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

DR 1.3:
Architectures and representations for introspection and motive management in a robot

The CogX Consortium

⟨cogx@cs.bham.ac.uk⟩

Due date of deliverable: July 31 2011
Actual submission date: July 31 2011
Lead partner: BHAM
Revision: final
Dissemination level: PU

This deliverable reports on our current work on architectural issues for generating robot behaviour, with a focus on motive management and the trade-offs inherent in the choice of knowledge representations. On the former issue we present our current work on *goal activation*, and discuss the problems of creating mechanisms and metrics for deciding which subset of all possible goals a system should pursue. On the issue of knowledge representation we present results from a study into the trade-offs apparent between integrated systems that use deterministic knowledge and systems that use probabilistic knowledge.

1 Introduction **4**

2 Motive Management **4**
2.1 Goal Activation 6
2.2 Proposed Approach 7

3 Representations for Cognitive Systems **10**

References **11**

A Annexes **14**
A.1 Exploiting Probabilistic Knowledge under Uncertain Sensing for Efficient Robot Behaviour 14
A.2 Episodic-Like Memory for Cognitive Robots 15

Executive Summary

In the reporting period we have developed a new approach to *goal activation* (i.e. deciding which goals to actively pursue from a set of possible goals). This forms part of our overall theory for motive management. The approach treats the activation as a net-benefit planning problem, but also integrates a preference system based on a hierarchy of drives. This addition is necessary to allow the system to reason about the interactions of different types of goals, particularly when no meaningful comparison metrics can be created for these different types. For more information see Section 2.

We have also developed an Episodic-Like Memory system capable of representing the experiences of a cognitive robot in space and time. This work has been demonstrated on a Dora-like mobile robot. For more information see Annex A.2.

Finally, we have been exploring the trade-offs apparent between different architectural and representational approaches for generating robot behaviour. Specifically we have built and evaluated variations of the Dora system that can use, or not use, structural knowledge about its environment, and use, or not use probabilistic representations and algorithms to augment this knowledge. This has allowed us to compare a range of different robot systems on the task of finding an object in a real-world environment. For more information see Section 3 and Annex A.1.

Role of Introspection and Motive Management in CogX

Introspection and motive management are both necessary for any complex, autonomous robot required to robustly perform tasks for humans in normal environment. These abilities are also central to the proposed approach of CogX where systems must introspect in order to understand and explain their successes and failures when performing tasks, and must be capable of management multiple competing drives and their associated goals.

Contribution to the CogX scenarios and prototypes

The motivation system developed in WP1 is currently deployed in both the Dora and George demonstrators. The investigations into architectures and representations for robots have been based on the current Dora demonstrator system.

1 Introduction

The architecture and representations used to create a system define its capabilities, and thus shape its strengths and weaknesses. Creating systems that are able to autonomously self-extend during task-driven behaviour is requiring us to investigate a range of mechanisms and formalisms for generating behaviour, and for representing and reasoning about the knowledge used to inform this behaviour. Our research is uncovering fundamental issues apparent in the design of cognitive systems that must operate in the real world. This deliverable reports on two of these issues. The first is related to motive management, the problem of deciding which goals a system should tackle next. We have found that designing mechanisms to choose which subset of possible goals should be pursued holds a number of interesting challenges, particularly with reference to finding metrics to support comparisons of difference types of goals. This work is summarised in Section 2. The second issue is a fundamental one: the representation of knowledge in a cognitive system. This is particularly relevant to CogX as we are approaching introspection by tackling the issue of how what a system knows, and how this is represented, affects its ability to perform tasks in the real world. We have been exploring the advantages and disadvantages of using probabilistic representations and reasoning methods in the majority of an integrated system. Whilst our results show that in most circumstances systems that leverage probabilities perform as well as, or better than, purely deterministic systems, there are many challenges evident in building such systems (from integrating the various parts to solving problems in the much larger state spaces they present). Our work on this topic is summarised in Section 3.

2 Motive Management

One of the main responsibilities of WP1 is to research architectural mechanisms for what we called in the proposal (and thus the title of this deliverable) *motive management*. In summary, this is the problem of generating and managing goals for an integrated system. In the first year of the project we surveyed the literature and produced the initial architecture design pictured in Figure 1. This architecture features *goal generators* which are domain or task specific components which generate goals describing desired future states for the system. These goals are referred to as *unsurfaced* until they pass (surface) through an *attention filter* to be considered by a collection of management processes. These management processes are responsible for deciding which of these *surfaced* goals should be actively pursued by the system. We refer to the goals which are being pursued as *activated*.

Our literature survey and initial design was originally presented as DR.1.1 and recently published in AIJ [6]. We have implemented our architecture de-

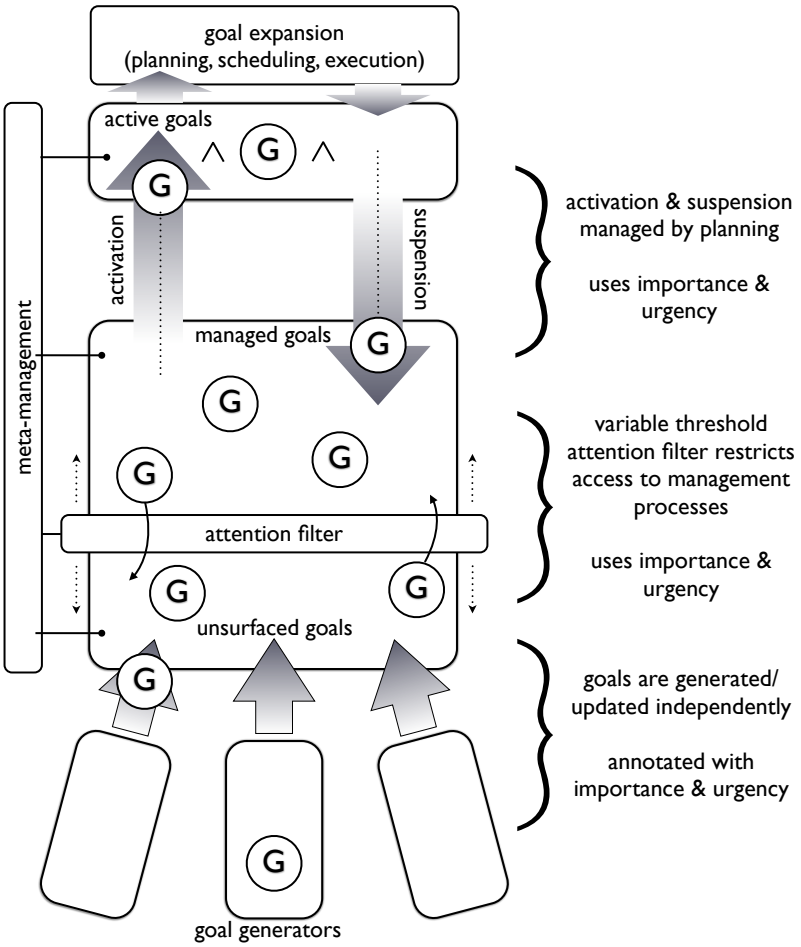

Figure 1: A visualisation of the design for a motive management framework.

sign and integrated the same system into both the Dora and George demonstrators. This has resulted in both an experimental validation of the design (see [11, 5], presented in DR.1.2 and DR.7.1 respectively) and contributions to novel system behaviour (e.g. [7, 9], both presented in DR.7.2). This year we have had limited time to spend on advancing the core theory for the motivation framework, focusing instead on realisation and testing of the existing framework within the demonstrator systems. What time we have had has been spent considering one element of the motivation framework in further detail. This element is *goal activation*, i.e. the process by which surfaced goals are selected to be the target of execution.

2.1 Goal Activation

We are interested in systems that can pursue multiple goals at the same time, i.e. have more than one active goal (and other surfaced, yet inactive ones). For this to be possible goals must be passed to the system's planner as a conjunction so that the planner can determine the best way of satisfying them all together. In fact it is only the planner that can determine whether any two goals can actually be achieved together, or whether they conflict in some way. This therefore implies that *planning should be considered a core element of goal activation*, rather than just a component waiting to receive active goals. This approach is considered explicitly when planning is posed as a *net-benefit* problem. In this formulation, the planner is given a list of goals, plus a cost for not achieving each one (referred to as *cost-to-drop*), and must generate a plan to achieve the most valuable subset of the goals at the lowest possible cost. Given this formulation, a naïve approach would be to treat the entire goal management process as a net-benefit planning process. This would imply passing a net-benefit planner the list of all surfaced goals and defining the activated goals (implicitly) as those which the planner generates a plan to achieve. This is an approach we experimented with in the Year 2 Dora system, using the Temporal Fast Downward planner [3] from WP4, which is capable of solving net-benefit problems. The remainder of this section is built on the insights that this experience provided.

One reason why treating goal activation purely as a net-benefit problem is an oversimplification is its reliance on a cost-to-drop value for goals. The CogX systems are capable of achieving a wide range of types of goals. For all goals to be treated fairly by a net-benefit planner, a system must be devised for assigning every goal a cost-to-drop value that, when compared to the value for other goals, reflects the relative importance of the system trying to achieve either goal right now. In our experience, producing these cost-to-drop values is a complex process and often values need to be redetermined when the system changes or is deployed in a new environment. For an example of this, consider the Dora year 1 system [7]. This system could produce goals to fill gaps in its knowledge via the exploration a new areas of space (one goal for every new bit of unexplored space) and via room categorisation (one goal to for every room). The values of exploration goals were directly determined by the amount of new known space that exploration might yield. If Dora was only required to consider exploration goals then this would be a fair metric to use for their comparison. However, Dora had to decide whether to achieve exploration goals or room categorisation goals. Room categorisation goals were assigned values based on the size of the target room. As a room is almost always larger than a single place (real or hypothetical), with all other things being equal Dora would always choose to categorise a room rather than explore.

Whilst the comparison of exploration and categorisation goals looks su-

perficially valid as their values can at least be expressed in the same units (metres squared), there is a deeper problem here. The reason a system decides to pursue (activate) a goal should be based on the expected changes in system state that achieving the goal might yield. In CogX we have been considering this in terms of the information gain (in an informal sense) expected from a goal. This is an approach that aligns well with the idea of filling knowledge gaps. Achieving a categorisation goal yields information about the category of a room (or a distribution over all categories). Achieving an exploration goal yields information about the extent of space, including perhaps whether a previously unseen room exists. These are qualitatively different types of information, with both influencing possible future behaviour in different ways. So, even though we consider both types of goals similar as they allow Dora to fill gaps in its knowledge (they are arguably derived from the same drive – to self-extend), there is no simple way of directly comparing them. This is the first challenge of goal activation: *finding approaches to support the comparison of goals of different types.*

An additional problem faced by a goal activation system is that goals may also be valid for different time periods. When Dora is just exploring, the order in which parts of space are explored or rooms are categorised makes little difference to the final outcome. Different orderings may increase or decrease the time it takes to build a complete model of the environment, but given enough time this will be achieved anyway [7]. However, when Dora is given a task such as reporting the position of an object (as in the Year 2 and Year 3 systems), it is safe to assume that this task is not free of time constraints (as a hungry human will usually only wait so long for their cornflakes). The easiest way to ensure that a goal given by a human is achieved as quickly as possible is to make this the system's only goal. However, this prevents the system from being able to pursue more than one goal at once, and rules out achieving goals provided by multiple humans (where plans for each goal may have overlapping plans), or interleaving human-provided goals with internally generated goals such as exploration (although if internal goals are task-related, they can be handled as subgoals – see [1], included in DR.4.3, for an example). This latter combination is particularly relevant to CogX, where we would like our systems to have the flexibility to choose to self-extend even when operating under user-provided instructions. However, a useful system should not self-extend to the exclusion of achieving human-provided goals. Thus the second challenge of goal activation is to design *a system capable of respecting the different constraints that may apply to, or between goals*, from time constraints to priority orderings.

2.2 Proposed Approach

We do not yet have a working system capable of meeting the two challenges described above. However, we have produced the following proposal which

we aim to implement and evaluate in the coming year.

Based on our work on the integrated systems and our previous literature review, we have decided to classify all of the underlying motivations of the CogX systems into three different drives. These are, in order of decreasing priority:

1. Keep resource levels within acceptable bounds (homeostasis), e.g. battery level. This allows the system to keep operating.
2. Do what humans ask, within an appropriate time.
3. Perform self-extension. We assume self-extension exists to improve the system's ability to satisfy the preceding two drives.

Each drive will be realised in a system as a collection of goal generators. The system's drives provide an ordering over the sets of goals produced by the goal generators for each drive. We will treat this ordering as encoding preferences for choices when conflicts occur: the system must not drop a goal from a more preferred drive in order to achieve one from a less preferred drive. In addition a drive can dictate the activation semantics of a goal which it has given rise to. Some drives (human tasks, homeostasis) will state that any goal derived from it must be activated if surfaced if this is the highest preference drive. Other drives will state that their goals can be active or not even when they are the highest preference (this is to deal with self-extension cases).

A further assumption we must make is that all goals can have deadlines attached. These may be provided at generation time, or inferred by other mechanisms (e.g. default or learnt deadlines). This assumption is necessary because without any constraints on when any goals must be achieved, there is no need to choose some collection above some other collection. We allow for goals that have no deadlines, but we must also allow for goals that do (in particular to support the human-provided tasks).

