it could perform immediately (such as clearing up the packaging it arrived in) and tasks that it might need to perform as acts of routine (self) maintenance (such as connecting to its manufacturer's website to check for software updates, or recharging its batteries). Alfred must also be aware that a human might give it a task at any time (such as fetching something or passing on a message). So, how should Alfred proceed? First it must be able to construct a list of the things it thinks it should do, and from this reason (in whatever way) both about which of these things it can do, and which things it might prefer to do. In this process Alfred must take into account at least its limited hardware resources (it can only be in one place at any time and has only one set of sensors and effectors). In addition to basic physical constraints, Alfred would ideally be able to consider the utility (in a general, not-necessarily-numeric, sense) of doing particular things over others, and also consider attempting to do some subset of all the possible things at once, rather than each thing in sequence. There are many different rationales that Alfred could use when making a selection. For example, after the tour Alfred could first choose to visit every room in the house in order to build up a map of the building (the rationale being that such knowledge is required to support many other things it may be required to do). Alternatively Alfred could assume that the rooms it has been shown are all the important areas of the house, and instead spend its immediate future locating objects which it might commonly need to know about to perform future tasks (vacuum cleaner, PC, TV, oven, fridge etc.). Or it could interleave these two types of behaviour. Or Alfred could instead attempt to meet other members of the household (to fill the gaps in its local vocabulary). Or perhaps, following a different approach entirely, Alfred should position itself in a prominent position in the house in order to solicit chores. Or immediately start doing the chores it determines need doing. As you can see from this brief and incomplete treatment of a simple scenario, the possibilities for future behaviours are numerous, and a variety of information is needed to help choose between them.

Given that Alfred is not stymied by choice and starts following a selected course of action, we are faced with another set of related problems. What should Alfred do if it encounters a member of the household who then provides it with something else to do? Should Alfred stop what its doing, or continue until its free? Also, what if Alfred's behaviour uncovers new possibilities for action (e.g. if a new room is discovered), should it now take this into account immediately or later? Also, what if new information discovered through action changes the information used during the previously discussed selection process or renders some of the proposed future behaviours pointless (or at least less desirable?),

These problems start to outline the requirements for a motivation management framework capable of managing the goals of an intelligent system. Although we have illustrated the need for such a system with a far-fetched scenario, current integrated systems, including the ones we are actively developing, are starting to present similar requirements. To address this need we are ultimately interested in developing a motivation system for a future general-purpose architecture suitable for intelligent robots operating in the real world. To move towards this point we must first make the requirements and possible designs for such a system more explicit and explore existing solutions to previous problems. This is the purpose of this article.

2. Background

We wish to provide an architectural account of mechanisms that can cause a system to act to address one or more of its needs. A need is defined in Sloman et al. (2005) as the relationship between a system and that which is necessary to bring about or maintain a possible state of affairs. This work also defines *desire-like states* which are states whose function is to get a system to maintain or achieve a possible state-of-affairs, and *belief-like states* which provide information to allow the system to achieve its desire-like states. Goals are an example of a desire-like state. Information derived from sensors, and the information derived from this and other (stored) information, provides belief-like states.

Our approach to architectural work in informed by Sloman's CogAff Schema (Sloman, 2003). The schema is a method for characterising (i.e. describing, not prescribing) the different

forms of processing which can occur within an intelligent system. It distinguishes three types of processes: reactive, deliberative and meta-management. In brief, reactive processes can be considered as collections of responsive condition-action rules which maintain no explicit representations of alternative choices of action. Deliberative processes can be considered as processes which explicitly represent and choose between alternative processing and action opportunities. Finally, meta-management processes are those processes which manage the processing of other management processes within the system (processes of all other types, including meta-management processes). A system can be composed of processes from just one of these three types, all of them, or some selection.

By considering system composition using the CogAff schema, we can start to investigate what it means for differently-composed systems to demonstrate goal-directed behaviour. To do this we will first informally define this type of goal-directed behaviour as a sequence of actions which alter the world to bring about a state which satisfies one or more needs of the system performing the actions. This broad definition will cover not only the examples given in our far-fetched scenario, but also much simpler artifacts performing the tasks they been designed to do (e.g. computer game characters playing their roles, robots following humans, chatbots answering questions). It is also applicable to biological systems from humans to bacteria. Given that this behaviour exists, we want to (invoking Dennett's design stance (Dennett, 1978; Sloman, 2002)) consider what possible designs can give rise to it. Or, more precisely, what possible design can give rise to just a small part of this behaviour: the systems which produce, select and otherwise manage the purposes to which the behaviours are put.

Given the CogAff schema's definition of reactive processes (as not making use of explicit representations of alternative courses of action), we can consider that they mostly make use of implicit knowledge to generate goal-directed behaviour. That is, the prior knowledge of the effects of the goal-directed behaviour generated by reactive systems must be specified at design-time. In the case of artifacts this will be done by a human designer. In the case of natural systems this job is done by evolution. Given our knowledge of behaviour-based systems (e.g. (Brooks, 1986; Agre and Chapman, 1987)) we know that a collection of behaviour-producing modules, where each module is intended to serve some pre-defined purpose, can give rise to goal-directed behaviour. Goal management in reactive systems is often performed by allowing particular behaviours to control the system (e.g. via winner-takes-all selection or action fusion(Arkin, 1998)). Whilst these mechanisms prove useful for situations where a close coupling between sensors and effectors can solve all problems a system is presented with, we have seen that they are not adequate in more complex situations, and not easily extended or maintained over long periods. In particular, where the trade-offs between two courses of action are not immediately apparent (e.g. between Alfred exploring the house further or cleaning the floor of the room it's in), a ballistic selection mechanism is just not capable of the (possibly counterfactual) reasoning necessary to come to an informed decision. Taking a step back, it is clear that the systems we are interested in designing in both the short- and long-term must make use of many deliberative mechanisms to determine how they will achieve the things they are designed to. Purely reactive mechanisms – mechanisms which, by definition, cannot determine the results of a deliberative process – will be inadequate to successfully judge choices between different deliberatively-determined behaviours. However, we must not completely ignore the benefits of a reactive approach to managing goal-directed behaviour; reactive processes are quick to change behaviours in the face of new stimuli, and deal well with the asynchrony and parallelism inherent in the real world.

Although it is possible to consider a system composed purely of deliberative processes, there are very few examples of such systems produced as autonomous intelligent agents in their own right[2]. Agents require mechanisms for sensing and acting on their world, and these mechanisms are usually best designed as reactive processes (to reduce latencies). A purely deliberative agent would generate goal-directed behaviour by reasoning both about which behaviour to engage in, but also about which stimuli and world states should give rise to which behaviours. It is this latter process which restricts the usefulness of purely deliberative agents; the combinatorics of the problem (explicitly comparing all combinations of stimuli to all possible goals and behaviours) in any non-trivial domain will result in undesirable latencies when applying deliberative reasoning[3].

In practice, most autonomous, intelligent systems with deliberative processes are designed as reactive-deliberative hybrid systems (which we will refer to as "hybrid systems" or "hybrid architectures" for the remainder of this article[4]). Such hybrid systems typically couple the low-latency operation and predefined search spaces of reactive systems with the flexible, exploratory reasoning of deliberative, or proto-deliberative (Sloman, 2006), systems. This often results in the use of reactive processes to suggest various possibilities for action, deliberative processes to work out how a particular action should be performed, and then reactive processes to execute the action. There are many possible variations on this scheme, but, as we shall see, this has become a standard approach for building autonomous systems. Therefore it is within the space of reactive-deliberative hybrid systems that we shall sketch our requirements and framework for a motivation system. We will occasionally return to the distinctions made in this section when we review the existing work on motivation systems.

3. Requirements for a Motivation Management Framework

Previous work by the Cognition and Affect group at the University of Birmingham (e.g. (Beaudoin, 1994; Wright et al.,

[2]Of course there are a great many examples of purely deliberative systems in AI. But they are almost always used as stand-alone problem solvers, rather than *autonomous* systems capable of pursuing their own goals.

[3]That is not to say that such an approach is impossible, just that in many domains where interaction with the world is required, a purely deliberative design is not one which adequately meets its requirements.

[4]There are many other uses of the term "hybrid system" in AI and the engineering sciences. Our usage of the term in this article should not be confused with the usage of the term in dynamic systems or neural networks.

1996; Wright, 1997) has examined both the requirements for motivation management systems and a design to satisfy the requirements they identified (MINDER1, as described in Section 4.2.3). In particular, Chapters 3 to 6 of Luc Beaudoin's thesis (1994) present a detailed investigation of motive processing. Whilst this work largely subsumes the requirements we discuss here (and thus we encourage interested readers to consult this too), we repeat a small part of the exercise here for a number of reasons. First, Beaudoin's work was performed from a theoretical perspective. Whilst this allowed him to address a wide range of issues, it meant that his work was only partially informed by experiences of implementing intelligent systems. Second, the implementation that was done assumed a largely homogeneous system using a unified representation and interacting with a simulated environment. Finally, the work was done at least fifteen years ago. We have recent experience of building a number of complex intelligent systems (robots and virtual characters) from heterogeneous, distributed components (a necessity for most embodied systems) to solve tasks which approximate some of those from our motivating scenario. This experience gives us a slightly different, if not totally unique, perspective on requirements for a motivation system[5].

We will start by assuming that a system can engage in two types of goal-directed behaviour: reactive goal-directed behaviour and deliberative goal-directed behaviour. Reactive behaviour may just happen without the system making an explicit choice to trigger it (e.g. obstacle avoidance), although it may be able to make choices to suppress or enhance such behaviour (but we will ignore that for now). In contrast to this, deliberative goal-directed behaviour requires that the system takes a number of explicit steps: a goal representation must be created, this goal must be selected as one of the goals the system intends to pursue, then the system must generate the necessary behaviour to achieve the goal. This first pass allows us to identify a number of initial requirements which we can explore in more detail.

