A Framework for Goal Generation and Management*

Marc Hanheide and **Nick Hawes** and **Jeremy Wyatt**
Intelligent Robotics Lab, School of Computer Science
University of Birmingham, UK

Moritz Göbelbecker and **Michael Brenner**
Institut für Informatik
Albert-Ludwigs-Universität
Freiburg, Germany

Kristoffer Sjöö and **Alper Aydemir** and **Patric Jensfelt**
Centre for Autonomous Systems
Royal Institute of Technology, Stockholm, Sweden

Hendrik Zender and **Geert-Jan M. Kruijff**
Language Technology Lab
German Research Center for Artificial Intelligence (DFKI)
Saarbrücken, Germany

Abstract

Goal-directed behaviour is often viewed as an essential characteristic of an intelligent system, but mechanisms to generate and manage goals are often overlooked. This paper addresses this by presenting a framework for autonomous goal generation and selection. The framework has been implemented as part of an intelligent mobile robot capable of exploring unknown space and determining the category of rooms autonomously. We demonstrate the efficacy of our approach by comparing the performance of two versions of our integrated system: one with the framework, the other without. This investigation leads us conclude that such a framework is desirable for an integrated intelligent system because it reduces the complexity of the problems that must be solved by other behaviour-generation mechanisms, it makes goal-directed behaviour more robust in the face of a dynamic and unpredictable environments, and it provides an entry point for domain-specific knowledge in a more general system.

Introduction

For an integrated system to be described as intelligent it is usually important for it to display goal-directed behaviour. The field of AI is rich in approaches for determining what actions a system should perform next to achieve a particular goal. However, a comparably small amount of time and effort has been expended on investigating approaches for *generating* and *managing* the goals of an intelligent system. In this paper we argue that such mechanisms can have a positive impact on the behaviour of a goal-directed system, and we support our position using data gathered from an integrated robot system that can be run both with and without goal generation and management capabilities.

Our research is motivated by the goal of implementing intelligent robot assistants which can perform tasks for, and with, humans in every-day environments. The complexity and open-ended requirements of the scenarios such robots must take part in has caused us to design our systems so that they are capable of generating their own behaviour at run-time, rather than having it determined explicitly by a programmer at design-time. The remainder of this paper assumes a general approach in which a system explicitly plans about how to bring about a particular state before executing the required actions.

Goal-directed behaviour requires a system to change the world in order to attain a predetermined state-of-affairs. We refer to the disposition to bring about a particular state-of-affairs as a *drive*. A waiter working in a busy restaurant may have a drive to make as much money in tips as possible, which could produce drives to keep his customers happy by making sure their food arrives quickly, that they do not run out of drinks, and that they are happy with their food. This example shows that a system's collection of drives may have internal structure. For this work we shall ignore this structure but instead focus on the problem of how a system's drives ultimately come to be realised in behaviour. For behaviour generation approaches such as planning to be applied, it is not enough to have an abstract specification of the state-of-affairs to bring about (e.g. making sure customers are happy with their food). Instead, these approaches typically require a description of a concrete state-of-affairs (i.e. an instance of the more general type) which can be brought about from the current state (e.g. that the customers sat at Table 12 are happy with their food). Following planning terminology we refer to this description of a concrete state-of-affairs as a *goal*. We refer to the process of producing a goal from a drive as *goal generation*. It is important that systems are able to perform goal generation in domains where all possible goals cannot be determined in advance. Goal generation allows a system to be autonomous in such domains by providing the ability to react to state changes where existing drives should give rise to new goals (e.g. unexpected customers entering the restaurant).

