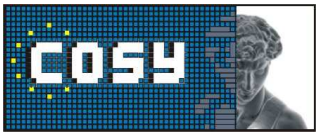

FP6-004250

CoSy

Cognitive Systems for Cognitive Assistants

Integrated Project

Information Society Technologies

DR.1.?

PlayMate Architecture Design

Due date of deliverable: June, 2006
Actual submission date: June, 2006

Start date of project: September 1st, 2004 **Duration:** 48 months

Organisation name of lead contractor for this deliverable:
University of Birmingham

Revision: Draft
Dissemination Level: PU

Contents

I General Design **4**

1 Introduction **4**

2 Key Features **4**

3 Design Principles **5**
3.1 Concurrent Modular Processing 5
3.2 Structured Management of Knowledge 5
3.3 Dynamic Contextual Processing 6

II Design of the General Components **7**

4 Sub-Architecture Design **8**

5 Goals and Control **10**
5.1 The Motive Manager 10
5.2 The Goal Manager 10

6 The General Memory **11**

7 The Instance Memory **11**
7.1 Sub-Architecture Search 11
7.2 Equivalence Binding 12

III Design of the Components for the Month 24 PlayMate **12**

8 What's Missing? **12**

9 Visual Sub-Architecture **14**
9.1 Behaviour Goals 15
9.2 Information-Processing Goals 15
9.3 Goal-Driven Component: Visual Search 15
9.4 Goal-Driven Component: Generate Depth-Map 15
9.5 Goal-Driven Component: Extract Objects from Depth-Map 16
9.6 Goal-Driven Component: Get Visual Attributes 16
9.7 Data-Driven Component: Hand and Arm Location 16
9.8 Data-Driven Component: Change and Motion Detector 16
9.9 Data-Driven Component: Context-Sensitive Saliency 16

10 Planning Sub-Architecture **16**
10.1 Behaviour Goals 17
10.2 Information-Processing Goals 17
10.3 Goal-Driven Component: Generate Problem Representation 17

10.4 Goal-Driven Component: Produce Plan for Problem 17

11 Execution Sub-Architecture 17
11.1 Behaviour Goals 18
11.2 Information-Processing Goals 18
11.3 Goal-Driven Component: Execute Plan 18
11.4 Goal-Driven Component: Execute Plan Step 18
11.5 Goal-Driven Component: Execute Step Actions 19
11.6 Goal-Driven Component: Manipulation Library 19

12 Linguistic Sub-Architecture 19
12.1 Behaviour Goals 20
12.2 Information-Processing Goals 20
12.3 Goal-Driven Component: Parse Utterance 20
12.4 Goal-Driven Component: Resolve References 20
12.5 Goal-Driven Component: Interpret Dialogue Utterance 20
12.6 Goal-Driven Component: Plan Dialogue 20
12.7 Goal-Driven Component: Generate Multi-Modal Content 21
12.8 Goal-Driven Component: Realise Utterance 21
12.9 Goal-Driven Component: Synthesise Speech 21
12.10 Data-Driven Component: Speech Recognition 21

A Worked Example: Linguistically Triggered Manipulation 21

Part I

General Design

1 Introduction

This document presents the detailed design for the proposed architecture for the month 24 PlayMate integrated system. The requirements for this architecture have been derived in other documents and are based on a set of tasks related to manipulating simple objects on a table top.

Before we discuss the architecture itself we present a brief list of key features of the architecture, followed by a more detailed discussion of the key design principles that motivate it. If you do not wish to read the whole document, then we suggest you read the list of key features in Section 2, and then proceed to II.

2 Key Features

The follow list provides a quick summary of the key features of the architecture. The motivations behind them are presented in the subsequent sections.

1. The architecture contains *sub-architectures* for separate competences. These sub-architectures are initially loosely coupled. In the initial system there are sub-architectures for vision, action, planning and language.
2. Information sharing between sub-architectures occurs in working memories that exist separately for each sub-architecture.
3. Each working memory is only writable by processes within the sub-architecture, and by a single global process (the *goal manager*).
4. Interaction between the sub-architectures is controlled by the goals generated within the architecture at run time, allowing it to support different approaches to processing (e.g. incremental processing, forward chaining, backward chaining etc.). The architecture defines three classes of goal: *task goals* (a goal that requires coordination across sub-architecture), *behaviour goals* (a goal that requires multiple processing steps within a single sub-architecture), and *information-processing goals* (a goal that requires a single processing step within a single sub-architecture).
5. The representations used within a sub-architecture are defined by an ontology for that sub-architecture.
6. The links between representations in different sub-architectures are defined by a general ontology.
7. The architecture defines three class of information: *local instance* (sub-architecture restricted information), equivalence class (sub-architecture information that is stable across time, and possibly bound to the general ontology) and *general instance* (information bound to the general ontology that cuts across sub-architectures).

3 Design Principles

The following design principles represent some of the aspects of the architecture which we view as central to the problem of designing an integrated cognitive system.

3.1 Concurrent Modular Processing

Our architecture design is composed of a fixed number of processing components. As many of these components as necessary can be active concurrently. We specify this because we believe such concurrency is necessary to handle the kinds of multiple-competence tasks that our system must perform (e.g. using visual information to guide grasping whilst being aware that a human might ask it a question).

In our design processing components are collected into *sub-architectures* for various competences (language, vision, manipulation, action planning) connected by a number of competence-independent components. Sub-architectures are required to give the architecture coarse internal structure. This structure is useful for the managing information exchange across processes (cf. Section 3.2), and controlling processing (cf. Section 3.3)

To reduce the dependence of components in one sub-architecture on the presence of other components within the same or other sub-architectures, our design forces sub-architectures to interact via local working memories. Our current design allows input to a sub-architecture's working memory by components within the sub-architecture and global processes. In principle any process can read from the working memory of any sub-architecture, although we will probably not make use of this in the month 24 PlayMate.

3.2 Structured Management of Knowledge

A key problem when designing (and implementing) a cognitive system is giving it the ability to manage information it processes in a well-defined manner. Many designs focus on structuring processing components but ignore the information that flows between them. This make the whole system more about different forms of processing, rather than a gradually evolving information processing context. To overcome this, we propose a structure for the management of information across the architecture. A great deal of this structure is inspired by the work on information fusion for the ComSys [Kruijff et al., 2006].