1. A system requires explicit goal representations which it can reason about.
2. A system requires at least one process capable of creating goals.
3. A system requires a process capable of collecting goals and then selecting which ones should be acted upon.
4. A system requires a process which can generate goal directed behaviour from a collection goal and the available resources.

To reduce the space of possible mechanisms under discussion, we will now make a further assumption: the required process (item 4) that allows a system to take goals and turn them into behaviour will be some kind of planner coupled with an execution system. This planner will take a conjunction of goals and produce a series of actions (i.e. a plan) which the system can execute to achieve the goals. The execution system will take this plan and drive the rest of the system to perform the actions it specifies. We make this assumption because these parts of the problem are relatively well-studied, and elements of existing systems (e.g. (Brenner and Nebel, To appear; Knight et al., 2001; Bonasso et al., 1997; Nilsson, 1994; Firby, 1987)) can be assumed without losing much generality in our work.

3.1. Goals

Given this assumption, we can now require that the goals used in our system should be at least expressible in a format that is usable by a planning system. This is typically a logical form describing a possible state of the world. The nature of this description is limited only by the power of the planning mechanisms used to generate system behaviour. For example, if the planner can handle numerical data (e.g. (Eyerich et al., 2009)) then (a logical representation of) "increase battery charge above 90%" is a valid goal, whereas a system incapable of handling numeric input may only be able to use "charge battery". Likewise the use of planners capable of reasoning about time, resource constraints and uncertainty would produce a system capable of being given more detailed goals than a system with using a STRIPS-like representation.

In addition to a state description, Beaudoin also identifies attributes which should be associated with a goal. These include descriptions of the goal's *importance* and *urgency*; information which will allow informed management decisions to be taken by the system. Beaudoin also identifies other attributes required in a goal structure. These will be introduced in Section 3.3.

3.2. Goal Generation

Once we require our system to have explicit goals, we require processes which can generate them. Following Beaudoin, we shall refer to these processes as *goal generators*[6]. Where goals come from is an issue that is often overlooked in work on intelligent systems, so it is something we shall expand upon here (and in the following literature survey).

Our basic assumption is that a goal generator is a specialised process which can explicitly predict and describe future states which will satisfy a need of the system. These states are communicated to the rest of the system as goals. Goal generators are therefore the processes which turn a system's *drives* into goals, where a drive is a disposition to satisfy a need (i.e. a desire-like state). For example, Alfred may have a drive to maintain a minimum level of battery charge (where the need is not to have a flat battery), and a drive to perform tasks specified by its owners (where the need is to be a useful domestic assistant).

To go from a drive to a goal, a goal generator must inspect the belief-like state it has access to (e.g. information from sensors,

[5]The main differences in perspective are down to the roles of different representations within a system, the influence of parallelism and asynchrony in robotics domains, and a more informed (due to developments in the fields of planning etc.) view on the possibilities for deliberative systems. None of these things lead to major deviations from the work of Beaudoin, but they do lead to us both make additions to the work and choose to emphasise some things he gave less weight to.

[6]Beaudoin (1994) actually uses the term *goal generactivator* because the processes can generate new goals or activate existing, but suppressed, ones. We use the shorter term *goal generator* for conciseness.

and information derived from this) and determine both whether it is appropriate to instantiate a new goal derived from its drive, and what the form of this goal should be. The exact nature of this process in an implemented system will depend greatly on the drives, and on the representations and algorithms used in the system. Regardless of this, it is useful to make the elements of the goal generation process explicit as it helps us to avoid mixing up similarly-purposed processes.

When an intelligent system is acting in the world, any state change which occurs internally or externally (regardless of whether this was a change enacted by the system or not) may provide the information necessary for a goal to be generated. State changes might be as simple as a scheduled time being reached, or a resource falling below a particular level, or they might involve more complex chains of processing based on the results of processing belief-like representations built up both from sensing and reasoning. As changes in the world (and thus a system's internal state) can be unpredictable, such state changes may occur at times which are unpredictable by the system. As these changes may provide opportunities for a system to satisfy one or more of its needs, it is important that the system is able to respond immediately (if appropriate). This gives rise to one of the important requirements for both goal generators and the motive management framework in general: it must be able to operate concurrently with other systems (particularly action-producing ones), and allow asynchronous changes in state to produce goal-directed behaviour. Without this requirement a a system with multiple drives operating in the real world may not be able to produce goal-directed behaviour in a timely fashion, and may thus miss opportunities for satisfying its needs.

To examine the requirements for goal generation further, we can briefly explore possible ways in which goal generators may operate. Perhaps the simplest goal generator we can imagine is one which encodes a desire to keep a given variable at a particular value. As the system runs, this goal generator monitors the given variable, comparing it to the desired value. When it determines that the value of the variable does not match the current state, it generates a goal which, if adopted and acted upon, will return the variable to the desired value. Although this example is simplistic, we can use it to highlight aspects of the process which are either desirable, or may cause problems during system design and implementation. First, the simple generator only monitors the information it needs to determine whether a goal should be generated. It does not monitor what the the system is currently doing, what other goals may have been generated, or could potentially be generated to satisfy other drives. In this sense it is specialised to generate a goal regardless of the rest of the behaviour of the system. Second, the generator must be able to express the future state which will satisfy its drive. In the simple example this is described as a goal which will cause the system to return the variable to its desired value. However, this assumes that the deliberative systems later in the processing chain use a representation which is able to express this future state in a direct manner. This will often not be the case in hybrid architectures, or architectures composed of heterogeneous modules. It is possible to foresee two reasons for this. First, it could be possible that the representations used by the goal generator to monitor the variable are not expressible in the goal language used by the subsequent systems. Many planning systems cannot handle numerical values, and so would not be able to process a direct expression of the desired variable state provided as a goal. A second point is that, as a system may not possess actions which can directly influence particular states (notably those which are mediated by interactions with the external world), the goal generator must translate its desired state into a state which is achievable by the system, and where the achievement of this state has the effect of also achieving the state the goal generator's drive is related to. For example, if our simple goal generator was monitoring battery charge, it would create goals when the charge fell beneath a particular threshold. If the system's deliberative processes cannot reason about battery charge directly, and instead must use abstractions such as *batteryCharge(full)* or *batteryCharge(half)*, the goal generator must be able to generate the appropriate abstraction (*batteryCharge(full)* in this case) from the state it has access to. In general, given the assumption that intelligent systems must employ a heterogeneous collection of processing modules to function effectively, we can deduce that any goal generator not operating directly on the representations of the planning or goal-management systems will need to perform some kind of translation between the precise state which will satisfy its need and the goal which will drive the system to satisfy it.

3.3. Goal Management

As the Alfred example demonstrates, an intelligent system can have multiple goals. Following on from our previous deconstruction of goal generation, we can highlight two forms of multiplicity: a system can have multiple drives, where each drive is encoded in one or more goal generator; and each goal generator could produce multiple goals, representing multiple possible routes to satisfying a need. Because goal generators should not monitor other states beyond those related to their needs, it would be unsafe and inefficient to let any generated goal become the current goal of the system. Instead we require some form of *management system*. This system must accept the goals produced by the goal generators; select which goal or goals should be pursued (with reference to any ongoing goal-directed behaviour); and then trigger the appropriate behaviour generation mechanisms (e.g. planning and execution) to achieve the selected goals.

The requirements on the input functionality of the management system are quite simple. As goal generators produce goals asynchronously and in parallel, the management system must be able to accept new goals in this manner too. The management system should not block the operation of goal generators when it accepts new goals, as this would interfere with their ability to respond to new situations. Additionally, the management system must be able to respond to new goals in this manner; if new goals can be generated in such a dynamic fashion, the management system must also be able to engage its decision procedures in similarly (rather than delaying later processing stages until a fixed point is reached).

The selection of which goal or goals a system should peruse is perhaps the largest problem in this field. Before exploring requirements on the selection process and its dependencies, we will outline the basic elements of the selection process. As with previous descriptions, these descriptions can be read as summaries of the parts of (Beaudoin, 1994) which are relevant to this process (e.g. Chapter 4). The management process has a set of goals to choose from. We will refer to these (following Beaudoin) as the *managed goals* (i.e. those which the management processes have accepted from the goal generators). It's role is to *select* which goals from this set should be *activated* as the goals which will dictate the future (goal-directed) behaviour of the system. In an extension to the CogAff work we assume that multiple goals can be combined into single goal for the system to pursue, as a goal for a planning system is typically a conjunction of subgoals[7]. The aim of the selection process is to ensure the system engages in behaviour which best satisfies its needs given the available resources and various contextual restrictions. This implies that the system is attempting to maximise some general, not necessarily single-valued or numeric, measure of utility. For the time being we will ignore formal, decision theoretic, views of utility maximization (e.g. due to the lack of reliable information available to a robot for this kind of decision making Scheutz and Schermerhorn (2009)), and instead adopt the requirement that, given the knowledge and resources available to it, the selection process should aim to take the management decision which results in behaviour leading to as many of its needs being met as possible.

Given some currently active goal-directed behaviour, the possible actions a goal management process can take are to either interrupt the current behaviour with a new goal (or conjunction of goals, possibly including the current goal) or to allow the current behaviour to continue. This ignores many other, more complex, possibilities such as scheduling goals for later adoption. We ignore these possibilities here to allow a simpler treatment of the subject.