Once we have deadlines we can consider the activation problem as a decision about which *other goals* should be achieved in the gap between the time it will take the system to actually achieve its highest priority goals and the deadlines associated with them. When making this decision we assume that the system has decided to achieve all the goals it can from drive with the highest priority that has surfaced goals. It can now try to (opportunistically) include as many goals as it can from the next highest priority set of goals, provided that no deadlines are violated from previously added goals. When doing this it must be able to select some subset of the goals at this next priority level (assuming that there is not enough time to achieve all of them). To do this we will assume that goals from the same drive can be directly compared using some (dynamic, contextual) ordering or reward scheme. As we saw above, it is not easy to generate such a scheme,

but it is at least more meaningful to compare goals from the same drive, than comparing goals across drives. This proposed process of including goals from lower priority drives can be repeated for all drives, provided no deadlines are violated.

Following this approach, a complete set of activated goals can be built up by solving a series of net-benefit planning problems. To ensure that goals activated by a previous planning session are not rejected by a later session, we propose that all previous goals are turned into hard goals when additional (lower priority) goals are added to the goal conjunction. The algorithm we envisage implementing to achieve this in our motivation framework is presented below. In this we use G_p to refer to a set of goals with a drive-derived priority p, where higher values of p represent higher priorities. Each member of G_p is a tuple $\langle g, d, c \rangle$ where g is the goal state, d is the deadline and c is the cost-to-drop for this goal (where the value of infinity is used to represent a hard goal).

1. Take the highest preference drive goal set G_p, setting all c to infinity (i.e hard goals), unless the drive states otherwise.
2. Find plan P_p which achieves all goals in G_p within their d values.
3. If no such plan exists, fail (or trigger conflict resolution mechanism).
4. Take the next highest drive-preference goal set G_{p-1} (taking all c values as provided).
5. Set all c values in G_p to infinity if they weren't previously
6. Find a plan P_{p-1} for the net-benefit problem $G_p \wedge G_{p-1}$. P_{p-1} should achieve all goals in G_p and some planner-selected subset from G_{p-1}.
7. Repeat for remaining drives, each setting the c values in the preceding goal set (i.e. G_{p+1} for the current value of p) to infinity.

One problem with our approach is that a drive could specify that all its goals should be treated as hard, but then no plan can be found to satisfy all of them at once. At this point we have an interesting possibility for reasoning. If the goals were provided by a human then the robot could ask if it could violate certain constraints (e.g. a default deadline assumption), or the human could just be told that the goal will not be possible given the previously existing goals. Alternatively the system could loosen any self-imposed constraints until a plan can be found (i.e. perform metareasoning).

A further problem is that the temporal, net-benefit, planner used in the CogX systems, Temporal Fast Downward [3], does not support goals with deadlines. We are currently investigating ways to make progress on this issue.

In order to address the need to generate meaningful costs-to-drop for goals within the same drive but from different goal generators (as in the previous Dora example), we propose to investigate a mechanism that explicitly considers the range of possible future behaviours that could be facilitated by the successful achievement of the goal in question. This approach would use a collection of hypothetical future tasks that the robot might expect to be given. Goals that allow a greater number of these future tasks to be performed (or performed better in some way) should be preferred by the goal activation mechanism.

Following [2] we would prefer to represent homeostatic drives as resource constraints within the planning domain. This allows the planner to reason about them in conjunction behaviours that consume these resources. We will also allow the explicit generation of goals to achieve homeostatic drives. This is to deal with the case where no other goal is active but the drive needs to be satisfied (an identified weakness in [2]).

3 Representations for Cognitive Systems

Probabilistic approaches to representation and reasoning are core to the capabilities envisaged for the new generation of integrated systems we hope to create. Many components of the CogX integrated systems (and other state-of-the-art robots), such as localisation and vision, are founded on probabilistic principles. However, we had not previously been able to design and implement a system that took advantage of probabilistic knowledge for its entire operation. This was mainly due to lack of support for such knowledge in core decision-making systems. In Years 2 and 3, thanks to developments such as the switching planner in WP4 (see DR.4.3), the conceptual map in WP3 (see DR.3.2), and the belief models developed in WP1 (see DR.1.2), we have been able to create an integrated system (Dora) which is able to both represent its knowledge about the world probabilistically, and solve planning problems in the large state spaces this entails.

Now we have this system, we have been investigating whether its is actually an improvement on its non-probabilistic ancestors stretching back from the Year 1 Dora system (see DR.7.1) to the CoSy Explorer systems [8]. We are performing this evaluation using the Dora Year 2 task of locating a known object given an existing map of the environment. To find the object Dora must decide first in which room to look, then where to look in this room, and finally how to interpret the results of running its visual recogniser. In the purely deterministic version of the problem, Dora knows exactly what category of room the object appears in, can unambiguously determine the category of a room, and interprets the results of its recogniser as always reliable. In the case of being instructed to find the cornflakes, the behaviour of deterministic Dora can be characterised as follows: head straight for the

kitchen, look in all places that can support an object, and if the recogniser ever returns a positive result, then the cornflakes have been found (or vice versa). In the probabilistic case Dora has a distribution over which rooms the cornflakes can appear in, a distribution over the possible categories of each room, and uses an observation model to interpret the results of its recogniser. In this case Dora chooses to look in the room with the highest likelihood of containing the cornflakes (i.e. the one that is mostly likely to be the one that is most likely to contain the object) and treats the results of the recogniser as influencing its belief about the presence of the object. This behaviour continues until Dora's certainty about the presence of the object passes a given threshold. This will entail looking in all possible rooms, and integrating the evidence provided by observations into its knowledge about its current environment.

The details of our study are presented in Annex A.1. The overall behaviour of the different systems is closely linked to the structure of the environment and the Dora's sensing abilities. If the target object is not placed in the room that Dora believes it should be in (the *non-canonical* case in the annex), then the deterministic system will never find it, but the probabilistic system should do eventually. If the object is placed in the room where it is expected to be (the *canonical* case) then whether the deterministic system reports its existence correctly is determined by vision: if the recogniser correctly detects it then it is found. However this system cannot cope with false positives or false negatives, and so might fail to report the object when present (if occluded from the chosen view point) or report it to be present when it is absent. In contrast, the probabilistic system is able to use the observation model to choose how many views and recognition attempts to use in order to reach a predetermined level of certainty in its belief about the presence of the object. Thus a deterministic system will be as good as any other system in the case where the object is clearly visible in its canonical room, but will be outperformed by a probabilistic system in cases where either the object location is non-canonical, or when the observation of the object yields non-deterministic results (which is the norm).

References

[1] Alper Aydemir, Moritz Göbelbecker, Kristoffer Sjöö, Andrzej Pronobis, and Patric Jensfelt. Plan-based object search and exploration using semantic spatial knowledge in the real world. In *Fifth European Conference on Mobile Robots (EMCR'11)*, 2011.

[2] 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.

[3] Patrick Eyerich, Robert Mattmüller, and Gabriele Röger. 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)*, 2009.

[4] Marc Hanheide, Charles Gretton, Moritz Göbelbecker, Andrzej Pronobis, Alper Aydemir, Hendrik Zender, Richard Dearden, Nick Hawes, Patric Jensfelt, Geert-Jan Kruijff, and Jeremy L. Wyatt. Exploiting probabilistic knowledge under uncertain sensing for efficient robot behaviour. Technical report, CogX consortium, 2011.

[5] Marc Hanheide, Nick Hawes, Jeremy Wyatt, Moritz Göbelbecker, Michael Brenner, Kristoffer Sjöö, Alper Aydemir, Patric Jensfelt, Hendrik Zender, and Geert-Jan M. Kruijff. A framework for goal generation and management. In *Proceedings of the AAAI '10 Workshop on Goal Directed Autonomy*, July 2010. http://home.earthlink.net/ dwaha/research/meetings/aaai10-gda/.

[6] Nick Hawes. A survey of motivation frameworks for intelligent systems. *Artificial Intelligence*, 175(5-6):1020–1036, 2011.

[7] Nick Hawes, Marc Hanheide, Jack Hargreaves, Ben Page, Hendrik Zender, and Patric Jensfelt. Home alone: Autonomous extension and correction of spatial representations. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '11)*, May 2011.

[8] Kristoffer Sjöö, Hendrik Zender, Patric Jensfelt, Geert-Jan M. Kruijff, Andrzej Pronobis, Nick Hawes, and Michael Brenner. The Explorer system. In Henrik I. Christensen, Geert-Jan M. Kruijff, and Jeremy L. Wyatt, editors, *Cognitive Systems*, volume 8 of *Cognitive Systems Monographs*, pages 395–421. Springer Berlin Heidelberg, April 2010.

[9] D. Skočaj, Matej Kristan, Alen Vrečko, Marko Mahnič, Miroslav Janicek, Geert-Jan M. Kruijff, Marc Hanheide, Nick Hawes, Thomas Keller, Michael Zillich, and Zai Zhou. A system for interactive learning in dialogue with a tutor. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2011.

[10] Dennis Stachowicz and Geert-Jan M. Kruijff. Episodic-like memory for cognitive robots. *Journal of Autonomous Mental Development*, 2011.

[11] Jeremy L. Wyatt, Alper Aydemir, Michael Brenner, Marc Hanheide, Nick Hawes, Patric Jensfelt, Matej Kristan, Geert-Jan M. Kruijff, Pierre Lison, Andrzej Pronobis, Kristoffer Sjöö, Danijel Skočaj, Alen Vrečko, Hendrik Zender, and Michael Zillich. Self-understanding and self-extension: A systems and representational approach. *Autonomous*

Mental Development, IEEE Transactions on, 2(4):282 – 303, December 2010.

A Annexes

A.1 Exploiting Probabilistic Knowledge under Uncertain Sensing for Efficient Robot Behaviour

Bibliography Marc Hanheide, Charles Gretton, Moritz Göbelbecker, Andrzej Pronobis, Alper Aydemir, Hendrik Zender, Richard Dearden, Nick Hawes, Patric Jensfelt, Geert-Jan Kruijff, and Jeremy L. Wyatt. Exploiting probabilistic knowledge under uncertain sensing for efficient robot behaviour. Technical report, CogX consortium, 2011

Abstract Robots can made more robust by employing task level planning that takes explicit, probabilistic account of uncertainty in state, action effects, and observations. To achieve this we need both good representations and efficient algorithms for planning with them. Our general approach [11] is to have the robot explicitly represent and reason about the informational effects, as well as the physical effects of its actions. The main contribution of this paper is to show how to reason efficiently about the informational effects of unreliable sensing actions. The major contributions are a switching planner and a formulation of view planning and visual sensing actions that allows us to pose the planning problem for visual search as a POMDP (partially observable Markov decision process). The switching planner is a continual planning system which is able to plan in large problems posed by that model, by automatically switching between decision-theoretic and classical procedures. The decision theoretic planner is a POMDP solver and is able to reason precisely about both state uncertainty and the unreliability of sensing. POMDP planning can only be applied to small problems (hundreds or thousands of underlying states), whereas the task we tackle here has of the order of 10^{27} states. Our switching planner therefore decides on the fly how much of the problem the POMDP solver can handle, and how much should be tackled by a fast but heuristic planner based on a continual planning framework. This switching approach allows us to handle much larger problems than previously while retaining plan quality. Satisfying the conditional independence assumption made by the POMDP framework is not trivial, and so we show how to generate a set of possible views of a scene which satisfy this requirement. We evaluate our contributions in a robot that performs object search tasks in real-world indoor environments. The results show that our approach of combining probabilistic and common-sense knowledge improves on using common-sense knowledge alone, which in turn improves on an uninformed search strategy.

Relation to WP WP1 ultimately aims to yield an architecture, or at least advice on constructing architectures, for robots that can self-extend in order to robustly operate in real-world environments. Acting under uncertainty,

whilst still being able to exploit the regularities present in the world, is a key problem for any such robot. This article explores a range of options open to system builders facing this problem. It thus contributes to WP1 by demonstrating the effects of different architectural and representational options on a robot solving a problem in the real world.

A.2 Episodic-Like Memory for Cognitive Robots

Bibliography Dennis Stachowicz and Geert-Jan M. Kruijff. Episodic-like memory for cognitive robots. *Journal of Autonomous Mental Development*, 2011

Abstract The article presents an approach to providing a cognitive robot with a long-term memory of experiences a memory, inspired by the concept of episodic memory (in humans) or episodic-like memory (in animals), respectively. The memory provides means to store experiences, integrate them into more abstract constructs, and recall such content. The article presents an analysis of key characteristics of natural episodic memory systems. Based on this analysis, conceptual and technical requirements for an episodic-like memory for cognitive robots are specified. The article provides a formal design that meets these requirements, and discusses its full implementation in a cognitive architecture for mobile robots. It reports results of simulation experiments which show that the approach can run efficiently in robot applications involving several hours of experience.

Relation to WP Introspection for self-extension, one of the targets of WP1, requires systems that are able to represent the past experiences of a robot operating in the real world. Without such systems, this information would not be available for introspection. This article demonstrates an episodic-like memory design capable of collecting and storing symbolic information about a robot's experiences in time and space in a way that should allow introspection in future systems.