The information structure is defined by two levels of ontologies: *sub-architecture ontologies* and *general ontologies*[1]. Each sub-architecture has its own set of sub-architecture ontologies which define the information it can process. These ontologies will contain knowledge specific to the competences that the sub-architecture supports. It is assumed that a lot of this knowledge will only be understood by the processes within the sub-architecture (e.g. if they are intermediate results of a process), or that this knowledge may have an expected useful life-time that may make sharing it with other sub-architectures pointless (e.g. precise metrical information about the world which will change unpredictably). The general ontologies are not linked into any particular sub-architecture, but are competence independent and represent knowledge that is relevant across both the collection of

[1] We use the term "ontology" to mean a (structured) definition of the kinds of information that the cognitive system can process from either internal or external source. This is not the same as an ontology that could be used by the *designer* of such systems

sub-architectures and the general architecture mechanisms (such as task goals, coarse conceptions of space and time, plan structures etc.). The general ontologies are used for general-purpose planning and reasoning, motive generation and other tasks that must cut across sub-architectures.

To embed the knowledge of the sub-architectures within the knowledge framework of the whole architecture, concepts within the sub-architecture ontologies that are closely related to concepts within the general ontologies must be linked together (e.g. action recognition models could be linked to action verbs and to planning actions). This allows general mechanisms and sub-architectures to interact with other sub-architectures via the mediating presence of the general ontology.

The ontologies only define the categories of knowledge that the architecture can use, and must be connected with actual information within the architecture and sub-architectures. Our design recognises three levels of information: **local instance**, **equivalence class**, and **general instance**. A local instance is an item of information within a sub-architecture that is an instantiation of a concept from that sub-architecture's ontology which has no links to concepts in the general ontology. Local instances that all describe a single piece of information (i.e. models of an object extracted from subsequent frames, or a linguistic reference to a particular action in two different utterances) can be bound together into an equivalence class along with the related concept from the sub-architecture ontology. It is intended that the sub-architecture ontological concept that is linked to an equivalence class should have a link to an entity in the general ontology as well, providing access to this information across the architecture (as this kind of temporally extended information is more useful in more general processes). At the level of the whole architecture, we consider any equivalence class that is linked to an concept in the general ontology to be an instantiation of that general concept and hence denote it as a general instance. Because of the nature of presence of multiple sub-architectures, it is intended that general instances will bind together related equivalence classes from different sub-architectures (like belief structures in the ComSys).

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

It is important that the ontologies contain only information that is required by the PlayMate in order for it to complete the tasks in the scenario. Adding additional information that is unrelated to the competences necessary for the scenario may result in demonstrations of the system being misleading to some.

3.3 Dynamic Contextual Processing

Our previous requirement of multiple concurrent processing components results in the need for mechanisms to control which components can influence processing and when. We propose two ways in which a component in our architecture can be controlled. First, sub-architectures, or components with sub-architectures can be turned on or off by other components. This may be necessary if they run constantly and always require resources that could be better used by other components. Second,

components must always announce their intent to perform some processing by posting a goal stating what they want to do, and why. An additional control mechanism can then monitor the current state of the architecture and determine whether the goal should be adopted or not. A side-effect of this approach to control is that there can be no fixed path of information flow through the architecture as components may not always be operational, or permitted to process when they can.

To support this dynamic approach to information processing, our architecture presents three types of goals:

1. **Task Goals**: The first set of goals we will consider are task goals. These are the goals that control the global information processing behaviour of the architecture. Once a task goal has been adopted it can be considered as the ultimate justification of the system's subsequent behaviour. In the month 24 scenarios, the PlayMate's task goals will be to explore the world (i.e. gather as much information as possible) and to respond to an utterance (either just linguistically or through action or using a combination of both). Task goals will typically involve coordination across multiple sub-architectures.

2. **Behaviour Goals**[2]: Behaviour goals represent a coordinated chain of processing within a single sub-architecture. They will be invoked to satisfy the task goals (so they could be considered as task sub-goals). In the month 24 scenarios the behaviour goals will include producing an utterance and producing a particular physical configuration in the world.

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

Within the architecture, it is the job of the *goal manager* to decide whether the behaviour and information-processing goals should be adopted or not. This will depend on the overall task goals, and what other demands are currently being made on the architecture. This will require this component to have a knowledge of what the architecture is currently doing, which things can happen in parallel, and, more importantly, how the results of particular processes contribute towards the satisfaction of task and behaviour goals. This final type of knowledge could either be statically defined, or derived by a deliberative system such as a planner.

Part II

Design of the General Components

This part discusses the design of the parts of the architecture that are specific to the month 24 PlayMate, but will instead be present in any architecture based on this design. Ideally these components will be designed and implemented in a way that is completely independent of the task, so that they can be reused freely to produce other architectures based on the same general design. We expect the design and evaluation of these mechanisms to represent significant contributions towards WP 1.

[2]These could possibly be better named as sub-architecture goals.

4 Sub-Architecture Design

The core of the architecture design is the manner in which local processes in sub-architectures interact with the global information stores and control processes via local working memories. This section will describe the general template for this interaction. We start with this because it also demonstrates the role of the general architecture components such as the goal manager.

As mentioned in Section 3.3 to make the behaviour of sub-architectures as configurable as possible (to allow attentional mechanisms to have an effect) we propose that every time a processing step can be undertaken the sub-architecture proposes this step as a possible goal that must be adopted by a goal management mechanism. Depending on the current control profile of the whole architecture these processing goals may automatically be adopted, or they may get queued until the necessary resources are available (or some other condition is met). This level of indirection will also allow different components to process the same information in different ways with the goal manager selecting which one should be active (or if they all should). Such an approach may appear unwieldy in the short-term, but in the long term it will allows us to scale our architectures to bigger problems, and investigate various methods of control.

We propose that each sub-architecture should be designed along the lines of Figure 1 (the rest of the global components are not shown). Each sub-architecture should be composed of the following components:

- **Sub-Architecture Working Memory**: A sub-architecture specific working memory that holds the results of the processes contained within the sub-architecture. It could also maintain a local history. Entries should conform to the information and ontology structure of the architecture.
- **Data-Driven Processing Components**: A set of components that monitor data external to the sub-architecture for simple, easily determined cues for further processing within the sub-architecture. In the current design these will typically be linked to a robot's sensors. These components can add information to the working memory without proposing a goal. Examples of such components may be a motion detector in a visual architecture, speech-detector in a linguistic architecture, or a pattern recognition mechanism in an alarm architecture.
- **Goal-Driven Processing Components**: The main set of processing components for the sub-architecture. Each processing component must monitor the working memory for information that it can process. When it recognises such information it must generate a goal that indicates what it intends to do. This goal is placed into working memory, and the component must then wait for it to be adopted by the goal manager. The goal manager signifies that the goal has been accepted by adding this fact to the working memory (or it could toggle a flag in the goal itself, depending on implementation).
- **Sub-Architecture Ontologies**: An ontology or set of ontologies that define the knowledge used by the sub-architecture. Parts of these must be linked to parts of the general ontology to allow information to be shared across the architecture (see Section 3.2).