The decision of which goal to select is an involved and complex one. There are many metrics which could be used in this decision, and many different types of decision-making procedures. At a high-level we can consider two types of (complementary) approaches: those based on achievability and those based based on cost/benefit analyses. Achievability-based decisions will determine whether the system can actually achieve the goal from the current state, rejecting goals which cannot be achieved. This process will require planning-like reasoning, the results of which could be re-used if the corresponding goal is adopted. Decisions made on the basis of a cost/benefit analysis will compare goals based how much they benefit (see previous discussion) the system at what cost. Cost will be a feature of the plan generated to achieve the goal, so achievability-based decisions should be taken before cost/benefit analyses are performed. By combining these approaches, the selection procedure should aim to select the goal which is both achievable and the most beneficial.

To aid the decision making process, Beaudoin identifies additional information which goal generators should attach to their goals. These are *importance*, *rationale* and *urgency*. Importance is intended as a (heuristic) representation of the benefit to the system of adopting the goal. It is added by the goal generator, rather than determined at a later stage, because the goal generator represents the need that the goal ultimately satisfies, and *benefit should be tied to the need, not to the individual goal*. It is possible that the same desired future state may satisfy many different drives. In this case it is arguable that such a goal should have a higher importance than a goal that only satisfies a single need (although this is conditional upon the relative importance of the desires involved). The rationale information attached to a goal is intended to make the connection between goal state and need more explicit. This field should describe the reason why the goal was generated. In our single-variable example the rationale would be the difference between the current value of the variable and its desired value. In the Alfred scenario rationales may include formalisations of statements such as "because my owner told be to do X" and "because I saw dirty dishes on the table". The rationale is important as it supports both the maintenance of goals (i.e. determining whether they are still applicable or important), and provides a first step in reasoning from a goal state back to the need which required it. However, it remains to be seen whether there are clear practical benefits of including rationale information; it is neither a complete description of the reasoning behind the generation of the goal (much of which could be implicit in the design of the goal generator) nor a concise but imperfect heuristic summary, thus it may not be truly useful for either deliberative or reactive decision making.

The final type of information that Beaudoin attaches to a goal, urgency, describes its temporal qualities. Its purpose is to tie importance to the passage of time, and possibly changes in state, in the system's world. The simplest notion to consider is that of the deadline beyond which the goal becomes unachievable or invalid (such as when the robot's battery runs out). This would provide one type of (binary) urgency information. This could be augmented with information that the importance of the goal increases monotonically as the deadline approaches. Richer representations of urgency may take into account time windows when the goal may be less costly to achieve or provide a greater benefit to the system. Urgency could also be specified in qualitative, rather than purely qualitative ways. For example, a goal's urgency may be dependent on state changes that may occur, or other behaviours the system may engage in. The ultimate purpose of urgency information is to support the goal management system in deciding upon and scheduling the future behaviour of the system. Goals that are more urgent (i.e. goals that should be achieved sooner rather than later) should take precedence over less urgent goals of similar (and perhaps greater) importance.

[7] In this framework we will consider disjunctive goals as separate proposals from goal generators, making the disjunction implicit in the selection process. A completely general framework would allow explicit disjunctions to be considered by the management process.

3.4. Summary

Before we leave this discussion of motive management, some final points should be made clear. Whilst the discussion has highlighted the requirements for a goal generation and management framework, it can only really motivate requirements on the types of information and decision-making procedures that such a framework should include. It cannot strongly motivate designs which group and structure representations and functions in particular ways. For this we must also be constrained by the domains in which we are deploying our hypothesised system, and, perhaps more importantly, the remainder of the architecture in which the motive management framework is deployed. This brings us to a related point. When considering possible designs and implementations for motive management frameworks, we must be aware of the co-dependence, or coupling, between many different aspects of possible systems. For example, if all the needs a system can have are pre-judged by a designer to have equal importance then its goal generators will not need to annotate goals with importance information and therefore its goal selection mechanisms will not need to take importance into account. A similar argument could be made for urgency. Additionally, given a potential scenario there can be much room for debate around the granularity and content of mechanisms involved, particularly the needs, drives and associated goal generators (i.e. what becomes a "top-level" goal for a system). This in turn influences the role of the subsequent behaviour-generating and -managing mechanisms: "smaller" goals may require less planning effort per goal, but a provide a planner with less opportunity to make optimisations across all future actions (as behaviour is generated in shorter chunks).

Even with many issues remaining open, we will find that the preceding discussion allows us to investigate different extant system designs in a more coherent way than would have been otherwise possible. In particular, we can use it to review the different approaches that have been taken, or can be taken, to:

- **encoding drives**: how an approach represents the needs of the system in its design.
- **goal generation**: how an approach combines the results of sensing and processing with its drives to produce goals to act on.
- **goal selection**: what mechanisms and information are used to select between conflicting goals.

4. Previous Work Related to Motive Management

In the following sections we will explore existing literature on architectures which feature motive management mechanisms. Most of the works surveyed explicitly address either goal generation or goal selection. The works that don't were included because they address it implicitly (i.e. they solve related problems as part of a larger system) and are representative of a collection of similar approaches. Some fields may appear to have been over-looked, but this is due to a desire to tackle the chosen literature in detail without expanding the survey to an off-putting length. We have not surveyed a great deal of work on action selection (e.g. (Tyrrell, 1993)), although there are many overlaps between work on action selection and on the various approaches to reactive planning and behaviour in the work we do cover. We have also not explicitly addressed work on rational decision making or meta-reasoning (e.g. (Zilberstein, 2008)). Although both of these fields are relevant to motive management (particularly to the issue of goal selection), we have chosen to focus on work which has resulted in complete intelligent systems, rather than individual algorithms. Once we have a motive management framework, this will provide a context in which we can look to apply results from these fields.

We present the literature in an order that roughly corresponds to the addition of the layers from the CogAff schema. We will start with reactive systems with minimal or no declarative representation of goals (i.e. behaviour-based systems, Section 4.1), proceed to proto-deliberative systems (i.e. behaviour-based systems with additional structure, Section 4.2), and conclude with hybrid systems (i.e. combinations of reactive and deliberative planning systems, Section 4.3). The categorisation of mechanisms this narrative structure provides is not particularly important (as the division between categories is often difficult to precisely position), but it allows us to discuss similar systems in subsequent (sub-)sections.

4.1. Reactive

Purely reactive systems typically encode their functionality as modules which compete in various ways to generate system behaviour. Such systems do not have explicit goal instances as described above, but instead implicitly encode drives into modules which perform the behaviour necessary satisfy their need: goal generation, selection and execution are merged into a single, ballistic, activation process. Such reactive systems are usually described as *behaviour-based*, as the behaviour which is generated is the key feature of such a system (rather than, for example, internal representations). Examples include the Subsumption Architecture (Brooks, 1986), the Agent Network Architecture (Maes, 1991) and Pengi Agre and Chapman (1987). If a reactive system is capable of merging the outputs of multiple active modules the we can describe the system as having multiple active goals. However, the lack of any explicit goal representation means that any such system is incapable of reasoning about the selection of one goal (or behaviour) rather than another. This is one of the reasons that behaviour-based systems have failed to scale to tasks beyond those requiring close coupling with the environment.

Bryson (2001, 2003) tackled the problem of a reactive system having multiple (possible conflicting) goals by embedding a reactive planning system into a structure called a *drive collection* in a Behaviour-Oriented Design (BOD) architecture. Each entry in the drive collection is a distinct reactive plan for satisfying a particular drive. Rather than allowing all drives to compete to control the system they are evaluated in a fixed priority order (i.e. the most important drive is checked first), with the first applicable plan assuming control of the system. This approach allows a designer more explicit control over the way the system chooses which behaviour to pursue. The ordering of

reactive plans in the drive collection represents a fixed importance evaluation across all of the needs a system has.

4.2. Proto-Deliberative

The limited applicability of purely reactive systems to a wide range of problems caused researchers to extend them in various ways. Typically these extensions add an additional layer to the system containing explicit representations of goals and the current state (i.e. desire-like and belief-like states), allowing behaviour to be directly managed at runtime (Gat, 1998; Bryson, 2003). In this section we will survey a selection of systems that demonstrate this approach.

4.2.1. Belief-Desire-Intention Systems

The issues of explicit goal representations and the ability to choose between goals are commonly associated with study of Belief-Desire-Intention (BDI) systems. Such is the prominence of this work that the term BDI has become short-hand for a general way of dividing up types of representations. In contrast to its general use, the term BDI also describes a rather specific class of system designs. We will treat these two uses separately.

The common usage of BDI assigns the kinds of things a system can represent into three classes: beliefs represent the information the system stores about itself and its world; desires represent things the system would like to do; and intentions represent desires which the system has committed to achieve. To position this usage relative to the terms we have already introduced (Sections 2 and 3) beliefs refer to the results of sensing and processing which are stored in the system somehow, i.e. belief-like states; the term desire encompasses both the drives which are encoded into goal generators and, depending on your view, goals which have been generated but not activated; and intentions are the goals which have been activated by the management mechanisms. Desires and intentions are both instances of desire-like control states. These folk-BDI definitions do not specify a design for a system architecture (except the requirement that it allows the BDI distinctions), and do not make some necessary distinctions explicit (particularly the decomposition of desires and intentions into the intermediate steps ranging from needs to behaviour).

The field which spawned this relaxed usage of the term BDI is the field of BDI agents (Georgeff et al., 1999). BDI agents are logic-based agents developed in the context of resource-bounded, practical or rational, problem-solving. There is no single canonical BDI architecture, rather there is a family of similar approaches inspired by the work of Bratman (1987), including the Procedural Reasoning System (PRS) and its derivatives (e.g. (Georgeff and Ingrand, 1989a; Myers, 1996)), and the Intelligent Resource-Bounded Machine Architecture (IRMA) (Bratman et al., 1988). So, rather than review the goal generation and management properties of the whole field, we will choose a popular exemplar, PRS-CL, and examine that[8].