Episodic-Like Memory for Cognitive Robots

Dennis Stachowicz & Geert-Jan M. Kruijff

***Abstract*—The article presents an approach to providing a cognitive robot with a long-term memory of experiences – a memory, inspired by the concept of episodic memory (in humans) or episodic-like memory (in animals), respectively. The memory provides means to store experiences, integrate them into more abstract constructs, and recall such content. The article presents an analysis of key characteristics of natural episodic memory systems. Based on this analysis, conceptual and technical requirements for an episodic-like memory for cognitive robots are specified. The article provides a formal design that meets these requirements, and discusses its full implementation in a cognitive architecture for mobile robots. It reports results of simulation experiments which show that the approach can run efficiently in robot applications involving several hours of experience.**

***Index Terms*—episodic-like memory, long-term memory, event structure, experience, cognitive robot, human-robot interaction**

I. Introduction

Robots do not just live for the moment, act and interact for the moment. The general idea is for robots, for cognitive systems, to operate over longer periods of time. Periods in which the world may change, experience may be gathered, and what is observed at one point may become useful later on. That is the general idea. In this article we present an approach to episodic-like "long-term" memory for cognitive robots that tries to realize that idea. The approach is conceptually inspired by what we know about how humans remember events but also by related findings in research with animals. It moves beyond existing approaches for long-term memories for artificial agents like robots in that it provides a flexible, scalable setting for storing and efficiently retrieving experience.

When a human recalls what she experienced, her memories share many of the qualities of the original experience. The amount of information which she stores is considerable. Moreover, the addition of new information to memory appears to be done automatically, without effort. Even when she does not intend to memorize what she experiences, the information is preserved. Afterwards it can be retrieved in new and unforeseen situations. She can connect the past to new experiences, enabling an entirely different quality of their interpretation.

How could we make it possible for a robot to do the same?

Below, we discuss the background (§II), and formulate a design for episodic-like memory for cognitive robots on the basis of conceptual and technical requirements by biological systems (§III). We then present an implementation of the approach (§IV), and its evaluation in a cognitive system for a mobile robot (§VI). The evaluation is conducted on scenarios recording up to 100.000 events. It shows the implementation to be efficient (fast insertion and retrieval of events), and scalable (at most linear increase of retrieval time, given memory size).

We contribute an approach, conceptually inspired by biological insights, which makes it possible for a robot to remember things, connect experiences over space and time, and recall them in different ways. And do so over a longer period of time. When viewed as a structure, the memory system we propose provides a non-intrusive, integrated representation of events and their spatio-temporal contexts. It allows events to overlap or to be nested, and more generally to form partonomic hierarchies. Different levels on this hierarchy represent different levels of granularity in stored experience. Across all levels a homogenous notion of spatio-temporal context of events and events themselves is retained. This offers several advantages for processing events, by enabling parsimonious interfaces and processing principles. Viewed as a mechanism, the memory allows new experiences to be acquired "in one shot." It provides a flexible and modular architecture for the recognition of complex events and the construction of partonomic hierarchies. Cued retrieval mechanisms allow integrated event structures to be retrieved using underspecified events as cues.

German Research Center for Artificial Intelligence, DFKI GmbH, Saarbrücken Germany. The research reported in this paper was financially supported by EU FP7 IP "Cognitive Systems that Self-Understand and Self-Extend" (CogX, ICT-#215181) and EU FP7 IP "Natural Human-Robot Cooperation in Dynamic Environments" (NIFTi, ICT-#247870), both in the EU Cognitive Systems & Robotics Unit.
E-mail: dennis.stachowicz@gmx.de, gj@dfki.de
Manuscript submitted March 1, 2010; last revised February 10, 2011

II. Background

We provide a brief survey of the development of different notions of *episodic memory* in psychology and cognitive neuroscience. We particularly focus on the recent notion of *episodic-like memory* introduced by Clayton *et al.* This notion forms the conceptual basis for this article. We then review relevant approaches in cognitive robotics and related fields. (For a much more detailed discussion we refer the reader to [1].)

A. Psychology and neuroscience

Until the middle of the twentieth century memory was largely thought of as a singular faculty. Although scientists already expressed doubts that this might not be the case in the 19th century, there was neither a generally agreed view of different memory systems nor any compelling evidence of their existence [2], [3]. This situation began to change significantly around the middle the 20th century. At this time a distinction between a short-term memory (STM) and a long-term memory (LTM) was put forward [4]. It received part of its empirical support from case reports of the amnesic syndrome, e.g. in the famous case of H.M. [5]. H.M. who had both of his medial temporal lobes surgically removed seemed to be unable to store new memories for periods of time longer than minutes. On closer observation, however, not all forms

of learning on longer timescales were affected. Although he was severely impaired with respect to remembering events or facts [6], [7] he was still able to learn and retain new skills like mirror drawing [8].

Other abilities found to be preserved in amnesics often include artificial grammar learning, category learning, perceptual and conceptual priming [9]. Several terms have been proposed to summarise these kinds of memory, and to contrast them with the memory for facts and events which is mostly affected in amnesic patients. Explicit/implicit and declarative/non-declarative are the terminological pairs which remain most influential and are still being used. Declarative memories can be described as those memories which are consciously accessible. They are more easily verbalised and match the concept of memory in everyday language. Squire [2] characterises declarative memories as representational, as models of the external world to which truth values can be assigned. Non-declarative memories are neither true nor false. They are "dispositional and [...] expressed through performance rather than recollection" [2, p. 173]. Figure 1 shows the grouping of declarative and non-declarative memory systems in a widely used taxonomy.

In 1972 Tulving proposed to discern two subsystems of declarative based on the type of their contents [10]. Semantic memory was supposed to hold general, abstract facts about the world without the context in which they were learned, whereas episodic memory was thought to contain personally experienced events embedded in their original spatio-temporal context. Thus, episodic memory is about what happens when and where. It is a memory systems which allows us to "mentally travel back in time" to re-experience a past situation. Furthermore, Tulving suggested that re-experienced episodes are not just flat with respect to their structure but that they can also be nested or overlap [11][p. 37ff].

Later, Tulving changed his definition of episodic memory significantly [12]. His newer definition strongly emphasised the role of *autonoetic awareness* together with two other concepts needed for mentally travelling in time: a traveller, i.e. the *self*, and time, or more precisely, *subjectively sensed time*. Their conjunction was then suggested to be the essence of episodic memory.

Demonstrating episodic memory in any non-human species based on such a definition is obviously more than problematic. This has led Clayton *et al* to distinguish between phenomenological and behavioural criteria for episodic memory. They propose the term *episodic-like memory* [13], [14] for a memory system which meets the following criteria:

- Content: Based on a specific experience information about what happened when and where is stored.
- Structure: What- where- and when-information forms an integrated representation, i.e. "retrieving any one feature automatically retrieves the other features".
- Flexibility: The information stored is declarative in nature and can be flexibly deployed. In particular, it can interact with semantic knowledge even if the latter was gained after the episode was encoded.

Hence, episodic(-like) memory is a form of *one-shot learning*. A single exposure is enough to form a unique representation of a situation. In Clayton *et al*'s view this uniqueness and the integrated representation rule out that pieces of episodic information of different episodes (like the 'what' information of one and the where information of another episode) are mixed or become indistinguishable [15].

Furthermore, Clayton and Russell [16] point out that the knowledge of what happened when and where does not necessarily imply the engagement of an episodic-like memory system. For an instance of episodic-like memory these pieces of information must be present but they must also reflect the "perceptual relation" or "perspective" of the original experience. By means of this clarification Clayton and Russell refine the notion of episodic-like memory such that it also encompasses ideas underlying Tulving's newer definition of episodic memory. However, they do so by restricting the representation of memory contents instead of introducing a dependency on introspection as is the case with Tulving's newer definition.

B. Cognitive robotics and related fields

Episodic memory in ISAC: ISAC, short for Intelligent Soft Arm Control, is a humanoid robot [17]. It runs a cognitive system which comprises several sub-systems, multiple memory systems and a module simulating emotions. Among its memory systems is an episodic memory system which constantly records contents of the working memories with timestamps and combines all contents appearing between two goal changes into an episode. Later on, these episodes are retrieved again to support a planner based on the assumption that there is a "correct episode for a given situation". Candidate episodes are those which are highly similar to the current situation. A few additional heuristics (e.g. "prefer recent episodes") are used to rank candidates. The highest-ranked candidates are written to a small working memory for access by the planner. ISAC's episodic memory system thus clearly shows a form of one-shot learning of relevant data. The data stored cover the 'what', 'when' and 'where' components required by Clayton's content criterion. In that respect, ISAC's episodic memory system appears to be a sound approach to a case-memory for planning problems in a robotic system. However, it does not provide the flexibility expected of an episodic-like memory system. It is purely single-purpose. It defines episode boundaries exclusively in terms of changes of the robot's overall planning goal, and it does not allow for overlapping or nesting of event representations. Finally, it is unclear whether it is efficient and scalable enough to run at the timescales of a long-term memory system.

Episodic memory in EPIROME: TASER is a service robot targeted at dynamic, real-world office environments [18]. Sample tasks proposed for such an environment include delivering messages to project group members. Jockel *et al* argue that an episodic memory module which continuously stores experience could help in solving such tasks or in avoiding unnecessary overhead in task execution. For example, if experience indicated that usually there is nobody in the server room to whom a message could be delivered, this room would be excluded from places where the robot would look.

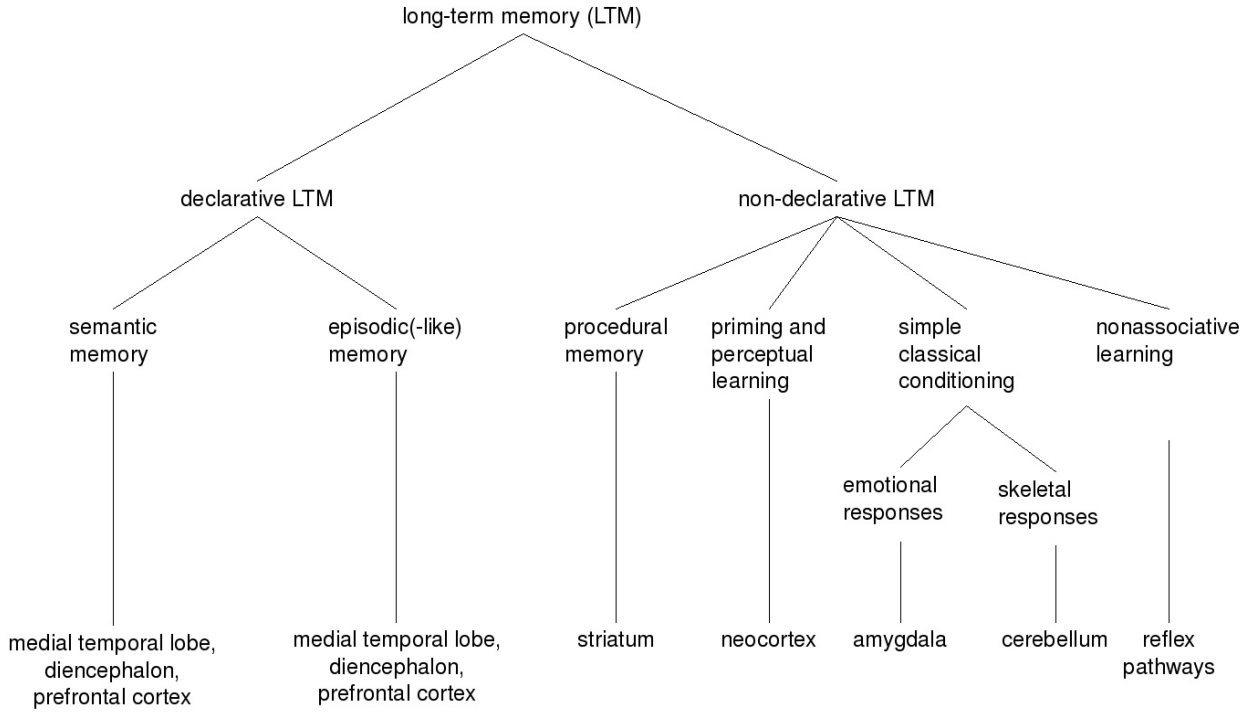

Fig. 1. A taxonomy of long term memory systems with selected neurobiological correlates. Adapted from [2].

To this end, Jockel *et al* propose the EPIROME framework [18], [19]. It provides a software module developed in an object-oriented programming language in which event objects can be generated and further processed. Events generated during a robot run can be visualised. It is planned for EPIROME to add long-term storage and retrieval functionality to it. Two different possible approaches are mentioned in [18], [19].

EPIROME differs from other approaches in that events are typed, and arranged in a hierarchy. This hierarchy is modeled as a hierarchy of interfaces and classes in an object-oriented programming language. It consists of two layers. One is the domain-independent layer which is a static part of EPIROME and contains four interfaces: A (root) `Event` interface and and three sub-interfaces of the latter (`CommandEvent`, `PerceptionalEvent`, `ExecutiveEvent`). The domain-dependent layer contains interfaces and classes derived from one of these three types.

In its current state [18], [19], EPIROME clearly does not fulfil requirements for an episodic-like memory system. Neither have long-term storage or retrieval mechanisms been detailed yet, nor representations of stored events or episodes. Nevertheless, it is interesting to note EPIROME's event type hierarchy as a new approach towards a conceptual understanding of experience generated during robot runs. This could potentially be useful in several respects (if integrated with corresponding modules): To allow deliberate reasoning about events, to engage in dialogue about events, or simply for more focused retrieval strategies from a memory store. A major drawback of this approach, however, results from the fact that the hierarchy must be specified in terms of interfaces and classes in an object-oriented programming language. Furthermore, the hierarchy must be defined a priori, and will thus always be static.

A few other memory systems developed for agents in simulated worlds or non-robotic cognitive architectures are outlined below. Most of these have been designed with the same purpose as ISAC's episodic memory, and thus often have the same flaws. In addition the systems are often not usable for the design of new cognitive robot systems because they are tied to a particular system context.