A typical processing cycle will look something like the list below. In this example, all sub-architecture components are in the same sub-architecture, the format and other presentation details are unimportant to the content, and the names of goals and types of data will be defined by the ontologies.

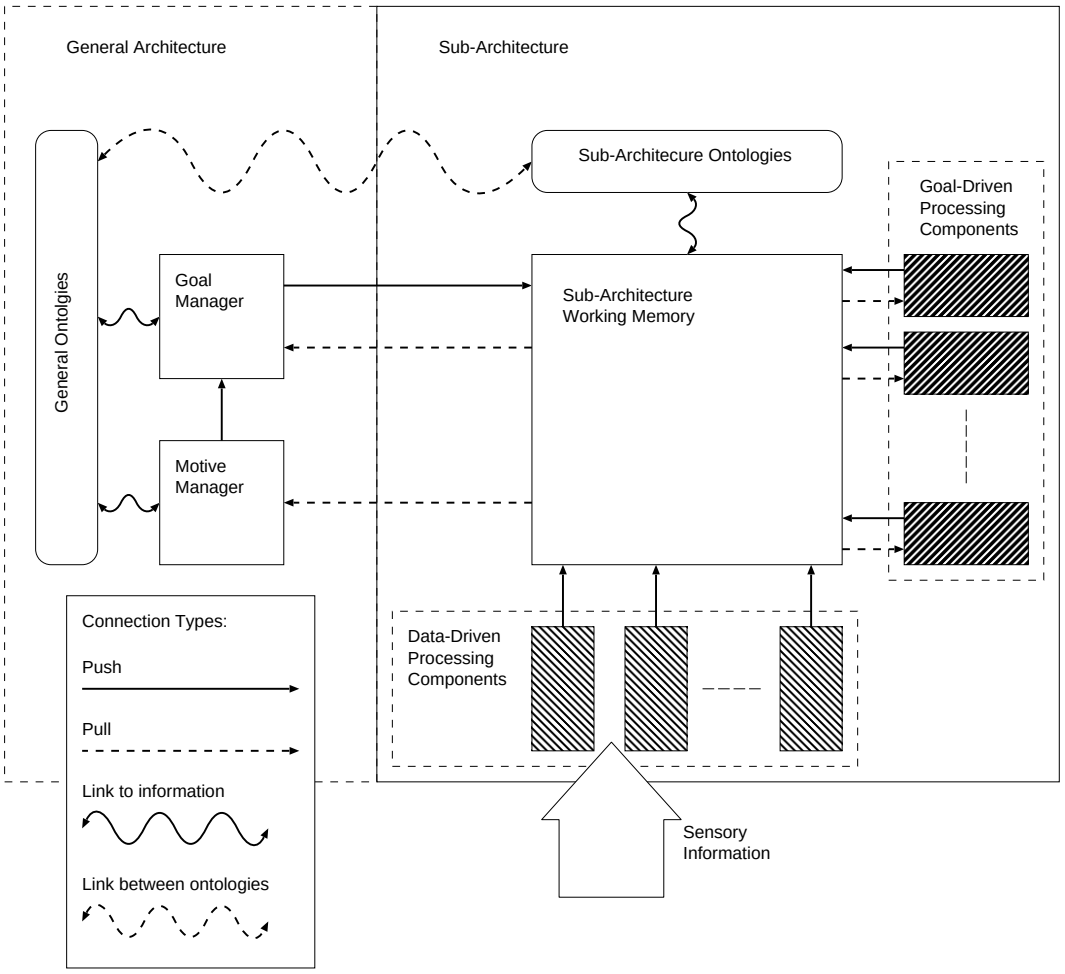

Figure 1: Sub-Architecture Design

1. A data-driven component extracts some raw data from a sensor, and adds it to the sub-architecture working memory: `[raw_data data]`. Where `raw_data` would be the name of an entry from the sub-architecture ontology, and `data` is the actual data.

2. The motive manager spots this and pushes a new behaviour goal into the goal manager `[behaviour_goal behaviour_goal_1 sub_arch_beh_goal [raw_data data]]`. Where `behaviour_goal_1` is the id of the goal and `sub_arch_beh_goal` is the name of the behaviour goal proposed by the motive manager.

3. A goal-driven component in the sub-arch also spots the `raw_data` and adds a goal into the working memory to process it: `[information_goal information_goal_1 process_raw_data [raw_data data]]`. Where `information_goal_1` is the id of the goal and `process_raw_data` is the name of the information processing goal.

4. Through some decision process, the goal manager adopts the behaviour goal for the sub-architecture.

5. Then, because `process_raw_data` is a sub-goal of `sub_arch_beh_1` the goal manager adopts the information processing goal. To signify this it adds the following to the sub-architecture working memory: `[adopt information_goal_1]`.

6. The processing component that proposed the information-processing goal sees this and performs the processing. The result of this processing is that `[processed_data pdata]` is added to the working memory.

7. This may be spotted by another processing component which will go on to propose another information-processing goal, or by the motive manager which may propose another behaviour goal for the goal manager.

In addition to the above design, we may also need to make allowances for specialised connections between sub-architectures that either circumvent some of these mechanisms, or are unaffected by parts of them. For example, how do we get images into visual servoing (perhaps via a link through the instance memory).

In terms of the CogAff schema, we assume that data-driven components will generally be reactive, and the goal-driven components can be either reactive or deliberative.

5 Goals and Control

The sub-architecture design diagram (Figure 1) and discussion (Section 4) referred to two components in the architecture which direct the processing based on various types of goals. These components are described in the following two sections.

5.1 The Motive Manager

The job of the motive manager is to monitor the working memories of the various sub-architectures (in a pull-based manner) for information that could cause the suggestion of a new goal. To enable this to happen, each sub-architecture must register its possible behaviour goals and their triggers with the motive manager.