Our busy waiter does not just have the problem of generating goals; he must also choose which goals to pursue from the collection of goals he has previously generated. For example, for each table of customers he might have a goal to take a drink or food order, enquire if they're enjoying their meal, clear plates from the table or bring their bill. Some goals may ultimately contribute to the waiter's original drive more (e.g. goals relating to a table of particularly generous customers he knows from a previous visit) and some less (e.g. goals from customers who didn't tip last time). Some goals may be quicker or easier to achieve (e.g. taking a drinks order), and some less so (e.g. bringing the food for a large birthday party). Some goals may be achiev-

*The research reported here was performed in the EU FP7 IP "CogX: Cognitive Systems that Self-Understand and Self-Extend" (ICT-215181)

able together and some not (e.g. depending on whether they fully occupy the waiter's arms), and some may have more pressing deadlines than others. Reasoning about constraints such as these in order to determine which collection of goals should be pursued is a process we refer to as *goal management*. It is important to realise that goal management is not a monolithic process in which a set of goals and constraints are considered and a subset selected for action; changes in a system's state (including new goals and observations) may require the reconsideration of any previously selected goals. When you consider planning and execution failures in such a framework, it becomes apparent that goal management can also play a part in increasing system robustness, in addition to prioritising which goals should be pursued.

Goal Generation & Management Framework

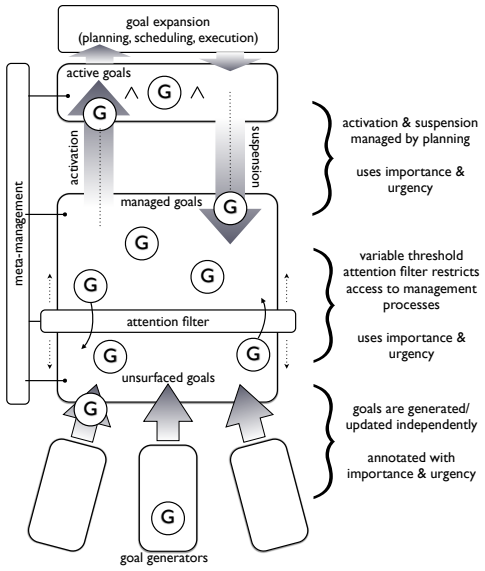


Figure 1: The goal generation and management framework.

We would like to endow our integrated systems with the goal generation and management (GGM) capabilities of our fictional waiter. To this end we have built on the work of Beaudoin and Sloman (1993), to produce the design illustrated in Figure 1. This design is a general framework, or schema, for an architecture for goal generation and management. It specifies a collection of interacting elements which must be included in any instantiation of the framework, although the precise details of the instantiation will inevitably vary between instances. The elements of the framework are described in more detail below.

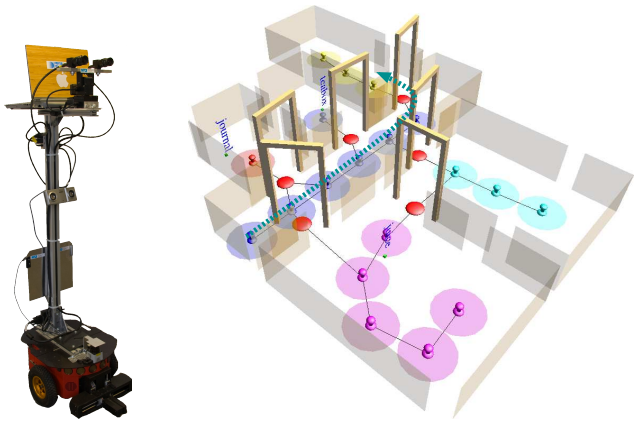
At the bottom of the framework, a system's drives are encoded as multiple *goal generators*. These are concurrently active processes which monitor the system's state (both the external world and internal representations) and produce *goals* to satisfy the system's drives. Generators can also remove previously generated goals if they are judged to no longer be appropriate. In this manner we can say that the system's drives are encoded in the goal generators (either explicitly or implicitly). We work from the assumption that as a goal passes up through the framework from a generator and influences a system's behaviour, it is inspected by processes of greater and greater computational complexity. Therefore the lower strata of the framework exist to protect these processes (and thus overall system resources) from having to consider more goals than is necessary (where this could be a contingent judgement). The main mechanism in the framework for protecting the management processes is the *attention filter*. This is a coarse barrier which uses simple, fast processing to let some some goals through to the management mechanisms whilst blocking others. Goals which make it through this filter are described as *surfaced*, thus the goals which fail to pass the filter are referred to as *unsurfaced*. A collection of management processes determine which of the surfaced goals should be combined to serve as the goals being actively pursued by the system. If a goal is selected in this way we describe it as *activated*. If a goal is removed from the set of goals being pursued by the system we refer to it as *suspended*.