[8] A similar approach to surveying BDI systems has been taken by other authors, including Georgeff et al. (1999) and Jones and Wray (2006).

PRS-CL (the fully-featured "classic" (CL) version of the Procedural Reasoning System) is a logic-based reasoning system which combines goal-directed reasoning and reactive processing in a single framework. It uses a unified logical language to represent beliefs, actions and goals. Its fundamental method of problem-solving is reactive planning. Its basic plan formalism is a Knowledge Area (KA), a fixed sequence of actions aims to achieve a particular goal. KAs can invoke other KAs to achieve subgoals, thus producing a plans through an hierarchical refinement approach. KAs are comparable to the Reactive Action Packages of Firby (1987) and the Teleo-reactive Programs of Nilsson (1994). Where PRS-CL differs from these systems is through the embedding of KA processing within an *intention structure* which manages which goals the system is currently pursuing. To achieve a goal the PRS-CL interpreter finds a KA that will satisfy it, then instantiates the KA with information from the goal and current state (i.e. the current database of beliefs). This instantiated KA (and subsequent refinements of it) is referred to as an *intention*. A PRS-CL system can have multiple goals, and can have multiple intentions too. In terms of our earlier requirements, a PRS-CL intention is comparable to a goal which has been activated by the management system (i.e. it is being actively pursued). PRS-CL does not have a goal management system; intentions are automatically created when a KA can satisfy a goal. In this manner PRS-CL reactively selects all goals which are achievable according to its procedural knowledge and beliefs, thus running the risk of becoming oversubscribed with things to do. In practice such situations are avoided either through appropriate KA/goal design, or though the use of meta-KAs. Meta-KAs are KAs which can operate on the intention structure itself, performing actions such as reordering the current intentions (thus determining which ones get acted upon) or suspending selected ones (allowing a distinction between active and non-active goals). Meta operations allow PRS-CL systems to implement some of the features we identified previously as desirable. For example, in a PRS-CL system for spacecraft system monitoring, a meta-KA is used to promote intentions to check the reliability of sensors ahead of all other intentions (Georgeff and Ingrand, 1989b). This is a system- and task-specific implementation of managing goals based on an implicit measure of importance.

In PRS-CL systems, drives are represented implicitly as *trigger conditions* associated with KAs. When a trigger condition is matched by a system belief the KA is automatically instantiated as an intention. This is perhaps the mechanism in PRS-CL which come closest to satisfying our previously stated requirements for goal generation (even though it generates intentions, i.e.. goals plus plans, rather than just a plan). Explicit goals in PRS-CL (i.e. statements which need to be matched against KAs) are typically added by an external source (e.g. a human operator or, as in the spacecraft monitoring example, another instance of PRS-CL).

In summary, PRS-CL has many features that satisfy our requirements for a goal generation and management system. Its primary weakness (in relation to our needs) is its inability to explicitly represent goals which have not been activated: all goals are ballistically activated (turned into intentions) as soon

as possible. This makes the process of goal management one of post-hoc scheduling of intentions, rather than one of selecting which goals should become intentions (although allowing intentions to be suspended after being first activated does allow some selection to take place). This limitation is understandable when one considers that practical (i.e. real-time, responsive) reasoning is one of the aims of PRS and that this is most easily achieved using reactive (ballistic) not deliberative (considering the consequences of alternatives *before* action) methods.

4.2.2. Soar

Soar is a computational model of human problem solving which has latterly developed into more general framework for developing knowledge-intensive agents Laird et al. (1987). It shares many features with PRS-CL, but is less formally constrained than other, logic-heavy, BDI systems[9]. One of the major similarities between the two approaches is their general approach to generating behaviour: the use of reactive, hierarchical refinement of pre-specified plan-like structures (KAs in PRS roughly map to *operators* in Soar). One of the major differences is the ability for Soar to overcome *impasses* (situations where no operator is directly applicable) by employing additional reasoning mechanisms and learning from the results (*chunking*).

In terms of goal generation and management, Soar has less built-in functionality than PRS-CL. Internal goal generation is only supported to overcome impasses, a fact which has resulted in some system designers overloading this mechanism to generate task goals (Jones and Wray, 2006), while others find other "idioms" to use to add the necessary functionality in implementation-specific ways (Lisse et al., 2007)). In general we can consider that these engineering solutions draw on the same (reactive) processing model of Soar, and thus can be directly compared to trigger conditions in PRS (i.e. goals are derived directly from matching rule conditions against a database of beliefs).

Unlike PRS-CL (which allows suspended intentions), Soar does not support non-active goals. As such it provides no standard mechanisms to select between different goals. If required, such mechanisms must be added on a system-specific basis using the existing functionality (i.e. via Soar operators). The lack of more general goal generation and management functionality in Soar is seen as a weakness by some of the researchers using Soar[10]. To overcome this, these researchers have started developing a BDI-inspired abstraction layer for Soar Lisse et al. (2007). This layer includes declarative goals (rather than goals implicit in generation rules) and *activation pools* for desired, active and terminated goals and their associated plans (where a goal plus a plan is comparable to a PRS intention). These pools give a Soar system the ability to have proposed but non-active goals (the desired pool), which is a pre-requisite of goal selection and management. The BDI layer uses Soar's built-in belief maintenance mechanisms to manage which goals should be active. This appears to be the only (documented) mechanism for determining which activation pool a goal should be in. No information is provided on, for example, methods to select which active goal should be preferred by the system.

4.2.3. MINDER1

The analysis and design work of Beaudoin (which yielded the NML1 architecture design (Beaudoin, 1994, Chapter 5)) was subsequently explored and implemented by Wright (1997) as the MINDER1 system. As with the previously described systems it takes a hierarchical reactive planning approach (using Teleo-reactive Programs (Nilsson, 1994)), but augments this with a *motive management* system which controls which goal the system is currently pursuing. MINDER1 operates in a 2D simulated world called the nursemaid domain. In this domain there a number of independent agents, called minibots, which the minder has to care for in a nursery. Minibots can run out of batteries or fall into ditches. The minder should prevent these situations from happening or rectify them if they do occur. As the minder only has a limited sensing range it must also patrol the nursery to detect the state of the minibots under its care. As the minibots are all operating in parallel in real time the minder's goals change asynchronously. This is an ideal environment in which to goal management approaches[11].

As MINDER1 is a direct descendant of Beaudoin's ideas, reviewing it allows us to review the performance of a design that was produced to satisfy the same requirements we have. MINDER1 has distinct goal generators for each system goal. Much like the previously reviewed systems it generate goals by matching conditions (which implicitly encode the system's drives) against a database of beliefs. As is required by Beaudoin's theory, goals are accompanied by an *insistence* value, which attempts to heuristically capture both the goal's importance (i.e. how crucial satisfying the drive that generates it is) and urgency (i.e. the pressure to satisfy to goal sooner rather than later). In MINDER1 these values are calculated as a function of the current violation of the underlying need. For example, the insistence of a goal to prevent a particular minibot from falling into a ditch is a function of the distance of the minibot from the ditch.

The insistence is used by components within MINDER1's motive management subsystem (Wright, 1997, p72). This subsystem maintains two disjunct sets of goals: *surfaced* and *unsurfaced* goals. The former set contains goals which are being actively managed (but not necessarily acted upon), the latter, goals which are currently entirely ignored by the management subsystem (and thus incapable of generating goal-directed system behaviour[12]). MINDER1 separates these sets using a

[9] For a more general comparison between Soar, BDI systems and other agent frameworks we direct the reader to the excellent comparison piece by Jones and Wray (2006)

[10] This view implicitly supports the research direction promoted by this article.

[11] This is perhaps true for systems of a mostly reactive nature, as the world almost always require a fast response. A comparable domain that has a slightly less frenetic pace is the artificial life domain of Tyrrell (as summarised in (Bryson, 2001)) in which a rodent must survive dangers, and make the most of the opportunities, of life on a simulated savannah.

[12] Although unable to generate goal-directed behaviour in the sense we are generally interested in, unsurfaced goals can still produce management behaviour which is ultimately serving the needs of the system.

threshold-based *attention filter*. If a goal's insistence value is greater than the value associated with the filter then it *surfaces* into the managed set, otherwise is remains unsurfaced (and unmanaged). Surfaced goals go through a series of state transitions before ultimately generating behaviour: first they are *scheduled* (which determines whether they are processed or suspended), if not suspended they are *expanded*. Expansion is a process of deciding whether the goal should be pursued (based on resource usage, current state etc.), planning the behaviour to achieve the goal (retrieving a reactive plan in this case), then executing the behaviour. MINDER1 can only expand a single goal at once (although Beaudoin's theory postulates the need for multiple concurrent active goals). Much like an intention structure in PRS-CL, a goal is expanded in place, so that it contains all of the information (plans etc.) necessary for resumption after suspension. In MINDER1 a goal can be suspended at any time by the management subsystem. Suspension means that a goal remains surfaced, but cannot influence behaviour (i.e. it is not the single goal currently undergoing expansion).