Episodic memory in SOAR: [20] introduce SOAR-EM, an episodic memory extension to the SOAR cognitive architecture, which focuses on case-based reasoning tasks (cf. [21]), similar to the corresponding system in ISAC. In a later publication [22], they described extensions to their system and broadened the range of considered applications. These include: Noticing novel situations, detecting repetition, virtual sensing (retrieving past sensory input to aid in a new situation), predicting effects of own actions or other perceived events, management of long-term goals, retroactive learning (re-analysis of experiences acquired under time pressure, at a time of low system load), and re-analysis of knowledge.

For three of these functions, Nuxoll and Laird tested whether their system could provide the necessary support. As

a test environment they used a simple simulation (TankSOAR) in which a tank can attack, or can be attacked by, other tanks. It also needs to maintain a certain energy level by regularly recharging its batteries. As it cannot perceive the whole simulated world at once, it is not always able to sense the nearest charging station. Nuxoll and Laird showed that their memory system can support virtual sensing, and that a tank able to use it can find the charging station more quickly than a tank which has to rely on random search. Similar cases studies involving the simulated tank have been conducted for two other tasks related to the prediction of action effects.

The memory system uses a SOAR-specific representation format for its episodes. Each time the SOAR agent takes an action an episode is formed which contains data from the architecture's working memory. Hence, episodes are much shorter than, for example, in ISAC's episodic memory but the same criticisms of episode structures hold here, too.

SOAR-EM's retrieval mechanism is more flexible than that of ISAC's episodic memory: SOAR-EM does not automatically and exclusively retrieve memories similar to the current situation. Instead it allows deliberate retrieval by placement of a retrieval cue onto the working memory. Its retrieval mechanism, however, was reported to be inefficient and difficult to optimise further [22].

Other approaches to episodic memory: The overview above is by no means exhaustive. There exist several approaches for equipping virtual characters with episodic or autobiographic memories, e.g. Rity [23], Homer [24], and story-telling characters [25] or [26]. The latter two are sophisticated in their capability for inserting and retrieving complex event content. The approach in [25] is however closely tied to planning, raising similar concerns as voiced above for ISAC and EPIROME.

III. Design

A. Design requirements

§II surveyed naturally occurring episodic-like memory systems and the development of the concepts of episodic and episodic-like memory. These concepts have inspired the artificial system (ELM) we describe below. Before we turn to the description of our system, we present a short summary of requirements (R1 - R11) which we consider important for the development of artificial episodic-like memory systems. We start with characteristics of natural episodic-like memory systems, reinterpreted as requirements, and extend these with properties desirable from a technical point of view.

The first three requirements are given by the criteria of Clayton and colleagues [14]:

- **Content** (R1) Previously experienced events are recollected within their spatiotemporal context.
- **Structure** (R2) Each event together with its spatial and temporal context form a **single integrated representation** which is retrieved as a whole upon retrieval of any one feature of the event.
- **Flexibility** (R3) Flexible deployment of this information in novel situations is possible because episodic-like memory is set within a **declarative framework**.

But there are more points to be taken into account: Acquiring new episodic-like memories is a particularly fast form of learning, even **one-shot learning** (R4). Unlike most learning scenarios there are no repeated presentations of training examples from which to infer generalizations. Instead episodic-like memory has to deal with the **specifics of a situation**.

Episodic(-like) memory is a form of **long term memory** (R5). Memories can be stored for seconds, minutes, or even months or years. It is also a form of **declarative** (or **explicit**) memory (cf. R3). We can talk about events and that our memories of events are accessible to introspection.

Although episodic-like memory shares this explicitness with semantic memory, the two are different. An episodic-like memory deals with situation specifics , which also implies a certain **perspective** (R6). That is, re-experienced events incorporate the same perspective as in the original experience [16]. Semantic memories, on the other hand, generally abstract away from both perspective and situational specifics.

Finally, events stored on an episodic-like memory can vary considerably in their length. They can even be **nested** (R7) or **overlap** (R8).

The characteristics listed above already restrict the design space for an artificial episodic-like memory. If such a memory is to be used in a cognitive robotics environment a few more properties are desirable. As the 'what' field can contain very diverse information which can be provided from very different software modules the system designed must be "**non-intrusive**" (R9), cf. [27]. It should not require software modules to use specific representations or algorithms internally. At the same time the system cannot rely on other software modules not to change representations and algorithms. Furthermore, the system should be **efficient** (R10) enough to handle the high throughput of events occurring in a running cognitive robot system. This also implies it should **scale** well (R11) to provide for efficient storing and retrieving even as the amount of collected data in an episodic-like memory grows over time.

B. Intuitions Behind the Formal design

We provide a detailed discussion of the formal basis for the the artificial episodic-like memory system ELM in Appendix A. Here, we discuss the basic intuitions behind the formal details. The formalisation provides the basis for the implementation we discuss in the next section.

The formal design is based on a notion of an event or episodic-like memory item, with respect to perceptual processes of the respective observer. According to the content and structure criteria, an episodic-like memory stores information about what happened when and where in an integrated form. Thus the information about an event takes the form $e = (c, l, t)$ where c denotes the event in the narrow sense (i.e. its content), l its location and t its time.

Imagine you are in your office. You turn to your desk and see a specific book on it. Then you turn away and lose sight of it again. There was an event e of you seeing this book during a certain interval of time, say $t = [t_{start}, t_{end}]$, which is defined to be the time (or temporal context) of e. Similarly, you have been standing in a certain place while

seeing the book. This place, more precisely the space your body occupied at this time, is defined to be the location l (or spatial context) of e. Think of it as a three dimensional "box" P_3 approximating this space, or just a simple two-dimensional area that approximates the projection to the ground. In this example, c contains visual information. In other cases it may stem from any other modality with information in a completely different representation. That is why non-intrusiveness must be taken into account when further specifying c. To this end any kind of content (or binary string b) is allowed here, which we extend with type information and features to enable fast retrieval of the event.

Our formal design specifies how we can define relations between events, for example in terms of sub- and super-events, or types of events. This makes it possible to consider complex events, built up from smaller events. The spatial and temporal extent of a complex event can be derived from the spatial and temporal extents of its parts. In this formal framework an episodic-like memory then basically is a set of such events, complex or otherwise, together with a set of operations that we can use to insert and retrieve events. Retrieval can be along any dimension of an event – its content, its type and features, its temporal and spatial extent, or its inclusion in a complex event structure. Crucial here is that we are completely free in designing the content of an event, and the way we decide to structure events. This makes the ELM (fairly) independent of the uses it might be put to in a specific cognitive system.

IV. Implementation of ELM

A. From requirements to index data structures

In the last section a more formal but still abstract representation for events was constructed following characteristics from empirical research. Among the various requirements which influenced the definitions was the structure criterion. This criterion does not only inform the immediate representation of events but also the storage and retrieval processes as "retrieving any one feature automatically retrieves the other features" [14]. Together with the fact that this must happen efficiently and scalable to enable the usage of the system as a long-term memory, this suggests the use of indices on each of the principal components of the integrated memory item. Indices are data structures which allow the storage and fast retrieval of stored data.

B. Index data structures for one- and multidimensional data

For one-dimensional data (like the start time of an event) there are many index data structures. Most widely known are B-trees [28] and hash tables. Indices based on hash tables are not suitable here because they do not support range queries. B-trees and similar data structures can serve this purpose well with only logarithmic worst-case complexity for insertion, search and deletion of entries. For a classic review of B-trees and related data structures see [29].

More interesting is the case of multidimensional data such as areas representing the location of events. Guttman introduced R-trees [30], tree structures which progressively divide an n-dimensional space into smaller and smaller bounding boxes. Figure 2 shows an illustration of an R-tree for the two-dimensional case.

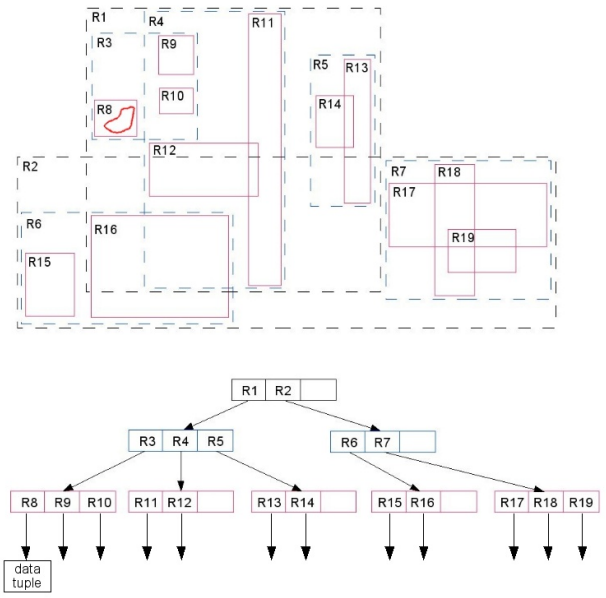

Fig. 2. R-tree for two-dimensional data. Adapted from [30].

Its root node contains pointers to two child nodes, each representing a large portion of the plane (rectangles R1 and R2). Each child node further splits the rectangle it represents into smaller ones, e.g. R1 into R3, R4, R5, until a leaf node is reached which contains a pointer to the actual data tuples containing the indexed shapes. When looking up an entry the algorithm starts at the root node, checks if the sought region lies within one of the rectangles pointing to child nodes and further descends until a leaf node is reached or there no child node representing the sought region. Note that the rectangles can overlap. Whenever they do the retrieval algorithm has to descend into more than one sub-tree. This and the fact that R-trees do not have a balancing mechanism rule out a strict worst-case complexity bound. In spite of that R-trees have been reported to perform well on many real world data sets.

Several variants of the R-tree structure have been proposed with the aim of ameliorating one or both of these potential problems: R^+-trees [31], for example, avoid overlapping rectangles by including objects in multiple leaves if needed (thereby increasing disk usage and insertion time for the benefit of shorter query times). R*-Trees [32] and Hilbert R-Trees [33] are related modifications. [34] introduce the Priority R-tree (or PR-Tree), the first R-Tree variant with asymptotically optimal worst-case performance for window queries (queries for all objects intersecting a given (hyper-)rectangle). To answers such queries it requires no more than $O((\frac{N}{B})^{1-\frac{1}{d}}+\frac{T}{B})$ I/O operations where N is the number of d-dimensional hyper-rectangles stored, B is the disk block size and T is the output size. In experiments Arge and colleagues showed that the PR-Tree performs comparable to other R-Tree variants on real-world data and are superior on synthetic data with extreme distributions. All in all the performance differences between

R-tree variants applied to real-world data are small enough to allow for greater consideration of other factors like availability or complexity of implementation.

C. Index data structures and database management systems

Several of the index structures outlined above have been implemented in current database management systems (DBMS). In particular B-trees are "ubiquitous" [29] and R-trees are available in some DBMS or can be added as extensions. This is the case with the PostgreSQL[1] DBMS and the PostGIS[2] extensions. PostGIS is a software package which was originally created to support geographical information systems (GIS). It uses PostgreSQL's Generalized Search Tree (GiST) interface [35] to add R-tree indices[3] and other functionality needed for GIS to PostgreSQL. This comprises a polygon data type and conversion functionality between different formats, e.g. the well-known text (WKT) and well-known binary (WKB) formats of the OpenGIS Simple Features Specification [36]. In summary, PostgreSQL and PostGIS provide the basic functionality to represent events (including their locations) and to index the principal components of each event.

In comparison to many other design and implementation approaches relying on other software or even implementing software from scratch a design and implementation based on the combination of PostgreSQL and PostGIS offers many advantages. It is stable and mature open source software, it is reasonably fast, it implements various additional optimisation techniques as well as data persistence functionality. It allows for flexible usage and enhancement, especially its standardised representations are beneficial for information interchange with other software in the future. Due to these numerous gains this approach was taken. It is described now in more detail.

D. Database design

Fig. 3 shows an entity-relationship diagram [37] based on the formal framework in section III. The most central entity 'Event' is depicted with its attributes time, consisting of a start and an end time, location and event-specific binary data. It is uniquely identifiable by an 'EventID'. This identifier is used in the construction of several relations. One of them is the relation determining of which events a given complex event consists. In this relation events can have multiple immediate sub-events. Furthermore every event is of a specific type and those event types form a hierarchy in which multiple inheritance is allowed. Finally, events can also have certain named, event-specific features against which matching is also possible.

Based on this ER diagram a table design for the object-relational PostgreSQL DBMS was developed. The table design, however, is not a direct one-to-one mapping from the ER diagram but includes a few minor optimisations and extensions. The most notable extension concerns the type hierarchy. Cues for retrieval do not necessarily include the exact type of an event but may include only a super-type. To avoid expensive traversal of the type hierarchy for each query its transitive closure can be computed in advance. The transitive closure is represented in a separate table.

[1] `http://www.postgresql.org`

[2] `http://postgis.refractions.net`

[3] Strictly speaking a plain PostgreSQL system already contains R-tree indices. PostGIS adds more capable ones which can also handle null values etc. and run with comparable efficiency.