5.2 The Goal Manager

The goal manager is the component that approves the information processing goals proposed by components within sub-architectures. The goals are monitored by pulling information from the working memories of sub-architectures.

To perform effectively the goal manager must keep track of current task and behaviour goals within the architecture, and know how information processing goals relate to these. To support this, each sub-architecture must register its list of behaviour and information-processing goals with the goal manager. The goal manager should also have some way of judging the current availability of resources and the relative importance of different goals so that it can make informed choices between information processing goals.

Another job of the goal manager is to manage dependencies between sub-architectures when one has requested processing for another. For this the goal manager must have knowledge about the states generated by components on the completion of particular tasks, and know how to communicate this information to other components. This can be encoded in a general ontology for internal processes.

The goal manager is also be able to trigger behaviour in sub-architectures, as well as just coordinate it. For example, in the worked example in Appendix A, after adopting the task goal of responding to an utterance, the goal manager has to trigger the linguistic and planning response to this by adding the appropriate facts to the working memories of the respective sub-architectures. To do this it must know which behaviour triggering facts are related to the behaviour goals for each sub-architecture.

6 The General Memory

The general memory in the architecture stores knowledge about concepts, types, processes, competences and anything else that can be used to define to a particular instance in the world or the architecture. It can be considered as the PlayMate's *semantic* memory. In particular the general memory contains the PlayMate's general ontologies (see Section 3.2). For the target scenarios these ontologies will be fixed, but later systems may add additional modules to manage ontological changes.

7 The Instance Memory

The instance memory sub-architecture manages the PlayMate's memory for general instances of entities at exist in its world or in its architecture.

The sub-architecture will provide the following functionality:

1. Given a reference to an equivalence class bound to a concept in the general ontology, it should be able to say whether or not it is in the memory, e.g. by returning the possible general instance matches.
2. Given a concept from the general ontology, return all the memorised general instances of that concept.
3. Bind together equivalence classes from different sub-architectures that relate to the same general instance.

The knowledge necessary to bind equivalence classes together is quite task-specific. The minimum will be that they are bound to concepts with common reflections in the global ontology. They should then also be temporally and spatially aligned if possible via histories or gestures or assumptions (if they are made clear).

7.1 Sub-Architecture Search

The binding of a general instance in the instance memory to equivalence classes in various sub-architectures can happen as the result of normal processing, but sometimes a component may require

that a general instance is bound to an equivalence class in a particular sub-architecture. If such a case arises then the sub-architecture search component will propose a goal to carry out the process of finding a suitable EC. It will do this by querying the relevant sub-architecture using concepts from the general ontology that describe the instance. The component only expects a return value to say when the search is over, the actual binding is handled separately.

As demonstrated in the worked example in Appendix A, sub-architecture search can be initiated by any sub-architecture by adding a particular fact into its working memory. It then must wait for an indication from the goal manager that the search has terminated.

7.2 Equivalence Binding

This component monitors the creation of new equivalence classes in the sub-architectures, and the corresponding instances in the instance memory. It attempts to conservatively bind equivalence classes into as few global instances as possible. It will do this by comparing their bindings to the general ontology, and their spatio-temporal information.

Part III

Design of the Components for the Month 24 PlayMate

This part describes the sub-architectures that are necessary to provide particular competences for the month 24 scenario tasks (see earlier scenario document for more information). Each section contains a list of the goals used by the sub-architecture, including the goal-subgoal relationships between the behaviour goals for the sub-architecture and the information-processing goals its goal-driven components will propose. Each section will also include a description of the sub-architecture goal-driven and data-driven processing components.

A schematic overview of the whole architecture design, including the general components, can be seen in Figure 2. This diagram excludes the internal sub-architecture connections, but these can be seen in subsequent figures.

8 What's Missing?

At the moment we currently have not added design elements for:

1. Visual attribute learning provided by UOL. Our intention is to have an additional behaviour goal for the visual sub-architecture to locate an indicated object (via the mechanisms described for other goals), then trigger learning using the given input.
2. Action recognition provided by BHAM, and related support for reasoning about process. The action recognition components will probably go into the visual sub-architecture too and run continuously to provide ROIs based on expected change to the world (and more).