It is worth noting that although MINDER1's sets of managed goals are similar to those maintained by PRS-CL and the Soar BDI abstraction layer (both support a distinction between active and suspended intentions), MINDER1 provides both a finer-grained distinction between goal management states and additional support for processes which change the management states of goals. The surfaced/unsurfaced distinction allows MINDER1 to represent goals which have been generated by the system, but which are not yet insistent enough to engage the management processes. The other systems we have seen ballistically select all generated goals for processing. Additionally MINDER1 provides processes for deciding and scheduling which goals from the surfaced set should be actually acted upon. The other systems we have discussed assume that any generated goal should be directly acted upon when possible (although this notion of possibility – usually encoded as conditions for the firing of a behaviour – implicitly captures part of the deciding process).

The work of Wright and Beaudoin assumes that the management subsystem of an agent is resource limited. As such it is necessary to protect it from situations where it may reach the limit of its resources. Without being too specific about the computational resources available to a particular instance of a MINDER1 system, we can accept this assumption by considering that we require the management subsystem to stay responsive to new inputs and able to make timely decisions (rather than unresponsively processing for longer than any particular goal may be valid for). In MINDER1 it is the role of the attention filter to prevent the management systems from becoming overburdened. Wright models the resource constraints of the management layer by adopting a (soft) design constraint that only three goals should be in the surfaced set at any time. This is an arbitrarily selected value, but it serves to prove a point. Within MINDER1 it is the role of the *meta-management processes* to enforce this constraint via the attention filter. In brief, these processes perform the following operations: if the surfaced set contains less than three goals, the value of the threshold is reduced, thus allowing lower insistence goals to surface; if the surfaced set contains more than three goals, the value or the threshold is increased until lowest insistence goals "dive" out of the surfaced set. In conjunction with the ability of goal generators to increase and decrease insistence values (e.g., as minibot gets closer to a ditch), this gives rise to system with a dynamic set of active goals. The contents of this set should ideally reflect both the computational resources of the system and the current state of the world[13].

Although MINDER1 provides a powerful starting point for a goal management system, it also raises a number of questions (some of which are raised in the original work). The calculation of insistence for goals in the nursemaid domain is trivial given the nature of the drives involved. Generating importance and urgency information for less quantifiable drives will pose new problems. The relationship between the attention filter, management processes and meta-management processes will also require further thought in systems with richer deliberative processes and wider scope. Ideally management resource constraints should be characterised by something more informative than a number limit on set size. Future work on this must span two issues: the variety of goals, and the frequency of their generation, in a particular system and domain combination; and the nature of the deliberative and management processes available to a particular system (e.g. the constraints on a system using a deliberative planner may be quite different to those only using stored reactive plans). This latter consideration will also influence the expansion process. In MINDER1 deciding, scheduling and planning for a goal are all separate steps in the management subsystem. A system which integrated all of these processes (perhaps using techniques from deliberative, rather than reactive, planning) would be able to take decisions which are better informed about the interactions both between the behaviours necessary to achieve different goals, and between these behaviours and the available resources. This view is expanded in Section 6.

4.2.4. GRUE

Another architecture which augments a teleo-reactive planning system with a goal management system is the Goal and Resource Using architecturE (GRUE) of Gordon and Logan (2004, 2005). GRUE is similar to MINDER1 in that it has reactive goal generators, annotates its goals with priorities, and uses a threshold-based filter to discard low-priority goals. Whilst GRUE appears to be a simplified version of MINDER1 in some ways (e.g. it has a much smaller range of possible goal management states), it goes beyond MINDER1's contributions in others: it allows multiple goals to be acted on at once (whereas MINDER1 only allows a single goal to be acted upon), and provides a rich language for managing resource constraints (constraints which are largely ignored in the MINDER1 implementation). These two contributions are combined in the GRUE *arbitrator* process. GRUE attempts to achieve each active goal

[13] The dynamics of threshold changes and the management processes in MINDER1 can give rise to emergent perturbances. These can be used to provided an architecture-based, characterisation of a number of emotional states (Wright et al., 1996; Wright, 1997; Sloman et al., 2005).

with a separate reactive plan, each of which suggests an action to perform in each processing cycle. The arbitrator collects these actions and decides which ones should be allowed to execute (thus ultimately deciding which goals are acted upon). The decision is made by inspecting the resource requirements of the suggested actions. If there are any conflicts (e.g. two actions which use the same effector) then the actions associated with the lower priority goals are dropped. Whilst this demonstrates the power of combining importance measures (priority) with scheduling (resources assignment), it does also highlight the weakness of reactive approaches to goal-directed behaviour. In the event of repeated conflicts over resources, a GRUE agent has no mechanism for choosing alternative goals or plans, or reasoning in more detail about the causality of combinations of actions. If GRUE was extended with a meta-management process which configured the goal filter to ignores contentious goals it might work around such problems. A longer-term solution (in this instance and for the more general problem of multiple, interacting goals) is to move away from reactive planning approaches and investigate deliberative methods.

4.3. Deliberative and Hybrid Systems

In this section we review goal generation and management approaches taken by intelligent systems with deliberative capabilities.

4.3.1. Spartacus, MBA and EMIB

An architecture featuring explicit motivations was used to control Spartacus, a mobile robot which attended the AAAI '05 conference (Michaud et al., 2007). The architecture was the Motivated Behaviour Architecture (MBA) which is an instantiation and generalisation of EMIB architecture (which roughly stands for Emotion and Motivation for Intentional selection and configuration of Behaviour-producing modules) (Michaud, 2002). MBA features reactive behaviour-producing modules (BPMS, comparable to behaviours in the reactive systems reviewed previously) which are structured into reactive plans called *tasks* (Beaudry et al., 2005). MBA features *motivations* which are processes which can propose tasks into the *dynamic task workspace* (DTW). Motivations are thus the MBA equivalent of goal generators. The DTW is directly comparable to the intention structure in PRS-CL (see 4.2.1) in that it automatically accepts and stores all active tasks, where tasks can be considered a goal-plus-reactive-plan structure. Motivations exist in MBA for domain- and system-specific drives (e.g. sticking to the agenda of the conference, navigating to particular points in space) and more "instinctual" drives such as curiosity and survival.

MBA allows multiple tasks to be active at once. Goal management is performed by deciding which tasks are allowed to active which behaviours (similar to the approach to action arbitration taken in GRUE). This decision is based both on the activation values of motivations (which can be considered as an importance measures related to how well each motivation can satisfy its own need) and the system know-how (SNOW) built into MBA. From the literature SNOW appears to be a collection of domain- and system-specific rules for deciding which tasks should be allowed to run in the event of a conflict. Whilst this is not a solution we can carry forward into a new goal management system (as it is not generally applicable), it does demonstrate that the goal-management problem may require engineering solutions for particular cases.

The reason that MBA can be considered a reactive-deliberative hybrid approach is that it also uses a planner as motivation source. Given a goal, the planner creates a list of tasks which must be sequentially proposed to the DTW in order for the goal to be achieved. This highlights that goals will exist at multiple levels of abstraction in an intelligent system; what appears to be a goal at one level (i.e. a task in this case), may be a subgoal from a more abstract level.

4.3.2. DIARC

The notions of importance and urgency (as first identified by Beaudoin (1994)) are employed in the DIARC ("distributed integrated affect cognition and reflection") architecture (Scheutz and Schermerhorn, 2009). DIARC uses explicit goals which are generated from natural language input (Dzifcak et al., 2009) and stored in an *affective goal manager* (AGM) component[14]. Each goal is assigned an *affective task manager* (ATM) which performs action selection for its goal (which equates to reactive planning in this case). If two ATMs produce conflicting actions (where a conflict is identified as a violation in resource usage), then the goal with the highest priority is allowed to continue. Whilst this approach bares a strong resemblance to previous approaches (including GRUE and MBA), it is distinguished by a clear proposition for calculating priority values for goals. The AGM calculates a priority value from a goal's importance and urgency values, and assigns the priority to the respective ATM. In DIARC, importance is calculated by subtracting the estimated cost of achieving the goal from the product of the estimated benefits and an *affective evaluation* of the goal (where all quantities are real numbers). The affective evaluation attempts to capture the influence of the current state and past successes and failures on the ability of the system to achieve the estimated benefit. Positive experiences yield a more favourable evaluation for a particular goal, increasing its overall importance value. Negative experiences have the opposite effect. The goal's priority is calculated by scaling its importance value by urgency. Urgency is a measure of the time remaining to achieve the goal and is derived from the ratio between the elapsed time since the goal was created and the time the system expects to take to achieve the goal.

This method of prioritising goals is intended to overcome the problems inherent in taking a rational approach to decision making (which requires information that a robot typically

[14]Generating goals from natural language input is often a very different type of goal generation process than those considered in this article. It is an approach which will certainly become more prevalent as interactive robots gain a wider range of capabilities. Examples of systems that convert linguistic statements into deliberative action range from SHRDLU (Winograd, 1971) to DIARC and other recent interactive robot systems (Dzifcak et al., 2009; Brenner et al., 2007).

doesn't have access to). The combination of designer specified, or possibly learnt, values for costs and benefits, combined with the contextual and historical information captured in the affect evaluation allows the system to make reasonably good choices (i.e. satisficing (Zilberstein, 2008)) whilst being responsive to changes in the current situation (see (Scheutz and Schermerhorn, 2009) for examples of the emergent properties of this combination of values). This said, this method of goal selection can only make relatively uniformed management decisions; the purely numerical measures of cost and benefit are devoid of causal structure which could allow a deliberative decision-making process (not necessarily a optimal/rational one) to select goals based on a more informed judgment of expected future states (cf. "rationale" in Beaudoin's work, and also Sloman (To appear) on architecture-based motivation vs. reward-based motivation).