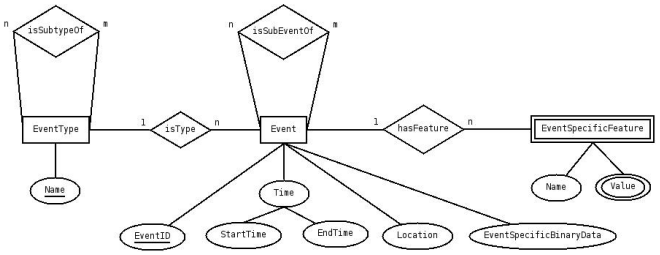

Fig. 3. Entity relationship diagram for the most basic ELM concepts

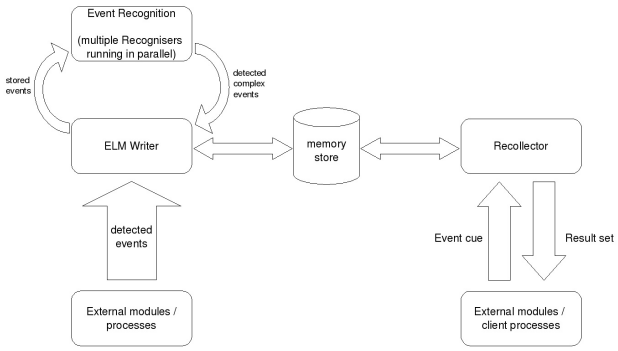

Fig. 4. Information flow in ELM (core components only)

E. Processing events - interfaces and information flow

Above we outlined the basic representations of data in the ELM system or, more precisely, their data base storage format. We now proceed with a broad outline of the implementation of the basic operations on episodic-like memories, and of their interfaces. Figure 4 presents a high level view of the components and information flow in the ELM system which are described individually below.

Adding new events to the memory store: Assume that the ELM system outlined in this chapter runs as part of a larger cognitive robot system and that another software module, e.g. a vision module, detected an event. The event is reported to ELM in an integrated form deduced from the event definition, i.e. as one `Event` object comprising all required information.

The receiving instance on the side of ELM is the `ELMWriter`. It encapsulates all communication with the underlying database related to the addition of new events to the memory store, supporting an abstract view of the memory store. It holds a permanent connection to the underlying database. Upon retrieval of newly reported events it transmits the event information to the corresponding database tables. To ensure full consistency of the memory store at any time the insertion of each event is treated as one transaction. After committing the changes to the database and reception of a

unique event ID each event together with its ID is passed on to the complex event recognition component.

Recognition of complex events: The recognition component's task is to find sequences of events which form a complex event and to report the newly found event with information about its sub-events back to the `ELMWriter`.

At its core the recognition component has a `RecognitionManager` class with which instances of classes implementing the `ComplexEventRecogniser` interface can be registered. Upon reception of an event with its ID it stores it in its internal queue, a circular buffer, and returns immediately. A separate thread blocks until the the queue is no longer empty. Then it passes the new events to the registered `ComplexEventRecogniser` for processing and deletes them from the queue. The `ComplexEventRecognisers` perform the actual recognition task, each for one or more complex event types. There is a huge design space for their implementations ranging from the simulation of automata with events as the alphabet of their formal language to machine learning algorithms with the ability to classify event sequences to arbitrary Java methods satisfying the interface. After a `ComplexEventRecogniser` found a sequence of events constituting a complex event the complex event is constructed from these sub-events. It is reported back to the `ELMWriter` and enters the recognition process itself then. This cycle enables the efficient formation of an event partonomy encompassing events on very different levels of granularity. At each level the complex event recognition process can follow the same principles, use the same interface and abstract from processing on other levels. This simplifies the design and implementation of the recognisers to a great extent as it unburdens them from constructing complete or partial hierarchies. Further simplification and flexibility in design are achieved by the fact that, even on the same level of granularity, recognisers for different event types can work independently of each other. This means that, in summary, all recognisers, irrespective of the level, run in parallel and keep on processing each others results until saturation is reached.

Recollection : Besides the `ELMWriter` there is a second component in ELM which holds a connection to the underlying database, the `Recollector`. It implements cued retrieval of events.

In order to initiate retrieval of events an object of type `EventCue` is passed to the Recollector. The `EventCue` class mirrors the structure of the event cue in definition 14. It contains all fields of the Event class and a field for the set of super-events. In addition it contains fields specifying how the previously mentioned fields with event information are to be matched against events in the memory store. Based on the respective cue the recollector dynamically composes and issues the corresponding queries, receives matching events and returns these to the caller - at any time hiding the underlying database representation.

V. Integration with a cognitive robot platform

Our artificial episodic-like memory system ELM was integrated with a cognitive robot architecture which combines various functional modules. We describe these modules and their composition in the following section. Before we can do so we need to introduce the conceptual framework and middleware which was used to compose the system.

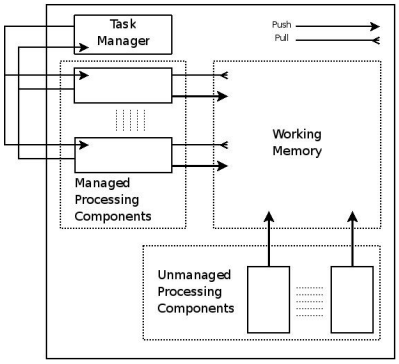

Fig. 5. The CAS Subarchitecture Design Schema (Adapted from [39])

A. The Cognitive System Architecture Schema Toolkit

The middleware we use is a result of research carried out by Hawes and colleagues [38] in the context of the EU FP 6 project Cognitive Systems for Cognitive Assistants (CoSy).[4] Hawes and colleagues analysed requirements for cognitive architectures Based on these requirements they formulated design principles and combined them into a theoretical framework called CAS (Cognitive systems Architecture Schema). Furthermore, a software toolkit called CAST (CoSy Architecture Schema Toolkit) was developed which provides a sample implementation of the schema [39].

a) Components: An architecture instantiating the Cognitive systems Architecture Schema consists of one or more loosely coupled sub-architectures (SAs). An SA is an architectural unit comprising a working memory (WM), a task manager and a varying number of processes. Processes run in parallel and can be either managed or unmanaged. Managed processes are assumed to perform potentially expensive operations. For the latter they can request permission from the task manager which ensures that the amount of expensive concurrent operation remains within an acceptable range. Unmanaged processes, on the other hand, are low-latency processes running all the time, independently of the task manager. Fig. 5 illustrates the general design of an SA following CAS.

b) Information interchange: Processes use the WM to share information. In CAST, processes can add new entries to the working memory, and overwrite or delete existing entries. WM entries are persistent, i.e. once an entry is added to a WM it remains in place until it is explicitly deleted. All WM entries are typed and each type is associated with a specific data format defined in an interface definition language (IDL). When WM contents are modified, information about the change is broadcasted. Managed processes can subscribe

[4] `http://www.cognitivesystems.org`

to receive a subset of this change information. They can, for example, subscribe to changes to a particular WM entry, or to changes where the WM entry instantiates a specific type. Moreover, the received changes can be restricted according to access type (addition, modification, deletion). After receiving the information that a WM entry has changed a managed process can retrieve the changed item and use it to perform further computations. Unlike managed processes, unmanaged processes are only allowed to modify WM entries, but not to retrieve them.

By default it is assumed that processes only access their local WMs (i.e. WMs within the same SA as the process). Nevertheless, it is also possible to modify external WMs or receive change information for these.

c) High-level system design : The Cognitive systems Architecture Schema does not prescribe a way in which processes need to be composed to form SAs. It is suggested, however, that components working on shared information are grouped together, i.e. only very few WM operations should be non-local. In CAST, local operations are typically much cheaper than non-local ones. They usually incur lower costs for broadcasting changes and for filtering broadcast change information. That means that even though many architectural designs can often be found which lead to the same observable behaviour of a system, some are preferable over others as they lead to a lower communication overhead. The effects of such architectural choices and methods to evaluate the same have been investigated by Hawes and colleagues in [40], [41], [42].

B. The Explorer scenario

The Explorer is a mobile robot system which was originally developed in the CoSy project [43] and whose development is continued in the CogX project.

As a hardware platform it uses RobOne, a MobileRobots[5] Pioneer 3 PeopleBot (cf. fig. 6) equipped with a SICK LMS 291 laser scanner. It also comprises a pan-tilt unit with a Videre stereo vision camera. RobOne moves using two drive wheels (tires are visible on the left and right sides of RobOne in fig. 6) and one caster wheel (at its back; not visible in fig. 6). A Toshiba Satellite Pro notebook mounted on RobOne is used to run the Player robot interface. It is also used to operate a CAST instance running a few of the CAST components introduced below. Other CAST components including the ELM module can be run on other machines. For our tests we used an IBM T42 notebook as a second computer. The mounted notebook communicates with other computers using a wireless local area network.

The Explorer explores previously unknown, large-scale space in a dynamically changing environment. It explores space about which it had only partial or no information and which cannot all be perceived at once. Within this space, the robot is not the only actor, humans can be present as bystanders, passers-by or as users.

Exploration can happen in one of two basic modes. One is *autonomous exploration*. In this mode the robot moves around on its own with the aim of reducing the amount of unknown space. While moving it builds up spatial representations on different levels, from low-level, metric representations to high-level, conceptual representations, e.g. room categories [44], [45]. Using the cameras it can detect and classify objects in visited rooms. A common sense ontology and a powerful reasoner allow the robot to draw inferences about the category of a room based on the objects it found there.

Fig. 6. Our hardware platform: An ActivMedia Robotics Pioneer 3 PeopleBot

Another mode of exploration is *Human Augmented Mapping (HAM)* [46], [47], [48]. HAM takes place in a home tour-like scenario. The user commands the robot to follow him and shows the robot around. The robot has the capability to detect and track people via its laser sensor. In this scenario it uses it to follow the user [49]. When the user comments on the environment, e.g. on rooms or objects, the robot uses this information to update its knowledge base. It is, however, not only the user who can initiate dialogue. When pieces of information are found to be ambiguous or contradictory the robot can also initiate a clarification dialogue [50]. The style of spoken dialogue with the user can, more generally, be described as situated, mixed initiative dialogue [48].

After an exploration phase, be it autonomous or in HAM mode, the robot can talk about the spatial entities it came to know about. It can, for example, answer questions about objects or rooms or it can go to a specific room when commanded. When it does not know where an object is, it can search for it, and once it is found, come back to the user to report the object's location.

For the sake of simplicity, the two modes of exploration and the functioning thereafter have been described above as if they were three strictly separate modes of functioning. That, however, is not true. Functionality from all modes can be used in turns. The robot can answer questions in

[5]formerly ActivMedia

between exploration. And the teaching of new facts can happen incrementally with autonomous exploration phases in between.

As a CAS architecture the Explorer consists of several SAs which group processes according to major fields of functionality.[6] The Explorer consists of the following SAs:

- the *Spatial SA* which computes the lower levels of the spatial model, i.e. the metric, navigational, and topological information,
- the *Conceptual Mapping SA* which augments the spatial model with a conceptual level and allows reasoning with objects and regions of space,
- the *Vision SA* which performs object recognition based sensor data from the cameras, and
- the *ComSys SA* which provides the dialogue system.

Besides these already mentioned modules the Explorer also contains two SAs with more integrative functions:

- the *Motivation & Planning SA* which contains a general purpose planner, capable of continual collaborative planning (CCP) (cf. [51], [52], [53]), and
- the *Binding SA* which combines modal representations from other SAs into amodal ones (cf. [27]).

C. Integration with CAST and the Explorer

d) The CAST-ELM layer: ELM's design and implementation have been developed independently of a specific robotic platform or middleware. It was integrated with our robotic system through the addition of the CAST integration layer (C-ELM). The integration layer contains the main modules (`ELMWriter`, `ComplexEventRecognition`, `Recollector`, cf. section IV-E) wrapped into CAST processes and a wrapping of the main data structures (e.g. `Event`, `EventCue`) into CAST data structures. A conversion module supports the mapping of one representation to another.

This setup would already be sufficient for other subarchitectures to report events by writing event structures to the C-ELM working memory, to use complex event recognition and to retrieve stored events via WM interactions. But in order to achieve improvements in usage C-ELM was extended in two respects.

The first extension is related to the fact that SAs other than the navigation SA usually do not collect information about event locations. A dialogue system, for example, is usually not concerned with robot positions and it would introduce considerable overhead both in programming and information interchange between software modules if every other module reporting events would need to collect this information. To avoid these problems a CAST process called `LocationMonitor` was developed. Its design is outlined below.

Another class of additions to the wrapped core modules, a collection of event monitor processes, serves the purpose of reducing code dependencies due to detection and reporting of events and to encapsulate the corresponding code in very few places. Their design is outlined in section V-C0f.

e) Automatic fusion with position information: SAs reporting events for storage by ELM are effectively relieved from the burden of knowing about event locations by a process called `LocationMonitor`. This process allows incomplete event data structures to be written to the C-ELM WM where they are complemented efficiently for further processing.

In more detail, the `LocationMonitor` process works as follows. It listens for changes related to `RobotPose` data structures on the Spatial SA WM. `RobotPose` structures contain information about the robot's position and orientation together with a timestamp indicating when the corresponding measurements and computations took place. When a new `RobotPose` is found or a previously existing one is updated, its timestamp and coordinates are extracted and added to a buffer, internal to the `LocationMonitor`. This buffer is a circular buffer into which elements are inserted in the order of their timestamps. The design as a circular buffer allows the insertion in time $O(1)$. The `LocationMonitor` process also listens for newly appearing `PartialEvents`. These are event data structures in which a few fields are not specified. In particular, information about the location of an event is missing. At least the information about the event type and the event time, however, must be present. When the `LocationMonitor` receives a new `PartialEvent` it tries to determine the appropriate event location based on the data stored in its buffer. To this end it retrieves all positions with timestamps within the event time interval. The fact that positions are stored in the buffer in chronological order enables the usage of a binary search method to find the relevant first and last elements in the buffer within time $O(\log n)$ where n is the size of the buffer. The sequence of matching elements is then used to construct an appropriate event location polygon. Finally, the previously incomplete event data structure is augmented with the reconstructed event location and written back to the C-ELM WM for further processing by the `CELMWriter`.

f) Event monitor processes: Many processes in several SAs running in the Explorer system compute information which can be regarded as experience in the sense of ELM. One possible way to integrate ELM with the Explorer is to make these processes write (possibly partial) event data structures to ELM's WM which would then be handled further by the `LocationMonitor` or `CELMWriter`.