In order to fulfil their roles, the filtering and management processes require information on which to base their decisions. Following the original work of Beaudoin and Sloman, the framework requires that goal generators annotate each goal with a description of the goal's *importance* and *urgency*, and keep these descriptions up to date as long as the goal exists. Importance should reflect the significance of the goal to the agent (as motivated by the related drive). Urgency should reflect the necessity of achieving the goal sooner rather than later. As we shall see later, producing importance and urgency descriptions for use in such a framework is a problem in itself. In addition to these descriptions, the framework allows the management processes to use whatever approaches are required to select and maintain a set of active goals. Perhaps the minimum requirements on these processes is the ability to check whether a goal, or collection of goals, can be achieved (thus positing planning as a goal activation, as well as achievement, mechanism).

Dora the Explorer

To explore the implications of this design, we have implemented the framework described above as part of a intelligent mobile robot called *Dora the Explorer*. The following sections describe the domain in which Dora operates, the subsystems which have been integrated to create Dora, and the implementation of the GGM framework in this context.

Domain Dora is intended as a research precursor to a mobile service robot able to interact with, and perform tasks for, humans in indoor environments. Within this broad context, our particular interest lies in providing Dora with the capacity to model the limits of its own knowledge, and to plan and execute actions to extend its knowledge beyond these limits. To this end we have been investigating how Dora can represent its *knowledge gaps* (i.e. the things that it knows it doesn't know) and how it can fill these gaps. A service robot in an office environment could have a wide variety of knowledge gaps. For example, it may know that it does not



(a) Dora.

(b) A map constructed during a system run. The circles represent places Dora has visited.

Figure 2: Robot and spatial environment.

know the names, properties or functions of particular objects; the names or locations of certain people; or how to perform the actions required of it (e.g. picking up an unknown object). Although we ultimately wish to address as many of these gaps as possible, in this work we focus on two types of knowledge gap: gaps in Dora's knowledge about space, and gaps in Dora's knowledge of the function of rooms. This focus has produced an integrated robot system tailored to the problems entailed by these knowledge gaps, rather than the entire Dora domain, The design and implementation of this system is summarised in the following paragraphs.

System The Dora robot, pictured in Figure 2(a) is based on a Pioneer 3DX with a custom super structure. The system architecture for Dora is based on the PECAS architecture (Hawes et al. 2009). PECAS divides components within a system into *subarchitectures* (SAs). SAs are collections of components plus a shared *working memory* (WM) which serves as the communication channel between components (similar to a blackboard architecture). Each WM therefore contains the representations which are shared between components in that SA. As PECAS suggests that SAs be determined by function, Dora contains SAs for the major functions required in its domain: spatial mapping; conceptual mapping; vision; and communication. These are coordinated by the SAs for planning and cross-modal fusion which are part of PECAS itself.

As our work relies heavily on planning, we will briefly describe how the planning SA in PECAS operates. Planning state is provided by the cross-modal fusion SA, which fuses representations from other SAs to produce a single, unified view of the system's knowledge. Planning goals are written to the planning WM in a format known as MAPL, the planning language used by PECAS's planner (Brenner and Nebel 2009). The planner itself is a multi-agent *continual planner*, capable of replanning and execution monitoring. It is essential that a continual approach is used when planning in interactive robot systems such as Dora. Such an approach is required to handle changes in state, changes in goals, and the sensing and execution failures which naturally occur in such systems. In PECAS the planner is notified of such events via changes to WM content. These changes occur asynchronously with respect to planner operation. PECAS, and any additional goal generation and management mechanisms, must be robust in the face of these events.