4.3.3. MADbot

One of the few planning approaches to treat goal generation as part of the planning problem (rather than as precursor to it) is the MADbot (Motivated And Goal Directed Robot) system of Coddington et al. (2005). The architecture is comparable to previously reviewed approaches such as MINDER1 and GRUE, but uses deliberative, rather than reactive, planning to generate behaviour. MADbot represents the world as a collection of state variables (much like the databases of beliefs used by other approaches). A drive in MADbot is effectively a constraint on a single state variable with an associated importance value[15]. For example, the drive "conserve-energy" will monitor a system's battery charge level, and attempt to keep it above a particular threshold. Where the reactive systems reviewed previously would encode this drive implicitly in the trigger conditions of a goal generator, MADbot explicitly records them as part of the system description. This has allowed them to experiment with two different approaches to goal generation: motivated goal generation, and motivations as resources (Coddington, 2007a,b).

In the motivated goal generation case, MADbot behaves much like a deliberative version of the previously reviewed systems: when a drive's threshold is violated, a goal to return its state variable to below the threshold is generated. The goal is then passed to the *goal arbitrator* which decides whether the goal should be added to the current goals the system is pursuing or not. This is the only real mention of explicit goal management in the MADbot work: they note that it is important, but leave it for future work. Newly accepted goals are added to a conjunction of previous accepted goals then passed to a planner. The plans are executed and the success or failure of actions (and their outcomes) is monitored by the system. If a plan is successfully executed then the state variables should all be returned to their below-threshold levels and the associated goals are removed from the arbitrator.

This approach was evaluated in a simulated Mars rover domain where four different drives were combined with user specified goals to control MADbot. The drives were used to maintain battery charge, disk space and localisation accuracy, whilst occasionally capturing images of the surrounding environment. The state variables associated with these drives vary as the system acts. For example, battery charge is reduced by driving to the places where images should be captured, and disk space is used up by capturing images. The initial design of the system was validated as MADbot was able to perform autonomously and maintain its drives in some situations, but some problems did arise. The most fundamental problem was that, because the planner is unaware of the system's drives, it often generated plans that violated resource constraints (e.g. plans were generated which could exhaust the system's battery charge). When constraints are violated (as the system executes one of these plans) the drives create new goals to recover (e.g. by charging the battery), but this raises an additional problem: MADbot cannot guarantee that these goals will be dealt with before the resource is exhausted. As the importance of a drive is not used when the goal is conjoined with the existing system goals, the resource constraint goal just becomes something else MADbot should do, rather than something it *must* do before it does other things that use up that resource. Coddington (2007a) states that this could be addressed by using the importance of the drives that a plan satisfies as a factor in a metric for evaluating candidate plans. This approach is comparable to the use of importance in MINDER1 and GRUE (although they additionally vary importance as a function of the distance of the state variable from the threshold). A different way to address this would be use MADbot's (currently under-used) goal arbitrator to suspend the current task goal and replace it with the goal to satisfy the drive. Although this would ensure that the resource constraint is not violated, it would not really solve the problem; the planner would still be creating plans for satisfying resource constraints separately to plans for its other goals.

To address these problems an alternative approach to goal generation was tried: the aforementioned motivations as resources approaches. Rather than use the drives to reactively generate new goals when thresholds are violated, these thresholds are instead modelled as resource preconditions for the actions in its planning domain for the Mars rover example. For example, moving the rover consumes (and thus requires) a certain amount of battery charge, and the level of battery charge can never be allowed to fall below a threshold. This approach to goal generation changed the behaviour of MADbot in the rover domain: the plans generated for a particular task goal took into account the resource constraints ahead of time (e.g. recharging the robot as appropriate), rather than waiting for a reactively generated correction goal. This meant that the resource constraints were never violated. This improvement came at the cost of two problems. First, the lack of reactive goal generators meant that the no autonomous behaviour was possible when an additional task goal was not specified. This meant that periodically relevant drives, such as capturing images, could not be triggered unless a plan was also being generated for another goal. The second problem was that the computational com-

[15] The MADbot literature actually uses the term "motive" to refer to a drive, but we will use the term "drive" to maintain consistency with our earlier discussions.

plexity of processing the resource constraints made the planning problem much, much harder than previously. Some standard resource-handling planners could not find plans, particularly when more than two drives were modelled in the domain.

The author's suggested solution to the complexity problem is to employ a hybrid approach to goal generation. Drives relating to hard resource constraints (i.e. those which should never be violated) should be modelled in the planning domain, whereas less critical drives should be encoded as reactive goal generators. This appears to be a sensible approach which will benefit from the strengths of both approaches (autonomy and simplicity; and adherence to resource limits respectively). However there is an additional problem with the motivations as resources approach which will have to be addressed in future work: not all drives can be easily encoded in this manner. One reason is that some drives cannot be captured simply as small number of numeric quantities (e.g. the drives Alfred had in Section 1.1 to fill gaps in its knowledge), although these may all turn out to be safely assignable to reactive goal generators (or at least implicitly modelled drives). Another reason is that in real-world robotics resource usage cannot be modelled so simplistically, due to many factors beyond the control of the planner. This may mean that the planner will need to employ probabilistic models of resource levels and usage, again increasing the complexity of the planning problem.

Despite these potential problems with the motivations as resources approach, the work on goal generation in MADbot highlights how a key problem in intelligent system design, the trade-off between reactive and deliberative processing, is apparent in the problems of goal generation and goal-directed behaviour. In an ideal world, some general purpose decision making mechanism could assess all of the possible combinations of goals a system could have *and* all of the plans for all of these possible combinations (including the consequences of resource usage etc.) before selecting the optimal goal and plan combination and acting on it. However there are many reasons, rooted in both the complexity of such decision making and the difficulties of modeling all behaviour this way, why this approach is just not possible in practice. The solution is to therefore carve off parts of the problem into separate specialist systems (reactive or deliberative) with less knowledge about the rest of the process. This makes the overall approach less powerful (and capable of less rational decision making), but more likely to make a decision in a reasonable amount of time (i.e. while its goals are still valid). One of the challenges in the design of a goal generation and management framework is how to reduce the overall problem into appropriately specialised subsystems without reducing the decision making power of the combined parts to such a degree that it is unable to make reasonable decisions. The work on MADbot has demonstrated that a naïve approach to combining reactive goal generation and planning (albeit an approach which does not make use of importance or scheduling information suggested by other authors) fails to make reasonable decisions in certain scenarios.

5. Summary of Approaches

We can now summarize the approaches we have seen to the generation and management of goals for an intelligent system. In this discussion we will conflate the different versions of a goal plus a plan for achieving it (e.g. a goal plus deliberation, an intention structure, and various other forms of reactive plans) and just refer to "goals", except where the difference is salient. We shall also adopt the following terminology to describe the different sets of goals that a system can have: *managed goals* are those goals which have been accepted into a system's goal management process (equivalent to "surfaced" in MINDER1 described in Section 4.2.3); *active goals* are those goals which are causing goal-directed behaviour to be generated (this term is used by MINDER1 from Section 4.2.3, the Soar BDI layer from Section 4.2.2, and implicitly by other approaches); *suspended goals* are managed goals which are not active (again used by MINDER1, Soar etc.); and *unsurfaced goals* are those goals which have been generated, but not accepted into the management process yet (this term is taken directly from MINDER1, as no other system has this facility).

Where a system generated its own goals autonomously (rather than having them imposed by another system) it almost always did do reactively, and with no reference to the mechanisms used to subsequently expand the system's goals. Although this approach is generally effective where deployed (e.g. in PRS-CL from Section 4.2.1, Soar, MINDER1, GRUE from Section 4.2.4, MBA from Section 4.3.1 etc.), this is often because the scenario is amenable to decomposition into a multi-stage process (i.e. goal generation, goal expansion if required, arbitration then execution). The work on MADbot (presented in Section 4.3.3) demonstrates that not all problems can be tackled in this way. In particular problems where drives exist to maintain interacting resources may be better modelled as part of a planning process, although this increases the computational complexity of the deliberation task.

All but one of the approaches we have seen allow their systems to have multiple goals active at once (we shall refer to these as *goal conjunctions*). MINDER1, the approach that can only have a single active goal at a time, was also not intended to have this limitation. Given the ability to tackle goal conjunctions, there are two problems which must be tackled: which goals should be conjoined; and which goals can be achieved when conjoined. The former issue is one of determining priorities or preferences over all managed goals. The latter is one of resolving conflicts between the actions required to achieve the active (conjoined) goals.

The work we have surveyed typically takes one of four different approaches to selecting which goals should be conjoined. In BOD (from Section 4.1), DIARC (from Section 4.3.2) and MBA, all goals are activated without consideration and thus conjoined by default. In PRS-CL and Soar goals are also activated without consideration, but can be subsequently suspended (providing a choice of possible conjunctions). In MADbot goals pass through an arbitration step before coming active, i.e. they start suspended but can later become active (again providing a choice of possible conjunctions). MINDER1 and GRUE

go one step further by using an additional filter to determine which goals can become managed before then being activated. MINDER1 motivates the use of a filter by claiming resource limits in the management layer. Although the implementation of this approach does not support this, the work on motivation as resources in MADbot demonstrates the problems of putting too great a load on deliberative systems.

Once a system has a conjunction of active goals it must act to achieve them. However, actions that help to achieve one goal may conflict with actions that help to achieve another. The existing literature mostly deals with one type of conflict, resource usage, but other types of conflict can exist too (notably the actions for one goal undoing preconditions established by actions for another). In almost all cases resource conflicts are dealt with by inspecting the goals which the actions achieve, then awarding the resource (thus the right to execute) to the action for goal with the greatest priority. The existing systems assign priorities in various ways, from the fixed priority schemes or MINDER1 and BOD, to the activation-based approach of MBA, to the more dynamic, affect-based, approach of DIARC. Regardless of implementation this scheme amounts to using a single partial ordering to decide which goal is actually active during a conflict.