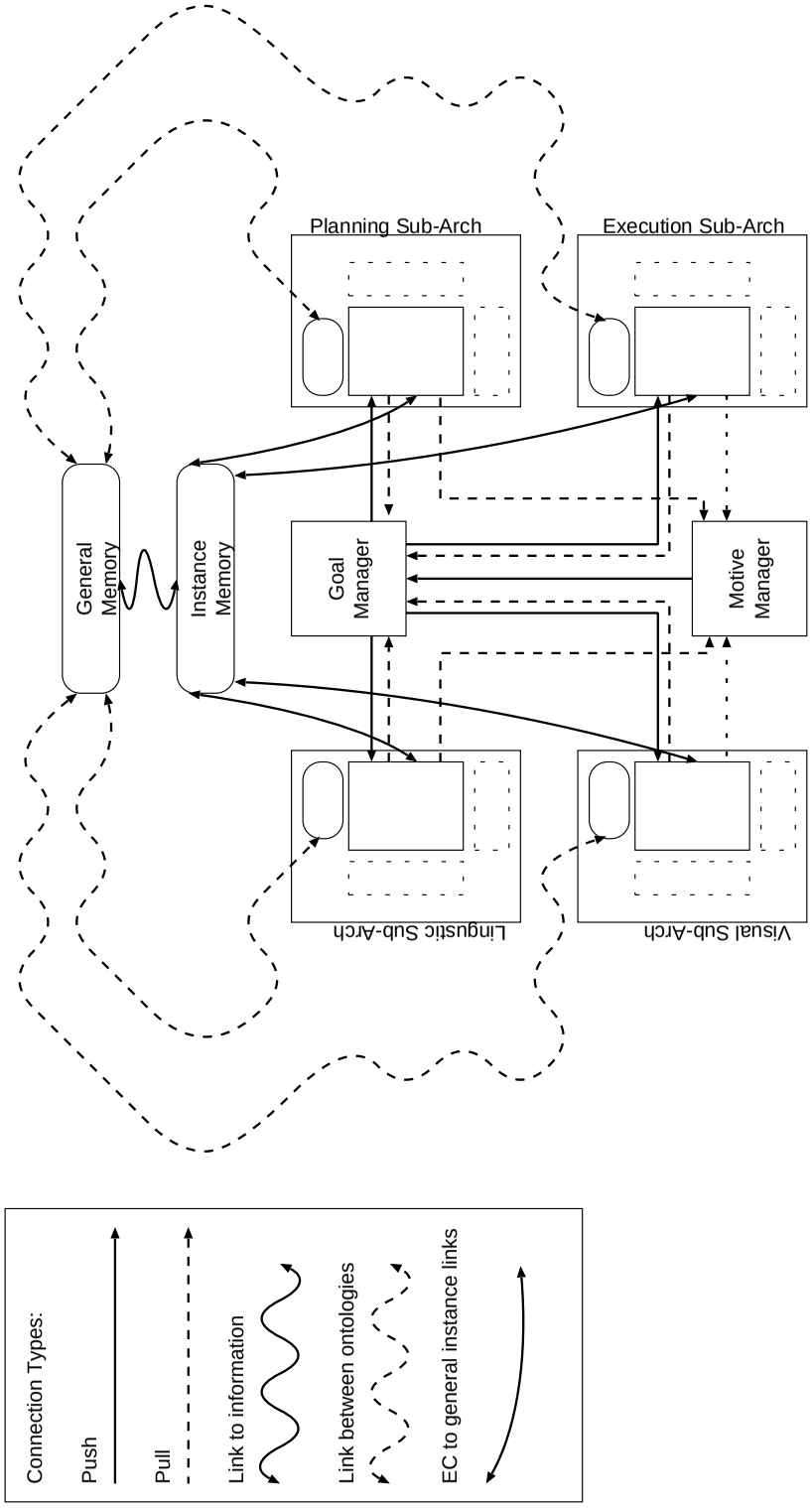

Figure 2: Architecture Overview (sub-architecture details not shown)

3. Action model learning for the earlier scenarios. This will be handled by an additional task goal for the system that will be triggered by interest in something particular, idleness etc.

Also all the ontologies need to be specified in collaboration with partners.

9 Visual Sub-Architecture

In the architecture for the month 24 PlayMate, the visual sub-architecture will be chiefly concerned with extracting 3D information about the scene to facilitate manipulative actions. It will also extract information about the visual attributes of objects (such as shape, colour and size) which will be used, along with spatial information, to disambiguate objects when they are involved in an utterance about a manipulation act. The design of the sub-architecture is pictured in Figure 3.

The visual sub-architecture works in two different modes, based around its two behaviour goals. The Process ROI mode requires “interesting” ROIs to be suggested by the change, motion and saliency components. These ROIs are then processed to obtain any information they contain. The Visual Search mode takes a list of visual attributes and searches the current scene until it has found objects that match the attributes.

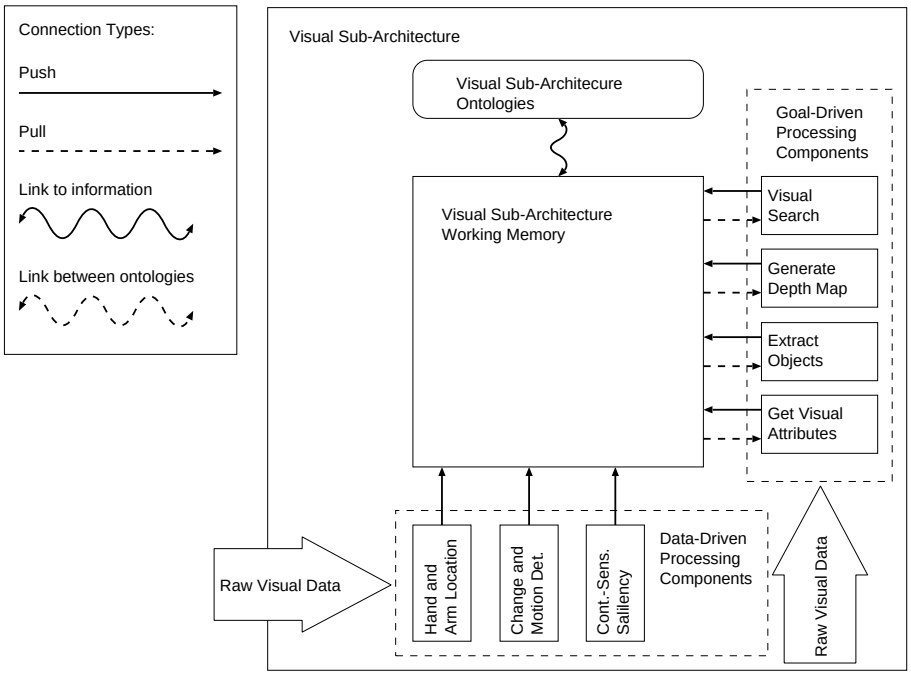

Figure 3: Visual Sub-Architecture Diagram

9.1 Behaviour Goals

1. Visual Search. A visual search behaviour goal should describe the visual properties being searched for. This may include object shape, size, surface type, colour etc. The expected behaviour is that either nothing is found, or all entities with matching properties are returned to the source or the goal and/or the instance memory.
2. Process ROI. Given a region of interest, process it for relevant visual information. This behaviour goal will be generated by low-level (i.e. pixel-based) change, motion or saliency components within the visual sub-architecture.

9.2 Information-Processing Goals

1. Visual Search.
   (a) Propose ROIs
   (b) Generate Depth Map
   (c) Extract Objects from Depth-Map
   (d) Get Visual Attributes
2. Process ROI.
   (a) Generate Depth Map
   (b) Extract Objects from Depth-Map
   (c) Get Visual Attributes

9.3 Goal-Driven Component: Visual Search

Given the current information-processing context and a search target this component proposes a list of regions of interest to be searched. This may be based on change, motion or saliency information, and may also draw on scene interpretation, gaze recognition or other more advanced properties. Each ROI it proposes will get further processed by other components until either the search target has been found (if a single target was requested) or all of the ROIs have been processed. At this point this component indicates that the search is over so that other components waiting for results (e.g. the goal manager) can continue.

9.4 Goal-Driven Component: Generate Depth-Map

Produce a depth map of the current scene. It is expected that the processes supporting this will run more-or-less constantly, and the goal to generate the map will just return the current map. This will then be added to the visual working memory to serve as input to other processes.

How this will be implemented will depend on the site and hardware. UOL are using a Bumblebee and related software, DFKI are using 2 firewire cameras and SVS, and BHAM are using 2 cameras attached to capture cards and have SVS available. The interface to this component will obviously have to be independent of these issues.