Dora has two SAs which contribute to its understanding of space: the spatial mapping and conceptual mapping SAs. The former of these interprets sensor data (primarily laser scans and odometry) to perform simultaneous localisation and mapping (SLAM), enabling the robot to determine its position in a local metric map. On top of these local metric maps, the spatial SA constructs a representation based on small connected regions within the the world called *places* (Pronobis et al. 2009). These places represent a first order abstraction of the maps generated during SLAM and are the lowest level of spatial representation available to other SAs within the system. Furthermore, the spatial SA detects gaps in its spatial knowledge. So-called *placeholders* associated with free space represent potential places that could be explored further. The spatial SA also provides functionality for collision-free robot movement within both local metric maps and the place graph. A map produced by the spatial SA (during an experiment) can be seen in Figure 2(b). The conceptual mapping SA allows Dora to reason and abstract over entities which can appear in the maps produced by the spatial SA. It clusters collections of places which are bounded by doors into room representations. It also contains knowledge from the OpenMind Indoor Common Sense database[1] which allows it to reason from the presence of objects in a room to the possible functional categories which could be assigned to the room (e.g. the presence of a kettle in a room may support the inference that the room is a kitchen or coffee room).

Dora is able to use this functionality to determine possible categories for rooms. Dora's vision SA contains an object recogniser which uses the Ferns algorithm (Özuysal, Fua, and Lepetit 2007) to identify known objects in the images it receives from Dora's cameras. The process of choosing where to capture images from in order to find objects is known as *active visual search* (AVS). Dora performs AVS by sampling the space of views of object-containing space until a coverage threshold is met. Dora is designed to assume that all non-free space on its spatial map may potentially contain objects (forcing it to look at desks, worktops and shelves in addition to featureless walls). This requires that a room must be adequately mapped prior to AVS.

Dora is implemented in C++ and Java using the CAS Toolkit (Hawes and Wyatt 2010). It has been run on at least eight different robots (derived from Pioneer 3s) at six institutions[2]. It is composed of 28 major components. The majority of the code is available under open source license.

Current Implementation of Framework The framework described earlier has been implemented in Dora as an extension to the PECAS planning SA. Dora includes two goal generator components, one for each type of knowledge gap

[1] Available from `http://openmind.hri-us.com`.

[2] A video of Dora running can be viewed at `http://cogx.eu/results/dora/`.

corresponding to placeholders and yet uncategorized rooms. These components monitor representations on WMs and then generate goal representations on the WM in the planning SA. As Dora is primarily concerned with filling gaps in its knowledge, the importance measure proposed by the GGM framework is implemented using a heuristic measure of the *information gain* a particular goal may provide for Dora. Goals are also annotated with a heuristic value representing the estimated cost (usually in terms of time used) of achieving them. We currently do not use urgency measures as there are no meaningful time constraints in our domain. Additionally, we only employ a very limited attention filter mechanisms in order to focus on more general management principles. In accordance with the GGM framework, surfaced goals are actively managed. The goal management processes in Dora currently rank all surfaced goals by the ratio of information gain to costs, before selecting the highest-ranked goal as the next one to pursue.

In Dora, both information gain and costs are designed to reflect domain specific conditions, though their specific implementation is beyond the scope of this paper. In brief, the information gain for achieving the goal of visiting an unexplored place is derived from a measure of the amount of space predicted to be covered by this place. The information gain of categorizing a room is similarly designed, assuming that a categorising bigger rooms yields more information. The cost of exploring a place is determined by the distance to that place. The cost of categorising a room is the cost of getting there plus the predicted cost of performing AVS.

Room categorisation via AVS provides an example of how domain knowledge can be modelled in the GGM framework. Because AVS requires a map of non-free space in order to calculate a search plan, a room must be *adequately* explored before AVS can be performed in it. However, Dora does not have to *completely* explore the room (by visiting every place it contains) before this can happen. While a heuristic, dynamic definition of "adequate" can easily be implemented in a domain specific goal generator, it is more difficult to define this vague precondition in a planning domain. If we wished to remove the GGM framework from Dora (as we shall do for evaluation purposes in the subsequent section), the only natural way to encode the relationship between exploration and AVS is to add a precondition to Dora's AVS action. This precondition would have to state that no unexplored places can exist in a room if it is to be categorised, thus losing the notion of adequate exploration.