This approach, however, has a few disadvantages: Although the changes required in the reporting processes are tiny (at most a few lines) relatively many program locations are affected. And changes in these locations would usually require good knowledge of the altered code. Changing code in so many places would also mean that the whole system would become dependent on C-ELM's presence. Running or even compiling without C-ELM could become complicated quickly.

Our approach avoids these potential problems by exploiting design properties of the CAST-based Explorer system and of CAST itself, in particular the default data persistence and its support for loose coupling [54]. We developed a set of monitor

[6]The exact grouping of processes into subarchitectures changed several times with the ongoing development of the Explorer system. E.g. some instantiations merged a few SAs and their processes. But as the detailed partitioning of processes into SAs is not important for the presentation of ELM's integration we present only one of these Explorer instantiations here as an example.

processes called event monitors which use data already shared between components. For each WM on which data relevant to ELM is expected (at least) one event monitor process is defined. These processes register for change notifications for certain WM items. Most often this means that the processes register for additions of items which match a specific data type. When the awaited WM change occurs (e.g. when a new item of the type is added) the event monitor process is invoked. It retrieves the changed data structure. Using the new information it constructs an event structure and sends it to the C-ELM WM for insertion into the memory store.

This approach allows us to keep changes related to event detection separate from the underlying computations. The processes in other SAs do not need to be changed. Nor do their internal mechanisms need to be known. The event detection code can thus be considered maximally non-intrusive. It is easy to maintain and allows for simple and quick reconfiguration of the system. By changing a few lines in the CAST configuration file the system can be run with full ELM support, with ELM receiving only from a subset of the available SAs, or with ELM's event detection completely deactivated.

VI. Efficiency and scalability

In order to evaluate the performance of the ELM system a set of experiments in real and simulated environments have been conducted. In a pilot experiment we ran ELM in combination with the Explorer system (cf. V-B) to verify that ELM runs efficiently enough for the purposes of our system. The pilot experiment will be shortly described in section VI-A. In a series of further experiments we set out for a more systematic evaluation of ELM's performance using simulation methods. These will be outlined in section VI-B. Due to space limitations we will describe the experiments in a very condensed form, only.

A. Running ELM with the Explorer system

In a pilot experiment the event monitor processes approach was used to store a number of events which occur in our Explorer system while it explores space as described above. These include, for example, events of the robot entering or leaving rooms, learning a category for a given room, or detecting a person using its laser scanners. The thus enhanced Explorer system was run four times where each run lasted between 10 minutes and one hour. On average around 15 events were recorded per minute.

Approximately once every minute queries for subsets of the stored events were issued. The retrieved sets of events corresponded, for example, to the events which occurred within the last ten minutes, or to people detection events in a specific room.

Measuring the insertion and retrieval times it was possible to establish that both are within acceptable ranges (below one second for retrieval, in the range of milliseconds for insertion). Because of interaction with concurrently running software it was, however, not possible to acquire precise measurements. Moreover, the amounts of events recorded were too small to draw conclusions about the growth of processing times as the memory store fills with events. Similarly, not all variants of queries for stored events were tested in this experiment. For these reasons a series of simulation experiment have been conducted.

B. Simulation experiments

1) Experimental setup and methods: We constructed an *event simulator* and a *query simulator* to systematically test for ELM's efficiency limits, given a specific (moderate) hardware platform (see below). With these we conducted three experiments. Before we detail the differences between those in section VI-B1d we first outline the general setup common to all experiments.

a) Simulation software: The *event simulator* was constructed so as to provide a high amount of events with properties (e.g. size, number of features, etc.) observed in our service robot platform or expected for its future development. It simulates both atomic and complex events. More specifically, we use a hierarchy defining over 11k event types. Each event generated by the simulator is assigned a type taken at random from this hierarchy. The type hierarchy has the form of a directed acyclic graph with ten source nodes. All nodes excluding leaf nodes have ten children. Each event has five event-specific features, i.e. (feature name, feature value)-pairs, and carries a binary payload corresponding to a small serialised Java object (approximately 200 bytes). Simulated robot movement in randomly changing directions forms the basis for event locations and event times are based on time passing during the simulation. Every tenth event is a complex event to which the preceding nine events are sub-events.

The main objective of the *query simulator* is to test the speed of retrieval functionality provided by ELM. In order to know sensible values to construct cues from, copies of events generated by the event simulator are buffered and drawn randomly (with equal probability for all events generated and stored). Cues are then constructed based on one or more fields of the drawn event, e.g. the event type or the set of event-specific features. The relations between the specified fields and sought events are also drawn randomly. Combining the available fields and their associated relations (as specified in definition 14 on page 15) leads to 27 basic classes of one-dimensional queries (queries based only on one field).

b) Experimental runs: Each simulation experiment was repeated ten times under equal conditions. By equal conditions we understand an empty memory store and simulation with the exact same parameters. Random number generators used in the simulation have, however, been initialised differently each time. Each repetition (also called experimental run below) consisted of alternating event generation/insertion periods and query generation/execution periods. Insertion periods comprised 100 event insertions each. Query periods consisted of 25 queries. Per run 100k events were inserted and 25k queries (on average 1k per basic type) were issued.

c) Platform: The experiments were performed on an IBM Thinkpad notebook (Intel Pentium M 1.7 GHz CPU, 768 MB of physical memory) running Ubuntu Linux 9.04 and PostgreSQL 8.3 with PostGIS 1.3.3.

d) Differences between experiments: Three structurally very similar experiments were conducted with the simulation software outlined above. The main difference between the three regarded the number of fields of the event structures which were used to construct cues. In the first experiment event cues specifying one field were used. In the second and third two or three fields were specified, respectively.

2) Results:

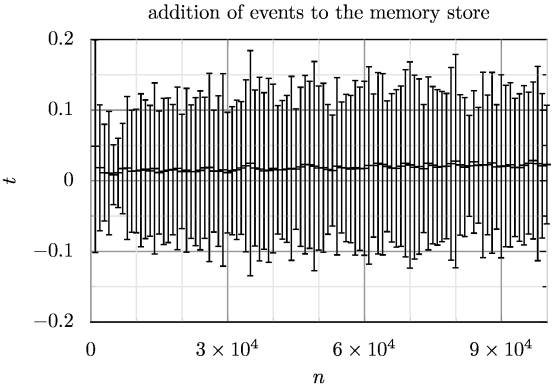

Fig. 7. Insertion of events

a) Insertion: Fig. 7 shows the measured processing times for event insertion. The size of the memory store in events at the time of a new event insertion is plotted against the x-axis. Processing time in seconds is plotted against the y-axis. Each data point indicates the mean processing time for 1k consecutive event insertions, averaged over ten runs. (I.e. each data point in the plot is based on 10k measurements.) Error bars indicate sample standard deviations for these insertions.

Over the course of the experimental runs mean processing times showed only a very mild increase with general levels relatively stable around the global average of 19 ms. However, the time for single insertions varies to a greater extent. The error bars in fig. 7 show that sample standard deviations were mostly above 100 ms (global average 110 ms). The maximum processing time for an individual insertion operation was 8.2 s.

b) Retrieval based on one component: Averaged across all query conditions, retrieval took 145 ms with 412 ms sample standard deviation. For the different classes of retrieval cues averages ranged from 14 ms (matching on the exact event-type) to 589 ms (event type incl. sub-types). Sample standard deviations ranged from 78 ms (matching on the exact event type) to 776 ms (matching event locations in intersection mode). The longest processing time of a query amounted to 13.5 s and was for measured matching an event location.

Because of the limited space we present a plot only for one of the classes of event cues, namely for retrieval based on type or subtype (fig. 8). Similar to the case of insertion this plot shows memory store sizes (measured in events, rounded up to the nearest multiple of 1k) at the time of an operation plotted against the x-axis. Plotted against the y-axis are now averaged processing times for retrieval operations. More precisely, each data point indicates the mean processing time for 100 queries *on average* (as the result of random query generation in ten runs, see above). Error bars indicate sample standard deviations for these chunks of processing times.

As mentioned above this kind of query is the most expensive on average. Its approximately linear growth is, however, typical of most of the query classes. (Some location-based queries show slightly milder increases.)

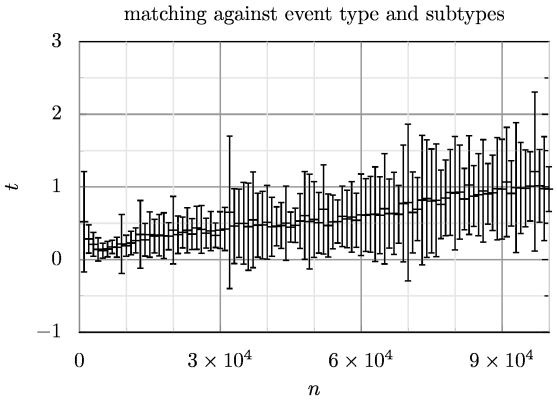

Fig. 8. Retrieval based on event type (including subtypes)

c) Retrieval based on two components: Retrieval time averaged over all queries was 109 ms, with a sample standard deviation of 301 ms. For the different classes of retrieval cues averages ranged from 60 ms to 221 ms. Sample standard deviations ranged from 196 ms to 405 ms. The longest processing time for an individual query was 18.1 s. We refrain from a detailed presentation of data for each query class, here. Nevertheless, we like to mention that, as above, the development of the runtimes as the memory fills with events approximates linear functions.

d) Retrieval based on three components: Retrieval time averaged over all queries was 129 ms, with a sample standard deviation of 315 ms. For the different classes of retrieval cues averages ranged from 83 ms to 219 ms. Sample standard deviations ranged from 256 ms to 422 ms. The longest processing time for an individual query was 8.8 s. As above run times showed linear increases.

3) Discussion: As stated above the prime reason for conducting the simulation experiments was to assess growth of processing times with increasingly filled memory stores. A second aim of the experiments was to determine orders of magnitude of processing times under the different conditions. Precise processing times, which can vary considerably with hardware platform or parameter choices, are of much less importance for our purposes.

a) Insertion: With average execution times around 19 ms and only very little growth of the latter, ELM can account for the throughput generated by our robotic systems even when run on very moderate hardware. The fact that processing times varied even including some outliers as the global maximum can be tolerated for our applications.

b) Retrieval: With global averages around 130 ms and at most linear growth the processing times of retrieval operations are in general promising as well. There are, however, large differences between the processing times of the different query classes as well as within query classes (cf. fig. 8).

The comparison of processing times of queries using cues with one field or two fields specified shows one way to ameliorate retrieval times on average. Specifying more than one component in an event cue allows the query planner of the underlying DBMS to arrange sub-queries in a favourable way. But more components do also incur higher costs for parsing and planning query execution which explains the difference between the corresponding values for two and three components.

4) Conclusion: In summary, we can conclude that ELM can be used to support robotic applications with episodic-like memory over longer time spans even when run on moderate hardware.

C. Scaling ELM to larger scenarios

Above we have shown how ELM can be used to store experience in robotic applications running over longer periods of time. But what if the scenarios become more demanding? Probably the simplest but also a rather limited approach to scale the system is to scale up the hardware running ELM. More advanced approaches involve, e.g., multi-tier architectures of server systems in which nodes in one or more tiers can be scaled out. Hard- and software in support of such multi-tier architectures are available from many of the large commercial DBMS vendors. Especially for researchers interested in the construction of inexpensive prototype systems it is interesting to note that there are also freely available alternatives which are not restricted to the use with a specific DBMS. [55] describe C-JDBC, a database clustering middleware, which allows several database systems to be combined into a virtual database. The performance of C-JDBC clusters was reported to approximate a linear function of the number of nodes included in the cluster (under the TPC-W benchmark). Systems like these could be applied to scale ELM when neccessary.

VII. Conclusions

In this article we presented the development of an episodic-like memory system for cognitive robots. The system is conceptually inspired by its natural counterparts. At the same time, it also takes technical requirements in cognitive robotics into account. We discussed essential aspects of the scientific background in psychology and neuroscience. We pointed out central characteristics of episodic-like memory which served as useful guidelines of the assessment of memory systems in robotics and related fields. Moreover, these characteristics formed the basis for the design and implementation of our episodic-like memory system ELM. The characteristics were interpreted as requirements for our memory system. They were supplemented with requirements related to the intended use as a long-term memory in complex cognitive robot systems. Based on this set of requirements a formal framework was developed, which served as a basis for the lower-level design and implementation in a relational database with extensions for spatial data types. The resulting system reconciles requirements from different fields. Viewed as a structure, the memory provides a non-intrusive, integrated representation of events and their spatio-temporal contexts. It allows events to overlap or be nested, and, more generally, to form partonomic hierarchies. Different levels in a partonomic hierarchy represent different levels of granularity in the stored experience. Across all levels the framework retains a homogeneous notion of the spatio-temporal context of events and of events themselves. This homogeneity offers many advantages with respect to the processing of events as it enables parsimonious interfaces and processing principles. Viewed as a mechanism, the memory allows new memories to be acquired "in one shot". It also includes a flexible and modular framework for the recognition of complex events and the construction of partonomic hierarchies. The included retrieval mechanism allows integrated event structures to be retrieved using underspecified events as cues. Throughout the design and implementation of the memory, efficiency and scalability were taken into consideration to enable its use as a long-term memory. In comparison to the approaches discussed in section II our approach thus overcomes the problems with domain- or purpose-dependency noted for other approaches, the rigidity of their event (content) type systems, and their limited scalability.