9.5 Goal-Driven Component: Extract Objects from Depth-Map

The VSA needs some method of extracting representations of objects in the scene. Depending on implementational details this may be done by fitting object models, object part models, or surface models to a depth map whilst using other visual cues for guidance. If whole object models are not used, then an additional process will be needed to bind other information into object representations for object-centred reasoning.

UOL have recently agreed to supply this component, so the exact details will depend on them. One of the hardest tasks will be to make the system robust to the noise inherent in stereo depth data.

9.6 Goal-Driven Component: Get Visual Attributes

Given a particular object, part of object, or surface, return a list of the visual properties of this entity. The attributes will be based on the incremental attribute learning being researched by Danijel at UOL.

9.7 Data-Driven Component: Hand and Arm Location

Produce a description of the robot's current hand and arm pose. This is not used by any of the behaviour goals, but it will be essential for the monitoring of plan execution.

9.8 Data-Driven Component: Change and Motion Detector

Detects change on the image plane and suggests ROIs for processing based on this. We already have components supplied by TUD to do this.

9.9 Data-Driven Component: Context-Sensitive Saliency

Suggests ROIs where objects might occur in the image plane. Should possibly also be aware of where objects already are. We already have a component implemented to do this.

10 Planning Sub-Architecture

The main purpose of the planning sub-architecture for month 24 will be to plan actions that change the configurations of objects in the world. It is currently being developed by ALU-FR and BHAM. We currently have a working prototype of the sub-architecture components that are connected in a fairly arbitrary way (i.e. not using the current architecture design).

The design of the sub-architecture is pictured in Figure 4.

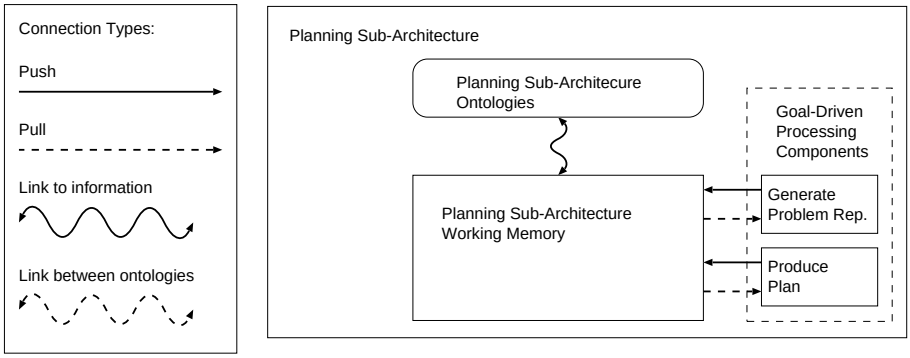

Figure 4: Planning Sub-Architecture Diagram

10.1 Behaviour Goals

1. Produce Plan for Goal. Given an input goal, produce a plan to achieve it if possible.

10.2 Information-Processing Goals

1. Produce Plan for Goal.
   (a) Generate Problem Representation.
   (b) Produce Plan For Problem.

10.3 Goal-Driven Component: Generate Problem Representation

Given the current world state (obtained from the instance memory) and goal state (e.g. to put all the square objects on the left of the table), this component will generate a planning problem in the planning domain used by the planner. Our current approach is to use the potential field spatial model from the ComSys to place object locations in appropriate places.

10.4 Goal-Driven Component: Produce Plan for Problem

This component produces a plan from the problem definition produced by the previous component. The planner being used is Michael Brenner's MAPL planner.

11 Execution Sub-Architecture

For the month 24 PlayMate the execution sub-architecture's purpose is to take a plan produced by the planning sub-architecture, and execute it whilst monitoring the outcome of each step. Execution is performed by splitting a plan into steps, then splitting each step into actions. Each level of execution is monitored by the component that does the splitting. Monitoring will involve checking the instance memory for the current state of the instances involved (or potentially involved) in the plan.

We currently have a prototype implementation of some of the functionality of this sub-architecture, but execution monitoring is not in place at any level yet.

The design of the sub-architecture is pictured in Figure 4.

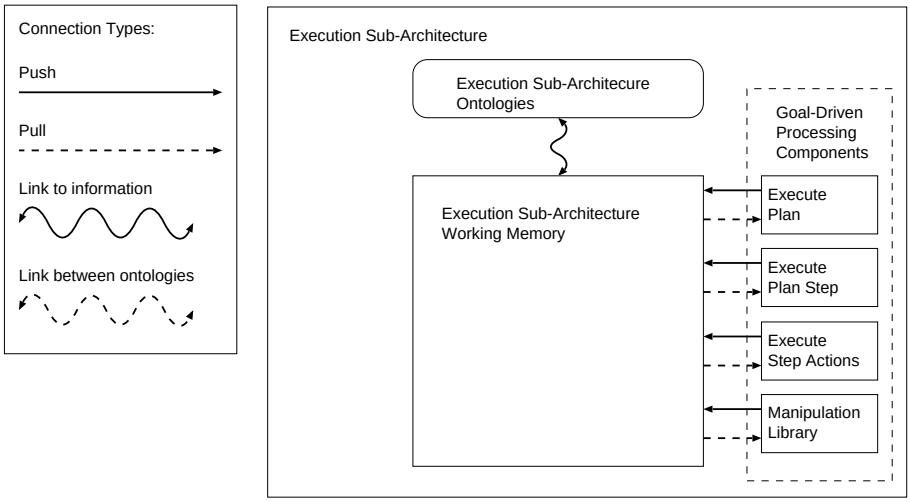

Figure 5: Execution Sub-Architecture Diagram

11.1 Behaviour Goals

1. Execute Plan for Goal. Take an input plan and execute it in the world.

11.2 Information-Processing Goals

1. Execute Plan for Goal.
   (a) Execute Plan.
   (b) Execute Plan Step.
   (c) Execute Step Actions.

11.3 Goal-Driven Component: Execute Plan

This component takes a plan and triggers then monitors the execution of each plan step. The monitoring will involve checking the preconditions and effects of each step.

11.4 Goal-Driven Component: Execute Plan Step

This component starts the execution of a particular step of the plan. This process involves translating planning actions into manipulation actions. For example, the planning action to move object X from point A to point B may need to be translated into pickup X from A, place X at B or similar.