Systemic Evaluation

To investigate the influence our GGM framework has on an integrated system, we have gathered data from multiple runs of two configurations of the Dora system. The first configuration explicitly encodes its two drives (to explore space and categorise rooms) as the single planning goal:

```
(and (forall (?p - place) (= (explored ?p) true))
     (forall (?r - room)  (kval 'robot' (areaclass ?r))))
```

This goal, literally interpreted as "have explored all places and know a category for every room", is passed to the continual planner directly, rather than via the GGM framework. In this configuration, termed *conjunct goal set (CGS)*, all system behaviour is determined by the continual planner in the PECAS architecture in response to this unchanging goal. The planner continuously monitors WM content, triggering replanning if any relevant state changes occur (e.g. unexplored places appearing on the spatial WM). The second configuration, termed *managed goal set (MGS)*, employs our implementation of the GGM framework. In this configuration the planner is fed the individual goals selected by the management mechanisms described previously.

Although Dora is usually run in the real world, we performed the following experiment using a simulated robot for the sake of repeatability. None of Dora's software was changed for the experiment, so the evaluated system is the same as the one run on the robot. We used a test arena with five rooms and a corridor in the robotic simulator *Stage*[3]. The structure of the environment, which covers approximately $93m^2$, is captured in the map in Figure 2(b). Dora's sensors are simulated by Stage, allowing us to employ the same system architecture in the simulated setting as we use in the real world, with the exception of the object recogniser. This is replaced with Stage's "blobfinder". Whenever the camera is pointed towards a near-by simulated object in Stage an object is recognised. Two simulated objects are placed in each room in the arena. The objects describe one of three categories according to the OpenMind Indoor Common Sense Database (room, office, and kitchen), allowing the conceptual mapping SA to categorise these rooms. Everything is run on a standard 2.4Ghz Core2Duo notebook with 4GB RAM.

A single run is defined as follow: First, the system starts from scratch every run. Next, the robot is given a short, predefined tour (superimposed on Figure 2(b) as a dotted arrow) during which the robot builds up its initial knowledge but it does not take any action itself. Finally, the system is switched to autonomous mode and starts acting in response to its goals.

In total we ran the system 15 times: 8 in the MGS configuration and 7 in CGS. This set is referred to as S_{all}. A run for the CGS configuration was defined as complete when the conjunctive goal was achieved (i.e. no places left unexplored and no rooms uncategorised). The MGS configuration was said to be complete when no more surfaced goals remained.

As we are evaluating Dora under full-system conditions (rather testing using isolated components) we have to accept the performance limitations of the real (research) components which can occasionally fail to perform correctly. The effect of this is visible in a large variation in the time taken by Dora to complete a run (between 10 and 23 minutes). When looking at the number of rooms actually found and categorized in each run, we find a number of runs in which the robot completed the mission to categorize all five rooms. We select these runs as a subset termed S_{R5} composed of four runs of each configuration. We argue that runs in this set are comparable because the qualitative extent of the information acquired is equivalent.

[3]Available from `http://playerstage.sf.net`.

	CGS	MGS
avg. time per planning call	0.621 s	0.292 s
avg. time spent on planning	48.843 s	8.858 s

Table 1: Planning time measures (all in seconds).

Results

To measure the influence of the GGM framework we can examine the time Dora spends planning. These measures are obtained from S_{all} and summarised in Table 1. The differences between the averaged timings taken for the two configurations are statistically significant with $p < 0.0001$ in Mann-Whitney testing for all measures shown in the table.

As the first row of the table indicates, there is a significant difference between the average time taken by a single call to the planner. A call occurs either when the goal management activates a new goal or when replanning is triggered by a state change. Planning calls in CGS take more than twice the time compared tp MGS. This is due to the higher complexity of the planning problems in the CGS configuration (it is planning for all possible goals rather than a single goal). If we look at the average time spent on planning in total in a run (second row in Table 1) the difference is more prominent. This is due to the fact that in the CGS configuration the planner is triggered more often: 79.0 times on average, compared to 31.1 times for the MGS configuration. This is because the longer plan lengths required in CGS are more likely to be affected by state changes and thus require more frequent replanning.