An alternative to resolving conflicts by ordering is demonstrated by systems with deliberative capabilities. MADbot and MBA (in some contexts), using planning, and to some extent Soar, using its impasse mechanisms, can choose selections and orderings of actions *in advance of acting* which do not have conflicting demands (providing the important resources etc. can be modelled in their representations). This again highlights the benefits of deliberation in domains where it is feasible: reactive systems will require arbitration mechanisms as their behaviours are independently generated and thus can conflict after the fact; deliberatively systems can reason about and avoid conflicts before the fact. Regardless of the overall approach, most of the surveyed approaches rely on representations of resources and priorities. Priorities are important beyond conflict resolution as they can also be used as preferences on goals in a deliberative system (e.g. (Brafman and Chernyavsky, 2005)), and as part of the process for determining which goals should be managed and active (as in MINDER1). This is because some needs may be inherently or contextually more important to a system than others, even if no conflict is evident.

An important issue which we only saw addressed by PRS-CL and Soar (although there is much related work in the BDI field) is goal maintenance or reconsideration. This is essentially the problem of deciding when a goal should be suspended or even dropped entirely. Both systems use variations of logical entailment to determine whether goals are still necessary. Beaudoin's work argues that goals should represent their dependencies (via beliefs and rational (Beaudoin, 1994, p47)) to support this kind of reasoning, but this was not implemented in MINDER1.

Finally, the issue of the temporal and behavioural dependencies of goals was only discussed with reference to scheduling in the theoretical work of Beaudoin and in the implementation of urgency in DIARC. Deciding *when* (not whether) a goal should become active, or some action should be executed, and then scheduling that to occur as appropriate (e.g. in two hours time, or after this goal is achieved, or the next time I'm in a particular room) may reduce the load on a goal management system. One, computationally undesirable, alternative to this is for all known future goals to be managed when generated and for the management process to keep checking them (i.e. polling) for activation until the correct time occurs. Urgency in DIARC and MINDER1 implicitly encodes a solution to this as the goal generators can make goals more urgent when appropriate in order to make it more likely that they will be activated.

6. A Design for a Motive Management Framework

Given the preceding summary of the literature, and the requirements we set out previously, we can start to sketch out an early design, or at least a list of desiderata, for a future architecture for managing goal-directed behaviour. Our intention is to develop something along these lines that will allow us to explore the trade-offs in design space for these sorts of mechanisms.

Although most of the work we surveyed used reactive planning approaches to generate behaviour, we will follow the work on MADbot (and many other intelligent robotic systems, e.g. (Bonasso et al., 1997; Hawes et al., 2007)) and use a planner, coupled with reactive behaviours, to determine future actions. The advantage of this approach is that interactions between active goals and conflicts over resources can be resolved through reasoning in advance of acting, rather than through post-hoc arbitration. To work effectively in an architecture for goal management for a real-world robot, any planner we use must satisfy the following criteria. First and foremost it must be a *continual* planner, capable of interleaving planning with execution and monitoring, and replanning when the world or its goals are vary from its expectations. Continual planning has been used to overcome the dynamism and partial observability which exists in robotic domains Brenner and Nebel (To appear) and in situations where an external source can change the goals of the system Knight et al. (2001). The planner ideally should be able to model resource constraints, which would allow us to model motivations as resources (where appropriate). An additional desirable feature would be the ability to behave as an oversubscription planner Menkes van den Briel and Kambhampati (2004); Brafman and Chernyavsky (2005). An oversubscription planner is a planner which can determine which subset of goals in a goal conjunction can actually be achieved together, and, by using cost measures (like importance and urgency), which goals should be pursued together. In essence this would place the onus of activating goals (i.e. selecting which ones should be acted on) on the planner, rather than an additional mechanisms.

Although the suggested planning approach would address many of the sub-problems we have noted in this article, there are two caveats. First, we know of no planner which currently meets all of the listed requirements. We expect developing one, or altering an existing one, to be a non-trivial exercise. Furthermore, as the MADbot work demonstrates, by placing additional responsibilities on a planner you make its task harder. Given the

combinatorial nature of planning problems, it is possible that the problems we wish to tackle may be too hard to solve in a single system. An alternative approach is to develop a planning architecture for goal management, e.g. using an oversubscription planner to determine which goals should be active before passing the active conjunction on to a continual planner (resource scheduling could also occur after plans are generated).

In addition to the hypothesised planner we require goal generators (reactive or otherwise) which are independent of the planing process. These will ensure that autonomous behaviour can be generated as opportunities become available (i.e asynchronously with respect to planning), and encode drives that cannot be reasoned about in a single planning system (e.g. drives that require access to information not modelled in the planning domain). To capture the rich variety of goals an intelligent system can have we may need to extend the state descriptions of planning goals with temporal operators (as in in PRS-CL (Georgeff and Ingrand, 1989a) and NML1 (Beaudoin, 1994, p45)), although this temporal information could be captured implicitly in the nature of the goal generators themselves (understanding the trade-offs here is one of many open problems). It may be desirable to encode homeostatic drives as resource limits in the planning models (as in MADbot), although we may also want to develop a reactive alarm system for the cases where deliberation does not satisfy safety- or mission-critical drives Sloman (1998)).

As required by many of the approaches surveyed above, goal generators will be required to annotate their goals with importance and urgency measures. These measures will be used in various places in the system. However, it is not clear how these values should be generated or represented. Initially they can be fixed or contextually generated from drives (cf. the distance between a minibot and a ditch in Section 4.2.3), and use either numeric values or partial orderings (e.g. the drive to recharge is more important the drive to clean the floor). Ultimately we may need additional, more powerful representations, particularly for scheduling various tasks based on urgency (which appears to be an aspect of the problem of intelligent behaviour with is overlooked in the literature).

The design should allow for active and suspended goal sets (as described in the previous section). The process of goal management (i.e. moving goals between these two set) can be (implicitly) performed by an oversubscription planner, or, if this proves too complex initially, by inspecting importance and urgency measures (although there are potential flaws in this approach). We will also adopt the idea of a pre-management filtering step from the NML1/MINDER1 and GRUE architectures. As our proposed approach of goal management through planning is potentially computationally demanding (cf. Section 4.3.3), filtering may protect the management system from become overburdened with decisions (and thus unresponsive to potentially important changes in the system's goals and state). On the other hand, if goal management does not prove to be too demanding, there may be less of a justification for an additional filtering step.

Although this design sketch gives us a starting point for future work which builds on the experience of others, it presents as least as many questions as it attempts to answer. These represent issues which we will explore, using the traditional methods of AI, via a more detailed requirements analysis for particular domains, an implementation of the system, and subsequent evaluations (i.e. we will follow a design-based methodology Sloman (1992)). Open questions include the following: what mechanisms should be used for removing goals from the unsurfaced or managed sets?; how often should goal-management decisions be taken? (a dynamically changing world could render decisions incorrect without the system acting); how should longer-term behaviour be modelled? (goal-generation and planning probably cannot account for life-long behaviour in a robot, cf. the use of a planner as motive source in Section 4.3.1); and how can deliberation and reactivity interact in way that plays to their strengths, rather than constraining one approach with the limits of the other.

7. Conclusion

In this article we presented arguments for the development of a motive management frame for an intelligent system. Building on previous studies we identified the required parts of such a framework, and particularly focused on processes which can generate goals and select goals to become the focus of goal-directed behaviour. Given this focus we reviewed the literature on architectures for intelligent systems and identified a number of trends in existing work towards addressing these problems. From this we proposed an outline design for a motive management framework. The design prefers deliberation to reactivity where possible, but freely admits the limitations of this approach. It also includes mechanisms from the literature (e.g. an attention filter) in order to explore the design trade-offs they present. In future work we will implement this architecture and evaluate it in both robotic domains and in simulated worlds (as both evaluation domains present different problems and opportunities to any proposed design).

Acknowledgments

The author would like to thank Aaron Sloman for discussions about motivation, attention and the definition of control states, and Charles Gretton, Richard Dearden, and Michael Brenner for discussions about the relationship between motivation and planning in goal-directed behaviour. This work was supported by the EU FP7 ICT Cognitive Systems Integrated Project "CogX". For more information see `http://cogx.eu`.

References

Agre, P. E., Chapman, D., 1987. Pengi: An implementation of a theory of activity. In: Proceedings of The Sixth National Conference on Artificial Intelligence. pp. 268–272.

Arkin, R. C., 1998. Behavior-Based Robotics. MIT Press.

Beaudoin, L. P., 1994. Goal processing in autonomous agents. Ph.D. thesis, School of Computer Science, The University of Birmingham.

Beaudry, E., Brosseau, Y., Côté, C., Raïevsky, C., Létourneau, D., Kabanza, F., Michaud, F., 2005. Reactive planning in a motivated behavioral architecture. In: Veloso, M. M., Kambhampati, S. (Eds.), Proceedings of the Twentieth National Conference on Artificial Intelligence. AAAI Press / The MIT Press, pp. 1242–1249.

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

Brafman, R. I., Chernyavsky, Y., June 2005. Planning with goal preferences and constraints. In: Biundo, S., Myers, K. L., Rajan, K. (Eds.), Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS 2005). AAAI, pp. 182–191.

Bratman, M. E., 1987. Intention, Plans, and Practical Reason. Harvard University Press, Cambridge, MA.

Bratman, M. E., Israel, D. J., Pollack, M. E., 1988. Plans and resource-bounded practical reasoning. Computational Intelligence 4, 349–355.