11.5 Goal-Driven Component: Execute Step Actions

This component executes the actions for each plan step. This will involve interacting with both the planning sub-architecture, the instance memory and visual sub-architecture to translate general ontology-based instances used in the plan steps into more detailed representations suitable for action. Once the more detailed representations are build actions from the PlayMate's action library are executed and the effects monitored.

11.6 Goal-Driven Component: Manipulation Library

The PlayMate will have a library of basic manipulation commands that correspond to the actions it will need to execute to achieve its goals. Our current manipulation library is very simple and will definitely need improving!

12 Linguistic Sub-Architecture

This section is largely based on the functionality of the ComSys developed at DFKI. Although the description below describes further integration into the wider architecture, a starting point would be to just develop an interface for the ComSys so that both sides can get at the appropriate functionality.

The design of the sub-architecture is pictured in Figure 6.

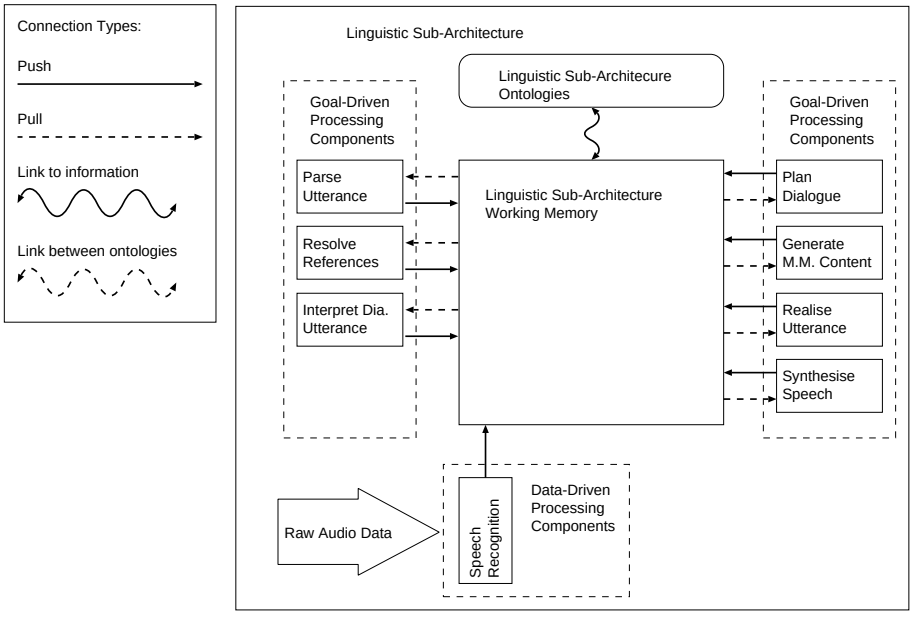

Figure 6: Linguistic Sub-Architecture Diagram

12.1 Behaviour Goals

1. Understand Utterance.
2. Produce Utterance,

12.2 Information-Processing Goals

1. Understand Utterance.
   (a) Parse Utterance
   (b) Resolve References
   (c) Interpret Dialogue Utterance.
2. Produce Utterance,
   (a) Plan Dialogue
   (b) Generate Multi-Modal Content
   (c) Realise Utterance
   (d) Synthesis Speech.

12.3 Goal-Driven Component: Parse Utterance

If an utterance is present in the LWM this component will propose a goal to parse it to extract a logical form representing its meaning.

12.4 Goal-Driven Component: Resolve References

To interpret the logical form produced by parsing, this component proposes a goal to bind any references within it to instances within the architecture. This initially involves binding references to equivalence classes within the LSA, and then to instances within the instance memory of the whole architecture. If the references need to bound to an equivalence class in another sub-architecture, then the global instance memory must be informed.

12.5 Goal-Driven Component: Interpret Dialogue Utterance

Once the utterance has been fully resolved, the purpose of the utterance can be interpreted. This should result in the PlayMate understanding why an utterance was made. This in turn could in turn result in new goals for the PlayMate.

12.6 Goal-Driven Component: Plan Dialogue

Given a set of general ontological entries that must be expressed, this component plans a dialogue move to make.

12.7 Goal-Driven Component: Generate Multi-Modal Content

This component produces the content (a logical form) for the utterance based on the dialogue move.

12.8 Goal-Driven Component: Realise Utterance

This component produces a text string from a logical form.

12.9 Goal-Driven Component: Synthesise Speech

This component outputs speech from the PlayMate based on a text string.

12.10 Data-Driven Component: Speech Recognition

The speech recognition component is the only linguistic sub-architecture component that is not controlled. It continually monitors the world for speech and attempts to recognise utterances. Any utterances are added into the linguistic working memory, and the behaviour goal Understand Utterance is proposed.

Appendices

A Worked Example: Linguistically Triggered Manipulation

In this example we have written the names and descriptions of important elements of the design **in bold**. Some of these are accompanied by references back to design sections, others are not and must therefore have their design fleshed out in future revisions.

There are four objects in the scene (A, B, C, & D) drawn from the set of objects for the PlayMate scenario. <H>is the human. <R>is the robot.

We will assume that the PlayMate has done no processing before the first utterance from the human.

<H>: "Put object A, object B & object C next to each other."

The utterance is heard by the PlayMate and is recognised by its **speech recognition component** (Section 12.10) in the **linguistic sub-architecture** (LSA, Section 12). The results of this process are placed in the **linguistic sub-architecture working memory** (LWM). This causes the motivation manager (Section 5.1) to propose a **behaviour goal** to **understand the utterance** which is then adopted by the goal manager (Section 5.2). The **parsing component** (Section 12.3) in the LSA notices the recognition result as something it can process, therefore it adds an **information-processing goal** to **extract a logical form** from the recognition result to the working memory in the LSA.

The **goal manager** determines that the goal of extracting a logical form **is a sub-goal of** understanding

an utterance and so **adopts the goal**, by entering its acceptance into the LSA working memory. This causes the parsing component to parse the utterance and place the resulting logical form (LF) into the LWM. The presence of the LF causes the **reference resolution component** (Section 12.4) in the LSA to propose the information-processing goal of **resolving the references in an LF**. Again this is a sub-goal of understanding an utterance, so it is adopted by the goal manager. The reference resolution component binds the references into equivalence classes in the local memory and also determines that the references to the objects are exophoric, and must therefore be **bound cross-modally** to be correctly resolved.