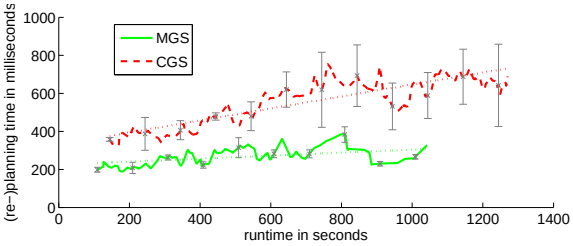
Figure 3(a) shows how the complexity of planning problems evolves as the system is running. This presents the length of single planner calls against the runtime of the system. This plot has been created using S_{R5} for comparability. We average the planning time at discrete time steps during the system's runtime, across all selected runs of each configuration. The error bars indicate the standard error in averaging. From this figure it is apparent that, in agreement with the data in Table 1, less planning effort is required in MGS compared to CGS. It can also be seen that the progression over runtime is different in the two cases. While the trend, indicated by a linear fitting shown as a dotted line in Figure 3(a), is a shallowly included line for MGS, a steeper increase in average planning time can be seen for CGS. This steeper increase can be associated with the increasing size of the planning problems the CGS configuration faces as Dora's knowledge increases: planning for all possible goals over a larger and larger state becomes increasingly difficult.

The results so far underpin our hypothesis that planning effort can significantly be reduced by employing GGM mechanisms that allow a system to select the goals that should be planned for now, rather than forcing it to plan for all goals that could be possible. The reduction in effort is not a surprise, as one can argue that we are only looking at parts of the problem at a time, and thus that we are trading planning speed against the opportunity to generate plans which satisfy all the system's desires in the most efficient way possible. Contrary to this view, our experiments show that the MGS configuration satisfies its drives in a shorter time, exploring the environment more efficiently on average than the CGS configuration. So, why is this the case in our system, even though the planner considers costs by means of minimizing the number of actions to achieve the conjunctive goal? Figure 3(b) can help us understand this effect and illustrates an advantage of the MGS approach.

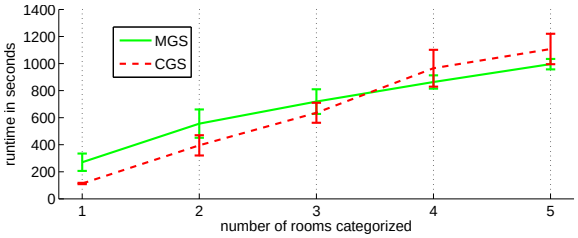
The plot depicts the time (y-axis) by which a given number of rooms have been categorised (x-axis), averaged over all runs of set S_{R5}. It shows that the MGS configuration takes longest to categorise the first two rooms before speeding up to be the fastest to complete four and five rooms. This observation can be explained by the use of domain-specific *information gain* for goal activation in the MGS configuration, information which is not available to the planner in either configuration. In our experiments, the MGS configuration did not choose to immediately categorise the room the tour ended in (see the arrow in Figure 2(b)). Instead it chose to visit unexplored places that were attributed with a high information gain, even though the precondition for categorising the first room was fulfilled. In contrast, the CGS configuration always categorises the first room immediately because it deems this to be the cheapest approach to achieve the overall goal. The domain knowledge encoded in the information gain used to select the goals leads to more efficient behaviour because it drives the system to explore large unexplored areas first due to their high potential information gain. This leads to the MGS configuration building up a map capable of supporting AVS on all rooms faster than the CGS configuration is able to, even if it means ignoring categorisation early on in a system run. For this reason we would expect the trend in Figure 3(b) to become more pronounced given longer runs in larger areas.

Discussion & Related Work

Although the problem of generating and managing goals for an integrated system has not been studied widely by the AI community, there is a body of work to which our work relates. The work of Coddington (2007) supports the link between motivation and planning complexity. In a limited setting she compared two approaches to generating goals for an agent: reactive generation (comparable to our MGS configuration), and encoding all the system's goals as resources in its planning domain (comparable to our CGS configuration). This work demonstrated that by only using reactive goal generators the system could not guarantee to satisfy all of its desires, as the effects of actions in current or future plans were not reasoned about by the generators. Coddington views using a planning process to decide which goals should be pursued as a solution to this. As a planning approach would consider all interactions between possible goals and current actions it prevents potentially deleterious situations occurring. However, Coddington demonstrated that the computational cost of encoding all possible goals in a planning problem prevented the system tackling problems beyond a certain size. Our findings, particularly those related to planning time, agree with this. Dora suffers from the problems Coddington identified with reactive goal generation, as we currently activate just one goal at a time for planning (our domain does not require more). In future work we will investigate methods for informed activation of multiple goals using oversubscription planning. This will provide



(a) Averaged planning time during a system run.