Brenner, M., Hawes, N., Kelleher, J., Wyatt, J., January 2007. Mediating between qualitative and quantitative representations for task-orientated human-robot interaction. In: Veloso, M. M. (Ed.), IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence. pp. 2072–2077.

Brenner, M., Nebel, B., To appear. Continual planning and acting in dynamic multiagent environments. Journal of Autonomous Agents and Multiagent Systems.

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

Bryson, J. J., June 2001. Intelligence by design: Principles of modularity and coordination for engineering complex adaptive agents. Ph.D. thesis, MIT, Department of EECS, Cambridge, MA.

Bryson, J. J., 2003. The Behavior-Oriented Design of modular agent intelligence. In: Kowalszyk, R., Müller, J. P., Tianfield, H., Unland, R. (Eds.), Agent Technologies, Infrastructures, Tools, and Applications for e-Services. Springer, Berlin, pp. 61–76.

Coddington, A., 2007a. Motivations as a meta-level component for constraining goal generation. In: Proceedings of the First International Workshop on Metareasoning in Agent-Based Systems. pp. 16–30.

Coddington, A. M., 2007b. Integrating motivations with planning. In: Proceedings of the 6th International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS07). pp. 850–852.

Coddington, A. M., Fox, M., Gough, J., Long, D., Serina, I., 2005. Madbot: A motivated and goal directed robot. In: Veloso, M. M., Kambhampati, S. (Eds.), Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference. AAAI Press / The MIT Press, Pittsburgh, Pennsylvania, USA, pp. 1680–1681.

Dennett, D. C., 1978. Brainstorms: Philosophical Essays on Mind and Psychology. MIT Press, Cambridge, MA.

Dzifcak, J., Scheutz, M., Baral, C., , Schermerhorn, P., May 2009. What to do and how to do it: Translating natural language directives into temporal and dynamic logic representation for goal management and action execution. In: Proceedings of the 2009 IEEE International Conference on Robotics and Automation (ICRA '09). Kobe, Japan.

Eyerich, P., Mattmüller, R., Röger, G., 2009. Using the context-enhanced additive heuristic for temporal and numeric planning. In: Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS 2009).

Firby, R. J., 1987. An investigation into reactive planning in complex domains. In: Proceedings of The Sixth National Conference on Artificial Intelligence. pp. 202–206.

Franklin, S., Graesser, A., 1997. Is it an agent, or just a program?: A taxonomy for autonomous agents. In: ECAI '96: Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages. Springer-Verlag, London, UK, pp. 21–35.

Gat, E., 1998. Three-layer architectures. In: Artificial intelligence and mobile robots: case studies of successful robot systems. MIT Press, Cambridge, MA, USA, pp. 195–210.

Georgeff, M., Pell, B., Pollack, M., Tambe, M., Wooldridge, M., 1999. The belief-desire-intention model of agency. In: Intelligent Agents V: Agent Theories, Architectures, and Languages. 5th International Workshop, ATAL'98. Proceedings. Springer-Verlag, pp. 1–10.

Georgeff, M. P., Ingrand, F. F., 1989a. Decision-making in an embedded reasoning system. In: Proceedings of the 11th International Joint Conference on Artificial Intelligence. pp. 972–978.

Georgeff, M. P., Ingrand, F. F., July 1989b. Monitoring and control of spacecraft systems using procedural reasoning. In: Workshop of the Space Operations-Automation and Robotics. Houston, Texas.

Gordon, E., Logan, B., July 2004. Game over: You have been beaten by a GRUE. In: Fu, D., Henke, S., Orkin, J. (Eds.), Challenges in Game Artificial Intelligence: Papers from the 2004 AAAI Workshop. AAAI Press, pp. 16–21.

Gordon, E., Logan, B., 2005. Managing goals and resources in dynamic environments. In: Davis, D. N. (Ed.), Visions of Mind: Architectures for Cognition and Affect. Idea Group, Ch. 11, pp. 225–253.

Hawes, N., Sloman, A., Wyatt, J., Zillich, M., Jacobsson, H., Kruijff, G.-J., Brenner, M., Berginc, G., Skočaj, D., July 2007. Towards an integrated robot with multiple cognitive functions. In: Holte, R. C., Howe, A. (Eds.), Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI 2008). AAAI Press, Vancouver, Canada, pp. 1548 – 1553.

Jennings, N. R., Cohn, A., Fox, M., Long, D., Luck, M., Michaelides, D., Munroe, S., Weal, M., 2006. Interaction, planning and motivation. In: Morris, R., Taressenko, L., Kenward, M. (Eds.), Cognitive systems: Information processing meets brain science. Elsevier, pp. 163–188.

Jones, R. M., Wray, R. E., 2006. Comparative analysis of frameworks for knowledge-intensive intelligent agents. AI Mag. 27 (2), 57–70.

Knight, R., Rabideau, G., Chien, S., Engelhardt, B., Sherwood, R., 2001. Casper: Space exploration through continuous planning. IEEE Intelligent Systems 16 (5), 70–75.

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

Lisse, S. A., Wray, R. E., Huber, M. J., March 2007. Beyond the ad-hoc and the impractically formal: Lessons from the implementation of formalisms of intention. In: Working Notes of the AAAI Spring Symposium on Intentions in Intelligent Agents.

Maes, P., 1991. The agent network architecture (ana). SIGART Bull. 2 (4), 115–120.

Menkes van den Briel, R. S. N., Kambhampati, S., 2004. Over-subscription in planning: A partial satisfaction problem. In: ICAPS 2004 Workshop on Integrating Planning into Scheduling.

Michaud, F., 2002. EMIB – computational architecture based on emotion and motivation for intentional selection and configuration of behaviour-producing modules. Cognitive Science Quarterlla 2 (3/4).

Michaud, F., Côté, C., Létourneau, D., Brosseau, Y., Valin, J. M., Beaudry, E., Raïevsky, C., Ponchon, A., Moisan, P., Lepage, P., Morin, Y., Gagnon, F., Giguère, P., Roux, M. A., Caron, S., Frenette, P., Kabanza, F., 2007. Spartacus attending the 2005 aaai conference. Auton. Robots 22 (4), 369–383.

Myers, K. L., 1996. A procedural knowledge approach to task-level control. In: Proceedings of the Third International Conference on AI Planning Systems. AAAI Press, pp. 158–165.

Nilsson, N. J., 1994. Teleo-reactive programs for agent control. Journal of Artificial Intelligence Research 1, 139–158.

Peltason, J., Siepmann, F. H. K., Spexard, T. P., Wrede, B., Hanheide, M., Topp, E. A., 2009. Mixed-initiative in human augmented mapping. In: International Conference on Robotics and Automation (ICRA).

Scheutz, M., Schermerhorn, P., 2009. Affective goal and task selection for social robots. In: Vallverdú, J., Casacuberta, D. (Eds.), The Handbook of Research on Synthetic Emotions and Sociable Robotics. Information Science Reference.

Sloman, A., 1992. Prolegomena to a theory of communication and affect. In: Ortony, A., Slack, J., Stock, O. (Eds.), Communication from an Artificial Intelligence Perspective: Theoretical and Applied Issues. Springer, Berlin, Heidelberg, pp. 229–260.

Sloman, A., Oct 1998. Damasio, descartes, alarms and meta-management. In: IEEE International Conference on Systems, Man, and Cybernetics. Vol. 3. pp. 2652–2657 vol.3.

Sloman, A., 2002. Architecture-based conceptions of mind. In: Gärdenfors, P., Kijania-Placek, K., Woleński, J. (Eds.), In the Scope of Logic, Methodology, and Philosophy of Science (Vol II). Vol. 316 of Synthese Library. Kluwer, Dordrecht, pp. 403–427.

Sloman, A., 2003. The Cognition and Affect Project: Architectures, Architecture-Schemas, And The New Science of Mind. Tech. rep., School of Computer Science, University of Birmingham.

Sloman, A., May 2006. Requirements for a Fully Deliberative Architecture (Or component of an architecture). Research Note COSY-DP-0604, School of Computer Science, University of Birmingham, Birmingham, UK.

Sloman, A., To appear. Architecture-Based Motivation vs Reward-Based Motivation. Newsletter on Philosophy and ComputersHttp://www.cs.bham.ac.uk/research/projects/cogaff/architecture-based-motivation.pdf.

Sloman, A., Chrisley, R., Scheutz, M., 2005. The architectural basis of affective states and processes. In: Fellous, J.-M., Arbib, M. A. (Eds.), Who Needs Emotions?: The Brain Meets the Machine. Oxford University Press, pp. 203–244.

Tyrrell, T., 1993. Computational mechanisms for action selection. Ph.D. thesis, University of Edinburgh.

Winograd, T., 1971. Procedures as a representation for data in a computer program for understanding natural language. Tech. Rep. 235, MIT AI.

Wright, I., 1997. Emotional agents. Ph.D. thesis, School of Computer Science, The University of Birmingham.

Wright, I., Sloman, A., Beaudoin, L., 1996. Towards a design-based analysis of emotional episodes. Philosophy Psychiatry and Psychology 3 (2), 101–126.

Zender, H., Jensfelt, P., Óscar Martínez Mozos, Kruijff, G.-J. M., Burgard, W., July 2007. An integrated robotic system for spatial understanding and situated interaction in indoor environments. In: Proc. of the Twenty-Second Conference on Artificial Intelligence (AAAI-07). Vancouver, British Columbia, Canada, pp. 1584–1589.

Zilberstein, S., 2008. Metareasoning and bounded rationality. In: Cox, M. T., Raja, A. (Eds.), Papers from the 2008 AAAI Workshop on Metareasoning. AAAI Press, Menlo Park, CA, pp. 55–59.