We described the integration of the memory system with a cognitive robot architecture called Explorer which was developed in the CoSy and CogX projects. We demonstrated how properties of CAST and the Spatial subarchitecture were used to develop C-ELM, an additional integration layer for ELM, which is non-intrusive, easy to use and allows a flexible reconfiguration of the resulting extended Explorer system.

While the fulfillment of most requirements for episodic-like memory systems can be immediately verified from ELM's design, the same is not possible for performance-related requirements. For this reason we ran several experiments in real and in simulated environments. In these experiments ELM's performance was tested under various conditions and with memory stores filled with up to 100k events. The results showed that, even on hardware as moderate as a notebook, ELM runs efficient enough for robotic applications involving hours of consecutive experience. Finally, we outlined approaches to scale the system to much larger scenarios.

Appendix A
Formal design

In this appendix we describe the formal aspects of our design in more detail. This formalisation provides the basis for the implementation discussed above.

a) General representation of events:

Definition 1. Set of event types, event type, (direct) sub-type. Let Θ be a strict partially ordered set with strict partial ordering relation $\prec$. Θ is called the set of event types and any element $\theta \in \Theta$ is called an event type. Let $\theta_1, \theta_2 \in \Theta$ be event types. θ_1 is called a direct sub-type of θ_2 if $\theta_1 \prec \theta_2$. θ_1 is called a sub-type of θ_2 if $\theta_1 \prec^* \theta_2$ where $\prec^*$ denotes the transitive closure of $\prec$.

Definition 2. Set of all event-specific features, feature names, feature values. Let F_N, F_V be (finite or infinite) sets. F_N is called the set of all event-specific feature names.

F_V is called the set of all event-specific feature values. The cross-product $F = F_N \times F_V$ of F_N and F_V is termed the set of all event-specific features. Given a set $f \subseteq F$ of event-specific features $\pi_1(f) \subseteq F_N$ denotes the projection to the set of feature names in f. Likewise, $\pi_2(f) \subseteq F_V$ denotes the projection to the set of feature values in f.

Definition 3. Event, set of all events. An event is a triple $e = (c, l, t)$ with $c = (\theta, b, f), l = P_2$, and $t = [t_{start}, t_{end}]$. $\theta \in \Theta$ is an event type, $b \in \{0,1\}^*$ is an event-specific binary string, $f \subseteq F$ is a set of event-specific features, $l = P_2$ is a polygon in two dimensions and $t = [t_{start}, t_{end}] \subseteq \mathbf{R}^2$ denotes a non-empty interval of time. **E** denotes the set of all events.

In addition to the requirements already mentioned this definition also complies with the overlap requirement. Events can occur simultaneously or within the same area.

b) Atomic events, complex events and sub-events: The event in the example above was simple or 'atomic.' There are no constituents which also have event character in the sense of an episodic-like memory item. Imagine another situation in which you see an empty glass on a table, try to grasp it but lose hold. You see it falling and breaking. This sequence of individual events makes up a more complex event to which you might refer as 'I broke the glass'. To represent such nesting of events a sub-event relation is introduced:

Definition 4. Sub-event relation, sub-events, super-events. Let $E \subseteq \mathbf{E}$ be a set of events. A sub-event relation is a strict partial ordering relation $\subset_E$ on E. For all pairs $(e_1, e_2) \in \subset_E$, e_1 is called a sub-event of e_2 and e_2 is called a super-event of e_1. $subevents_{E,\subset_E}(e) := \{e' \in E \mid e' \subset_E e\}$ and $superevents_{E,\subset_E}(e) := \{e' \in E \mid e \subset_E e'\}$ denote the set of sub-events or super-events of e, respectively.

This relation does not only touch on the nesting requirement but at the same time enables another variant of overlapping of events: overlap in sub-events.

The definition of the sub-event set $subevents_{E,\subset_E}(e)$ for a given event e depends both on the set of events E and the sub-event relation $\subset_E$ considered. In fact, these two provide all the necessary information about the contents of an episodic-like memory store (or episodic-like memory, for short):

Definition 5. Episodic-like memory, set of all episodic-like memories.An episodic-like memory (store) is a pair $M = (E, \subset_E)$ where E is a set of events and $\subset_E$ denotes a sub-event relation on E. **M** denotes the set of all episodic-like memories (episodic-like memory stores).

On this basis $subevents_M(e)$ will be used as a shorthand for $subevents_{E,\subset_E}(e)$ from now on.

Definition 6. Atomic event, complex event. Let $e \in E$ be an event in the episodic-like memory $M = (E, \subset_E)$. e is termed atomic if $subevents_M(e) = \emptyset$. Otherwise, e is termed a complex event.

According to the definition complex events are events with a non-empty set of sub-events. In order to arrive at a notion and representation of such events which is both consistent in itself and compatible with our common sense notion of a complex event we next define how the location and time of an event relate to those of the sub-events.

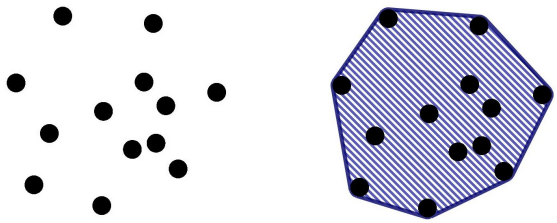

Fig. 9. A set of circles in the plane and their convex hull.

In the case of time, for example, we would expect a super-event's start time to equal the time of its earliest sub-event and to end with the latest sub-event. More formally:

Definition 7. Time of a complex event. Let e be a complex event at time $[t_1, t_2]$ with sub-events $s_1, ..., s_n$ and let $[t_{i1}, t_{i2}]$ be the time of s_i. Then the time of e is defined as: $t_1 := \min\{t_{i1} \mid 1 \leq i \leq n\}$ and $t_2 := \max\{t_{i2} \mid 1 \leq i \leq n\}$.

Similarly, the location of an event should cover the locations of its sub-events. Consider the set of circles on the left of Figure 9. If they were locations of sub-events of a complex event e then the enclosed area on the right could be called the location of e. It is derived from a set of locations by computing their convex hull which is defined below:

Definition 8. Convex set. Let C be a set of points in a real or complex vector space. C is called convex if and only if

$$\forall t \in [0,1]\ \forall x, y \in C : (1-t)x + ty \in C$$

Definition 9. Convex hull. Let X be a set of points in a real or complex vector space. The convex hull $H_{convex}(X)$ is the minimal convex set containing X.

Based on convex hulls the location of a complex event can now be formally defined:

Definition 10. Location of a complex event. Let e be a complex event with sub-events $s_1, ..., s_n$ and let l_i be the location of s_i $(1 \leq i \leq n)$. The location l of e is the convex hull of the union of the sub-event locations l_i:

$$l := H_{convex}(\bigcup_{1 \leq i \leq n} l_i)$$

It is worth to note that the time of a complex event as defined above equals the convex hull of the sub-events' times (viewed as point sets in $\mathbf{R}^1$). This allows for one homogeneous notion of time and location of complex events.

Moreover, note that the conceptualisation of an event, resulting from the inclusion of complex events, now also subsumes the idea of an episode. The consistent incorporation of the episode notion into our general event concept is possible since episodes share the characteristics of 'simple' events. This will allow units of experience on all levels of granularity to be

processed using the same interfaces and based on the same principles.

1) Operations on episodic-like memories: Above we defined the structures of events and episodic-like memories. Next, we define operations on theses structures.

a) Event Insertion: We first consider the simple operation of inserting events, as units of experience, into a memory.

Definition 11. Event insertion. Let $M = (E, \subset_E)$ be an episodic-like memory. The insertion operation for M is defined by:

$$\begin{aligned} ins_M : (\mathbf{E} \setminus E) \times 2^E &\rightarrow \mathbf{M} \\ (e, S) &\mapsto (E \cup \{e\}, \subset_E \cup \{(s_i, e) \mid s_i \in S\}) \end{aligned}$$

Given an episodic-like memory, an event not yet contained in the latter and a set of designated sub-events (already in the given memory) ins_M yields an episodic-like memory which also includes the new event with the appropriate new sub-event relation. This update takes place "in one shot." ins_M can be applied to both atomic and complex events. Below we introduce an update operation specifically for complex events.

b) Complex event recognition: Complex event recognition is a process in which experience is structured by forming meaningful, lower-granular units of several events which also have event character. These lower-granular units are complex events. The events which they combine are their sub-events. We define the process of complex event recognition as the application of a recognition function to a memory M.

Definition 12. Complex event recognition function. A complex event recognition function is a function $R : \mathbf{M} \rightarrow \mathbf{M}$ which maps an episodic-like memory $M = (E, \subset_E) \in \mathbf{M}$ to an episodic-like memory $M' = (E', \subset_{E'}) \in \mathbf{M}$ such that $E \subseteq E'$ and $\forall e' \in E' \setminus E \, \exists \, e \in E : e \in subevents_{M'}(e')$. An episodic-like memory M is termed saturated with respect to R if $R(M) = M$. For any (possibly unsaturated) episodic-like memory M, $R^*(M)$ denotes its saturated form.

At a finer level of detail the application of a complex event recognition function to an episodic-like memory M can be viewed as the application of (one or more) complex event recognisers to M and the extension of M with their results.

Definition 13. Complex event recogniser. A complex event recogniser is a function $r : \mathbf{M} \rightarrow \mathbf{E} \times 2^{\mathbf{E} \times \mathbf{E}}$ which maps an episodic-like memory $M = (E, \subset_E) \in \mathbf{M}$ to a pair $\Delta M = (\Delta E, \Delta \subset_E)$ such that $E \cap \Delta E = \emptyset$ and $\forall (e_1, e_2) \in \Delta \subset_E$: $e_1 \in E \wedge e_2 \in \Delta E$.

Thus it is possible to define one complex event recogniser per complex event type or split "responsibilities" along other lines. This possibility of viewing the results of the complex event recognition process as a superposition of complex event recogniser applications and insertions can be exploited for the design of the corresponding interfaces, cf. sub-section IV-E.

c) Recollection : Recollection is modelled with a cued recall approach in which the cues are underspecified events. In these event cues any component can be left unspecified. Thus event cues represent templates against which stored events can be matched. For any specified component, values can be matched with the corresponding values of stored events in several ways (which depend on the respective component).

Definition 14. Event cue. Let $M = (E, \subset_E)$ be an episodic-like memory, $I = \{\theta, b, f, t, l, S_{\subset_E}, S_{\supset_E}\}$ an index set of cardinality 7 and $I' \subseteq I$. An event cue is a function

$$\begin{aligned} Q_M : \mathbf{E} &\rightarrow \{0, 1\} \\ e &\mapsto \bigwedge_{i \in I'} C_{i, \sim_i, X_i} \end{aligned}$$

mapping an event

$$e = ((\theta_e, b_e, f_e), t_e, l_e)$$

to a conjunction of conditions of the form

$$C_{i, \sim_i, X_i} := \begin{cases} \theta_e \sim_i X_i & \text{for } i = \theta \\ b_e \sim_i X_i & \text{for } i = b \\ f_e \sim_i X_i & \text{for } i = f \\ t_e \sim_i X_i & \text{for } i = t \\ l_e \sim_i X_i & \text{for } i = l \\ subevents_M(e) \sim_i X_i & \text{for } i = S_{\subset_E} \\ superevents_M(e) \sim_i X_i & \text{for } i = S_{\supset_E} \end{cases}$$

with X_i an event type, a binary string or a set and relations

$$\sim_i \in R_i$$

where

$$\begin{aligned} R_\theta &= \{=, \prec^*\} \\ R_b &= \{=\} \\ R_t = R_l = R_{S_{\subset_M}} = R_{S_{\supset_M}} &= \{=, \subseteq, \supseteq, \cap_R\} \end{aligned}$$

In the context of the relation sets R_i the relations $=, \subseteq, \supseteq$ are defined as usually, $\prec^*$ is the sub-type relation as defined above. The relation $\cap_R$ is defined as follows:

$$\forall A, B : A \cap_R B :\Leftrightarrow A \cap B \neq \emptyset$$

The set R_f, finally, contains the relations $=, \subseteq, \supseteq, \cap_R$ as well as modified versions of the same which allow matching to be restricted to feature names

$$R_f = \{=, \subseteq, \supseteq, \cap_R\} \cup \left\{=_{\pi_1}, \subseteq_{\pi_1}, \supseteq_{\pi_1}, \cap_{R_{\pi_1}}\right\}$$

where for any event-specific feature sets f, f' and any relation $\sim \in \{=, \subseteq, \supseteq, \cap_R\}$ the modified relation $\sim_{\pi_1}$ is defined by

$$f \sim_{\pi_1} f' :\Leftrightarrow \pi_1(f) \sim \pi_1(f')$$

Definition 15. Retrieval result set Let $M = (E, \subset_E)$ be an episodic-like memory and $Q_M : \mathbf{E} \rightarrow \{0, 1\}$ be an event cue. The set $ResSet(Q_M) := \{e \in E \mid Q_M(e)\}$ is called the retrieval result set under event cue Q_M.