(b) Running time when rooms have been categorized.

Figure 3: Dora timing information.

a variable scale of complexity and deliberation between the two extremes of reactive and planned goal generation.

The problems and benefits of autonomous goal generation in an integrated system setting are demonstrated by the work of Schermerhorn et al. (Schermerhorn et al. 2009) present a system that can use the preconditions and effects of planning actions to generate new goals for a system at run-time. This approach has clear benefits in unpredictable worlds and would fit cleanly within a generator in our framework. (Schermerhorn and Scheutz 2009) highlights the problems of treating goal activation as a rational decision making problem. The primary difficulty is getting reliable, comparable models of actions and the environment. Dora faces this problem when attempting to compare measures of information gain from different types of knowledge gaps.

Other frameworks for goal generation or management have been proposed previously. These approaches typically fail to make the distinction between generation, surfacing and activation, instead assuming that generation implies activation (ignoring the requirement to deliberate about possible goals). We can consider such approaches (in terms of goal generation and management) as implementations of a subset of our framework. Examples can be found in belief-desire-intention systems (e.g. (Georgeff and Ingrand 1989)), behaviour-based systems (e.g. (Bryson 2003)), and reactive planners with goal management extensions (which represents perhaps the largest body of work on this subject) (e.g. (Gordon and Logan 2005; Michaud et al. 2007)).

Conclusion

In this paper we presented a framework for goal generation and management and demonstrated its efficacy in a first implementation in Dora the Explorer, an intelligent mobile robot. Our evaluation shows that a system using the framework outperforms (in terms of both planning effort and task completion time) a system in which the drives of the robot are encoded as a single planning goal. Though our current implementation is limited, we are convinced that a goal generation and management framework will allow us to work towards a principled coupling between reactive and deliberative behaviours in open-ended integrated systems in general and robotics in particular.

References

Beaudoin, L. P., and Sloman, A. 1993. A study of motive processing and attention. In Sloman, A.; Hogg, D.; Humphreys, G.; Ramsay, A.; and Partridge, D., eds., *Prospects for Artificial Intelligence: Proc. of AISB-93*. Amsterdam: IOS Press. 229–238.

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

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

Coddington, A. M. 2007. Integrating motivations with planning. In *Proc. of the 6th International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS '07)*, 850–852.

Georgeff, M. P., and Ingrand, F. F. 1989. Decision-making in an embedded reasoning system. In *Proc. of the 11th International Joint Conference on Artificial Intelligence (IJCAI '89)*, 972–978.

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

Hawes, N., and Wyatt, J. 2010. Engineering intelligent information-processing systems with CAST. *Adv. Eng. Inform.* 24(1):27–39.

Hawes, N.; Zender, H.; Sjöö, K.; Brenner, M.; Kruijff, G.-J. M.; and Jensfelt, P. 2009. Planning and acting with an integrated sense of space. In *Proc. of the 1st International Workshop on Hybrid Control of Autonomous Systems (HYCAS)*, 25–32.

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

Özuysal, M.; Fua, P.; and Lepetit, V. 2007. Fast keypoint recognition in ten lines of code. In *Proc. of Conference on Computer Vision and Pattern Recognition (CVPR '07)*, 1–8.

Pronobis, A.; Sjöö, K.; Aydemir, A.; Bishop, A. N.; and Jensfelt, P. 2009. A framework for robust cognitive spatial mapping. In *Proc. of 10th International Conference on Advanced Robotics (ICAR 2009)*.

Schermerhorn, P. W., and Scheutz, M. 2009. The utility of affect in the selection of actions and goals under real-world constraints. In *Proceedings of the 2009 International Conference on Artificial Intelligence (ICAI '09)*, 948–853.

Schermerhorn, P. W.; Benton, J.; Scheutz, M.; Talamadupula, K.; and Kambhampati, S. 2009. Finding and exploiting goal opportunities in real-time during plan execution. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '09)*, 3912–3917.