To bind these references cross-modally, for each reference the reference resolution component first queries the **instance memory** 7 to see if the reference is bound into a general instance with an equivalence class from the visual sub-architecture. This query does not happen in the goal-based manner, but through reference to the ontological entries, because it is assumed that the links formed by the instance memory exist throughout the architecture. In this case the general instances are not bound to anything in vision, so they must be.

To do this, the reference resolution component adds information to the LWM to the indicate that an additional equivalence class is required for the general instance. This is noticed by the motivation manager which adds a goal to **find an equivalence class for a general instance in a particular sub-architecture** (vision in this cases) to the goal manager. This is adopted by the goal manager (because this goal is a sub-goal of the the understanding utterance goal), and it adds the fact (possibly copied from the LWM) that an equivalence class from vision is required for a particular general instance to the working memory of the **sub-architecture search component** (see Section 7.1, used to trigger processes that may providing binding information from other sub-architectures, e.g. a visual search, a question or an information gathering action), which proposes an information-processing goal to **resolve the reference visually**. This goal is adopted and the sub-architecture search component triggers a visual search for the referent by adding all of the information known about the search target (e.g. a selection from **colour, shape, size, name** etc.) to the **visual working memory** (VWM) in the **visual sub-architecture** (VSA).

The information about the search target recognised by **visual search component** (see Section 9.3) which proposes, and has accepted, a goal of generating a series of ROIs to search. These are added to the VWM. The presence of an ROI will cause a number of possible **ROI-based components** (e.g. edge extraction) from the VSA to propose goals. The existence of an ROI with a search target will trigger a number of **ROI-based components based on the search terms** (e.g. colour-based segmentation or object extraction based on shape). The results of each search process are added to the VWM. This process (which may involve addition information-processing goals) continues until the search target is found (assuming it is), or until no more ROIs are left to search. As entities are extracted from vision they are bound into sub-architecture equivalence classes with the relevant entries for the VSA ontology. As these equivalence classes are created the global instance memory creates general instances in the instance memory and instances from different sub-architectures are bound together by the **equivalence class binding component** (see Section 7.2).

The visual search process completes when it has found all of the instances that match the search terms. The visual search component adds information to say that **the search is complete** into the VWM. This is noticed by the goal manager which passes on this fact to the sub-architectures search component. This also then adds the completion fact to its WM which again is noticed by the goal manager. The goal manager then informs the reference resolution component in the LSA that the search for equivalence class has completed so it can continue. This results in a **fully resolved LF**

being added into the LWM. The presence of this triggers the **dialogue interpretation component** to suggest a goal to interpret the utterance in terms of its purpose in the dialogue. which is adopted. The result of the interpretation is that the utterance is understood as a command to act on the world to achieve a particular result. The actual command is placed into the LWM. This is noticed by the **global motive generator** which generates a **task goal** to **respond to the utterance**, which in this case involves acknowledging the command then following it.

First the motive manager adds the **behaviour goal** to **acknowledge the previous utterance** to the goal manager. This triggers a series of adopted information processing goals in the linguistic sub-architecture for the **dialogue planning component**, **multi-modal content planning component**, **realisation component** and **speech synthesis component** and results in the robot speaking:

<R>: "OK."

Next the global motive generator adds the behaviour goal to **produce a plan for the action command** to the **action planning sub-architecture** (ASA). The translation from the resolved and interpreted utterance is made possible by mappings between the linguistic sub-architecture **ontologies for instructions, actions and spatial references** to general ontological entries that can be converted into planning goals and internal instructions.

The first adopted goal in the planning sub-architecture is for the **planning problem generation component** (see Section 10.3) which dynamically produces a discrete planning problem based on the current visual scene and the goal configuration. It does this by interacting with the instance memory to determine the extent of the scene and the current positions of the objects within it, and combines this with calculations from the **spatial modelling ontology** (which can model left of, right of, near etc.) to produce a **planning problem** based on **2D waypoints** on the table top. This planning problem is added to the **planning working memory**, where it is noticed by the **action planning component** (see Section 10.4) which proposes a goal to produce an **action plan** for solving the problem. This goal is adopted because it is a sub-goal of sub-architecture's current behaviour goal (produce an action plan).

The presence of the plan causes the global motive generator to add a behaviour goal to **execute the action plan** to goal manager. This in turn results in the plan being added to the **execution working memory** (EWM) in the **execution sub-architecture** (ESA, see Section 11). This triggers the proposal of a series of goals by the **plan execution component** (see Section 11.3) to **execute each step of the plan**. For each step goal (which should be adopted in turn given the successful completion of the previous step) a goal is proposed by the **plan step interpretation component** (see Section 11.4) which translates each step into at a number (zero or more) of goals to execute particular **action commands** which are then followed by the **action command component** (see Section 11.5) to act out the commands. The interplay between these components is quite important, because the action command component must monitor execution to check that each command is performed correctly, the plan step interpretation component must check that this results in satisfaction for each plan step, and the plan execution component must check that the plan step has satisfied the relevant parts of the overall plan.

<R>: Creates a plan. The plan it produces involves moving object A close to the left side of B, and object C close the right side of B.

<R>: Executes the first step of this plan by grasping object A, picking it up, then putting it down near

to the left side of B.

<H>: Before <R>can grasp object C, <H>picks up object A, and places it to the right of object B.

The VSA **notices that change is occurring** (see Section 9.8) and proposes a series of ROIs to check. These are processed in a similar way to previously (i.e. various process that result in the identification of the new positions of the previously seen objects), and the result is that the instance memory is updated with the new positions of the objects in the world. The plan execution component must **continually check the preconditions of its plan** with the instance memory. When it notices that object A is now in the place it intended to place object C (some precondition that would state that no object is at the waypoint), it triggers a **replanning** process given the current state of the world and the same goal state as previously. This is done initially without regenerating a discretisation, although if the planning process fails then this may be necessary. The new plan is then executed in the same way as previously.

<R>: Attends to event, concludes that the new position of object A still satisfies the goal, but that the planned position of object C is no longer free. It then decides to place object C somewhere else that is near object B and acts out this plan.

References

[Kruijff et al., 2006] Kruijff, G.-J. M., Kelleher, J. D., and Hawes, N. (2006). Information fusion for visual reference resolution in dynamic situated dialogue. In Andre, E., Dybkjaer, L., Minker, W., Neumann, H., and Weber, M., editors, *Perception and Interactive Technologies (PIT 2006)*. Springer Verlag.