The retrieval result set $ResSet(Q_M)$ for a given event cue Q_M contains all the events from memory M which match Q_M, not only partial information stemming from events. Thus the formalisations of event cues and result sets fulfil the structure requirement.

REFERENCES

[1] D. Stachowicz, "Episodic-like memory for cognitive robots," Master's thesis, Department of Computer Science, Saarland University / DFKI GmbH, December 2009.

[2] L. R. Squire, "Memory systems of the brain: a brief history and current perspective," *Neurobiology of Learning and Memory*, vol. 82, no. 3, pp. 171–177, Nov 2004.

[3] A. Baddeley, "The concept of episodic memory," *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, vol. 356, no. 1413, pp. 1345–1350, Sep 2001.

[4] D. O. Hebb, *Organization of Behaviour*. New York: Wiley, 1949.

[5] W. B. Scoville and B. Milner, "Loss of recent memory after bilateral hippocampal lesions," *Journal of Neurology, Neurosurgery and Psychiatry*, vol. 20, no. 1, pp. 11–21, Feb 1957.

[6] J. D. Gabrieli, N. J. Cohen, and S. Corkin, "The impaired learning of semantic knowledge following bilateral medial temporal-lobe resection," *Brain and Cognition*, vol. 7, no. 2, pp. 157–177, Apr 1988.

[7] G. O'Kane, E. A. Kensinger, and S. Corkin, "Evidence for semantic learning in profound amnesia: an investigation with patient H.M." *Hippocampus*, vol. 14, no. 4, pp. 417–425, 2004.

[8] C. Blakemore, *Mechanics of the mind*. Cambridge: Cambridge University Press, 1977.

[9] L. R. Squire, B. Knowlton, and G. Musen, "The structure and organization of memory," *Annual Review of Psychology*, vol. 44, pp. 453–495, 1993.

[10] E. Tulving, "Episodic and semantic memory," in *Organization of Memory*, E. Tulving and W. Donaldson, Eds. Academic Press, 1972, pp. 381–403.

[11] ——, *Elements of episodic memory*. Clarendon, 1983.

[12] ——, "Episodic memory: from mind to brain," *Annual Review of Psychology*, vol. 53, pp. 1–25, 2002.

[13] N. S. Clayton and A. Dickinson, "Episodic-like memory during cache recovery by scrub jays," *Nature*, vol. 395, no. 6699, pp. 272–274, Sep 1998.

[14] N. S. Clayton, T. J. Bussey, and A. Dickinson, "Can animals recall the past and plan for the future?" *Nature Reviews. Neuroscience*, vol. 4, no. 8, pp. 685–691, Aug 2003.

[15] N. S. Clayton, D. P. Griffiths, N. J. Emery, and A. Dickinson, "Elements of episodic-like memory in animals," *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, vol. 356, no. 1413, pp. 1483–1491, Sep 2001.

[16] N. S. Clayton and J. Russell, "Looking for episodic memory in animals and young children: prospects for a new minimalism," *Neuropsychologia*, vol. 47, no. 11, pp. 2330–2340, Sep 2009.

[17] W. Dodd and R. Gutierrez, "The role of episodic memory and emotion in a cognitive robot," in *Proceedings of 14th Annual IEEE International Workshop on Robot and Human Interactive Communication (RO-MAN), Nashville, TN, August 13-15, 2005*, 2005, pp. 692–697.

[18] S. Jockel, D. Westhoff, and J. Zhang, "EPIROME - A novel framework to investigate high-level episodic robot memory," in *Proc. IEEE International Conference on Robotics and Biomimetics ROBIO 2007*, 2007, pp. 1075–1080.

[19] S. Jockel, M. Weser, D. Westhoff, and J. Zhang, "Towards an episodic memory for cognitive robots," in *Proceedings of the 6th International Cognitive Robotics Workshop at 18th European Conference on Artificial Intelligence (ECAI), Patras, Greece, July 21-22, 2008*. IOS Press, 2008, pp. 68–74.

[20] A. Nuxoll and J. E. Laird, "A cognitive model of episodic memory integrated with a general cognitive architecture," in *Proceedings of the Sixth International Conference on Cognitive Modeling - ICCM 2004*, 2004, pp. 220–225.

[21] J. L. Kolodner, *Case–Based Reasoning*. Morgan Kaufmann, 1993.

[22] A. Nuxoll and J. E. Laird, "Extending cognitive architecture with episodic memory," in *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI)*. AAAI Press, 2007, pp. 1560–1564.

[23] N. S. Kuppuswamy, S.-H. Cho, and J.-H. Kim, "A cognitive control architecture for an artificial creature using episodic memory," in *Proceedings of the SICE-ICASE International Joint Conference, 2006*. IEEE, 2006, pp. 3104–3110.

[24] S. A. Vere and T. W. Bickmore, "A basic agent," *Computational Intelligence*, vol. 6, pp. 41–60, 1990.

[25] C. Brom, K. Pešková, and J. Lukavsky, "What does your actor remember? towards characters with full episodic memory," in *Proceedings of the 4th international conference on Virtual storytelling: using virtual reality technologies for storytelling*, 2007, pp. 89–101.

[26] W. Ho, K. Dautenhahn, and C. Nehaniv, "Computational memory architectures for autobiographic agents interacting in a complex virtual environment: A working model," *Connection Science*, vol. 20, no. 1, pp. 21–65, 2008.

[27] H. Jacobsson, N. Hawes, G.-J. Kruijff, and J. Wyatt, "Crossmodal content binding in information-processing architectures," in *Proceedings of the 3rd ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, Amsterdam, The Netherlands, March 12–15 2008.

[28] R. Bayer and E. M. McCreight, "Organization and maintenance of large ordered indexes," *Acta Informatica*, vol. 1, no. 3, pp. 173–189, Feb. 1972, also published in/as: ACM SIGFIDET 1970, pp.107–141.

[29] D. Comer, "The ubiquitous B-tree," *ACM Computing Surveys*, vol. 11, no. 2, pp. 121–137, Jun. 1979.

[30] A. Guttman, "R-trees: a dynamic index structure for spatial searching," *SIGMOD Record (ACM Special Interest Group on Management of Data)*, vol. 14, no. 2, pp. 47–57, 1984.

[31] T. K. Sellis, N. Roussopoulos, and C. Faloutsos, "The R+-tree: A dynamic index for multi-dimensional objects," in *VLDB'87, Proceedings of 13th International Conference on Very Large Data Bases, September 1-4, 1987, Brighton, England*, P. M. Stocker, W. Kent, and P. Hammersley, Eds. Morgan Kaufmann, 1987, pp. 507–518.

[32] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: an efficient and robust access method for points and rectangles," in *SIGMOD '90: Proceedings of the 1990 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM, 1990, pp. 322–331.

[33] I. Kamel and C. Faloutsos, "Hilbert R-tree: An improved R-tree using fractals," in *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB'94)*, J. B. Bocca, M. Jarke, and C. Zaniolo, Eds. Santiago de Chile, Chile: Morgan Kaufmann, 12–15 1994, pp. 500–509.

[34] L. Arge, M. de Berg, H. J. Haverkort, and K. Yi, "The Priority R-tree: a practically efficient and worst-case optimal R-tree," in *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: ACM, 2004, pp. 347–358.

[35] J. M. Hellerstein, J. F. Naughton, and A. Pfeffer, "Generalized search trees for database systems," in *Proceedings of the 21st International Conference on Very Large Data Bases (VLDB'95)*. San Francisco, Ca., USA: Morgan Kaufmann, Sep. 1995, pp. 562–573.

[36] "Open GIS consortium, Inc., OpenGIS Simple Features Specification For SQL Revision 1.1, OpenGIS Project Document 99-049, May 5, 1999, http://www.opengis.org/techno/specs/99-049.pdf," 1999.

[37] P. P.-S. S. Chen, "The entity-relationship model: Toward a unified view of data," *ACM Transactions on Database Systems*, vol. 1, no. 1, pp. 9–36, 1976.

[38] N. Hawes, J. Wyatt, and A. Sloman, "An architecture schema for embodied cognitive systems," University of Birmingham, School of Computer Science, Tech. Rep. CSR-06-12, November 2006.

[39] N. Hawes, M. Zillich, and J. Wyatt, "BALT & CAST: Middleware for cognitive robotics," in *Proceedings of IEEE RO-MAN 2007*, August 2007, pp. 998 – 1003.

[40] N. Hawes, A. Sloman, and J. Wyatt, "Towards an empirical exploration of design space," in *Evaluating Architectures for Intelligence: Papers from the 2007 AAAI Workshop*, G. A. Kaminka and C. R. Burghart, Eds. Vancouver, Canada: AAAI Press, July 2007, pp. 31 – 35.

[41] N. Hawes, J. Wyatt, and A. Sloman, "Exploring design space for an integrated intelligent system," in *Research and Development in Intelligent Systems XXV: Proceedings of AI-2008, The Twenty-eighth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*, M. Bramer, F. Coenen, and M. Petridis, Eds. Cambridge, England: Springer, December 2008.

[42] N. Hawes and J. Wyatt, "Benchmarking the influence of information-processing architectures on intelligent systems," in *Proceedings of the Robotics: Science & Systems 2008 Workshop: Experimental Methodology and Benchmarking in Robotics Research*, June 2008.

[43] H. Zender, P. Jensfelt, Óscar Martínez Mozos, G.-J. M. Kruijff, and W. Burgard, "An integrated robotic system for spatial understanding and situated interaction in indoor environments," in *Proceedings of the 22nd Conference on Artificial Intelligence (AAAI-07)*, Vancouver, British Columbia, Canada, July 2007, pp. 1584–1589.

[44] H. Zender, "Learning spatial organization through situated dialogue," Master's thesis, Department of Computational Linguistics, Saarland University, Saarbrücken, Germany, August 2006.

[45] H. Zender and G.-J. M. Kruijff, "Multi-layered conceptual spatial mapping for autonomous mobile robots," in *Control Mechanisms for Spatial Knowledge Processing in Cognitive / Intelligent Systems*, ser.

Papers from the AAAI Spring Symposium, H. Schultheis, T. Barkowsky, B. Kuipers, and B. Hommel, Eds., vol. Technical Report SS-07-01. Menlo Park, CA, USA: AAAI Press, March 2007, pp. 62–66.

[46] E. A. Topp and H. I. Christensen, "Tracking for following and passing persons," in *International Conference on Intelligent Robotics and Systems (IROS)*, Edmundton, Canada, Aug. 2005, pp. 70–77.

[47] G.-J. M. Kruijff, H. Zender, P. Jensfelt, and H. I. Christensen, "Situated dialogue and understanding spatial organization: Knowing what is where and what you can do there," in *Proceedings of the 15th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN 2006)*, Hatfield, Hertfordshire, UK, September 2006, pp. 328–333.

[48] ——, "Situated dialogue and spatial organization: What, where…and why?" *International Journal of Advanced Robotic Systems, Special Issue on Human-Robot Interaction*, vol. 4, no. 1, pp. 125–138, March 2007.

[49] H. Zender, P. Jensfelt, and G.-J. M. Kruijff, "Human- and situation-aware people following," in *Proceedings of the 16th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN 2007)*, Jeju Island, Korea, August 2007, pp. 1131–1136.

[50] G.-J. M. Kruijff, H. Zender, P. Jensfelt, and H. I. Christensen, "Clarification dialogues in human-augmented mapping," in *Proceedings of the 2006 ACM Conference on Human-Robot Interaction (HRI 2006)*, Salt Lake City, UT, USA, March 2006, pp. 282–288.

[51] M. Brenner, "Continual collaborative planning for mixed-initiative action and interaction (short paper)," in *Proceedings of 7th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Padgham, Parkes, Müller, and Parsons, Eds., Estoril, Portugal, 2008.

[52] M. Brenner and I. Kruijff-Korbayová, "A continual multiagent planning approach to situated dialogue," in *Proceedings of the 12th Workshop on the Semantics and Pragmatics of Dialogue (Semdial)*, London, UK, 2008.

[53] M. Brenner and B. Nebel, "Continual planning and acting in dynamic multiagent environments," *Journal of Autonomous Agents and Multiagent Systems*, vol. 19, no. 3, pp. 297–331, 2009.

[54] D. Stachowicz, N. Hawes, and G.-J. M. Kruijff, "Adding episodic-like memory to an event-based system," in *IROS Workshop on Event-Based Systems for Robotics*, D. Brugali, S. Wrede, and I. Lütkebohle, Eds., September 2009.

[55] E. Cecchet, J. Marguerite, and W. Zwaenepoel, "C-JDBC: Flexible database clustering middleware," in *USENIX Annual Technical Conference, FREENIX Track*. USENIX, 2004, pp. 9–18.

Dennis Stachowicz studied computer science and psychology in Saarbrücken and Edinburgh. He received a Diplom (MSc.) degree in computer science from Saarland University in 2010. He conducted the research for his thesis at the German Research Center for Artificial Intelligence (DFKI GmbH) under the supervision of Geert-Jan M. Kruijff.

Geert-Jan M. Kruijff received an MSc degree from the University of Twente (Enschede, the Netherlands) in 1995 , and a PhD in computer science from Charles University (Prague, Czech Republic) in 2001. Since 2004 he is a senior researcher and project leader at the German Research Center for Artificial Intelligence (DFKI GmbH). His research focus is on spoken dialogue in human-robot interaction.