

FP6-004250

CoSy

Cognitive Systems for Cognitive Assistants

Integrated Project

Information Society Technologies

DR.10.5

Status of implementation for the Playmate system

Due date of deliverable: 31/08/2006

Actual submission date: October 6, 2006

Start date of project: September 1st, 2004

Duration: 48 months

Organisation name of lead contractor for this deliverable:

University of Birmingham

Revision: Final v1

Dissemination Level: PU

Executive Summary

This provides a summary of the status of the PlayMate system at the end of year 2. We outline the process we went through in designing the targets for the Month 30 deliverable, and the progress on the science and the software components. We are currently integrating the agreed components using the architectures toolkit (CAAT) according to the schedule planned.

Role of the Playmate implementation in Cosy

The Playmate is one of the two scenario driven demonstrators within the project. One of the important project aims is to conduct empirical investigations on architectures and other integration issues. This report summarises the status of the current implementation of the Playmate system. We detail both the progress on components, and on integration, including the plan up to the 30 month date for delivery of an integrated system.

Status of implementation for the Playmate system

Jeremy Wyatt

October 6, 2006

1 PlayMate Integration: the planning process

We held a meeting of consortium members (BHAM, DFKI, TUD, UOL, FREI) involved in the PlayMate scenario between January 16-19 2006. At this meeting we agreed that work would proceed in parallel on several sub-scenarios or sub-problems, to be integrated into a single demonstrator for Month 24. These were:

1. Cross modal acquisition of categories (TUD,DFKI).
2. Cross modal learning about objects and ontologies (UOL,BHAM,DFKI)
3. Simple collaborative manipulation (BHAM, FREI, DFKI)

This led to the scientific work on each sub-problem being carried out at the partner sites between February and August 2006. At the same time BHAM presented a draft design for the architectural framework to be employed in the PlayMate integrated system to the consortium at the consortium meeting in April 2004. This was revised and a full design presented and discussed at a meeting on Architectures held on June 16 in Frankfurt.

2 PlayMate implementation

In addition to the work going on at each partner site we built a succession of integrated systems. Between January and May of 2006 an initial integrated system for manipulation and planning was completed, this supported a paper on planning and representations accepted to IJCAI. Following this the integrative work turned to the development of the architecture framework, and its adoption.

The design of the architecture framework used in the PlayMate demonstrator is split into two levels. The lowest level is a process communication framework called BALT (Boxes And Lines Toolkit). It allows CORBA based process invocation and communication using push and pull mechanisms and underpins all the more complex architectural mechanisms. This is currently in release V.0.5.1. BALT was initially released to the consortium on 12 July 2006.

The second layer of the architecture framework is CAAT (Cosy Abstract Architecture Toolkit). CAAT implements a collection of architectural mechanisms that allow rapid prototyping and integration of cognitive architectures. CAAT was started by BHAM in mid July and the first version was available in late August. The first official release was on Sept 13th.

This has been adopted for the current PlayMate integrated system. This system is currently in the process of integration, with a deliverable date for a report of Month 30. We plan to demonstrate an early version of the system at the November review meeting, and are integrating to that end.

A schema for the PlayMate integrated system, as described in the architectures deliverable DR.1.2 is shown in Figure 1.

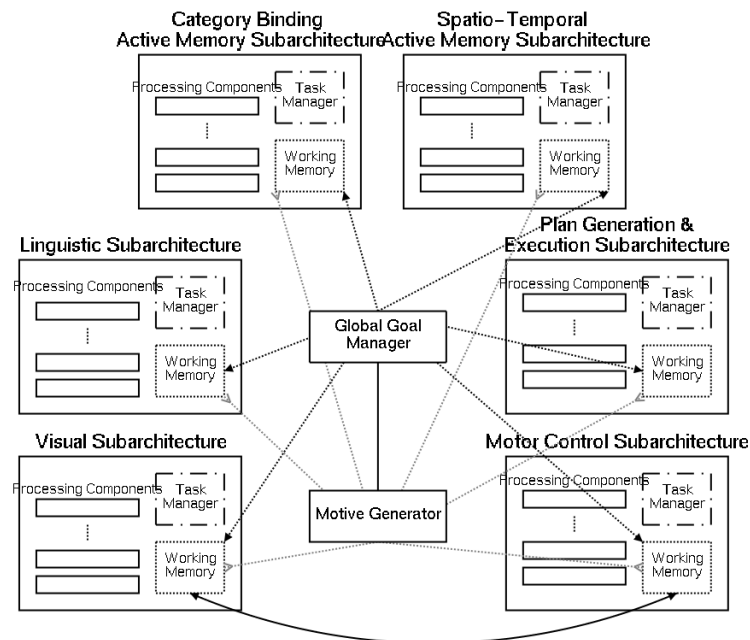


Figure 1: A schema for the architectural instantiation for the PlayMate.

As of mid-July the consortium members involved in working on the PlayMate integrated system have been wrapping existing components using BALT. We list the components that have been delivered, and revisions in an appendix. The consortium meeting in Birmingham on Sept 11 coincided with a PlayMate integration meeting from 9-14 Sept. At this the designs and integration timetable for the integration of the existing components were agreed. The project chart for short-term integration for cross modal ontology learning, and collaborative manipulation is attached as **Appendix 1**, this includes a detailed record of progress against tasks as of 27/09/06.

To make the structure of the system clearer we attach (as **Appendix 2**) a draft design document developed between April and the beginning of August 2006, that we used in the design process. This should make clearer how the PlayMate integrated system for Month 30 will be an instantiation of a general architecture framework. It is the case that the design is evolving, so the final design will have some component and architectural differences. An instance of this is that additional components to allow incremental processing across sub-architectures has recently been added. Another example is that the design now also has several general memories, rather than one (as shown in Appendix 2). These can be seen in Figure 1.

2.1 Progress on delivery of PlayMate components

A large number of software components are required for this system, which integrates several large previous pieces of work, including much of the Communication System demonstrated at the year 1 review. Some pieces of software have not been given official releases, but are being updated and maintained using a central subversion repository. In this case we give the approximate date at which components passed milestones. Components are listed by partner:

- DFKI: ComSys Mk2. 24/11/2005.
- UOL:
 1. Extraction of 3D point clouds from stereo, segmentation of surfaces, parameterised surface fitting. 11/08/06.
 2. System for cross modal ontology learning. 11/08/06.
- BHAM:
 1. BALT: Framework for rapid prototyping of systems with flexible process communication. Version 0.1.1 released 12/07/06, Version 0.2 released 18/07/06, Version 0.3 released 04/08/06, Version 0.3.1 released 11/08/06, Version 0.4.0 released 29/08/06, Version 0.5.1 released 22/09/06.
 2. New firmware and libraries for low level trajectory generation for Katana arm. Version 0.6.12 released 29/04/06.
 3. CAAT: Cosy Abstract Architecture Toolkit. Version 0.1.0 13/09/06.
- ALU-FR: Software for high level action planning 6/03/06. Revision of same, with execution monitoring and replanning software 22/09/06.

3 Summary

The first year PlayMate system concentrated on how vision and language can be connected via ontology based mediation. In the second year the PlayMate system will use manipulation together with language and vision to show how the architectural mechanisms we have devised allow incremental processing, mediation of representations between sub-architectures, and flexible responsive behaviour from the robot in the face of world change. The short-term integration is broadly on target, and we are confident that we will have a full system, which has been thoroughly analysed and experimentally profiled by the Month 30 deliverable date.

Appendices

3.1 Appendix 1: Short Term Integration Plan

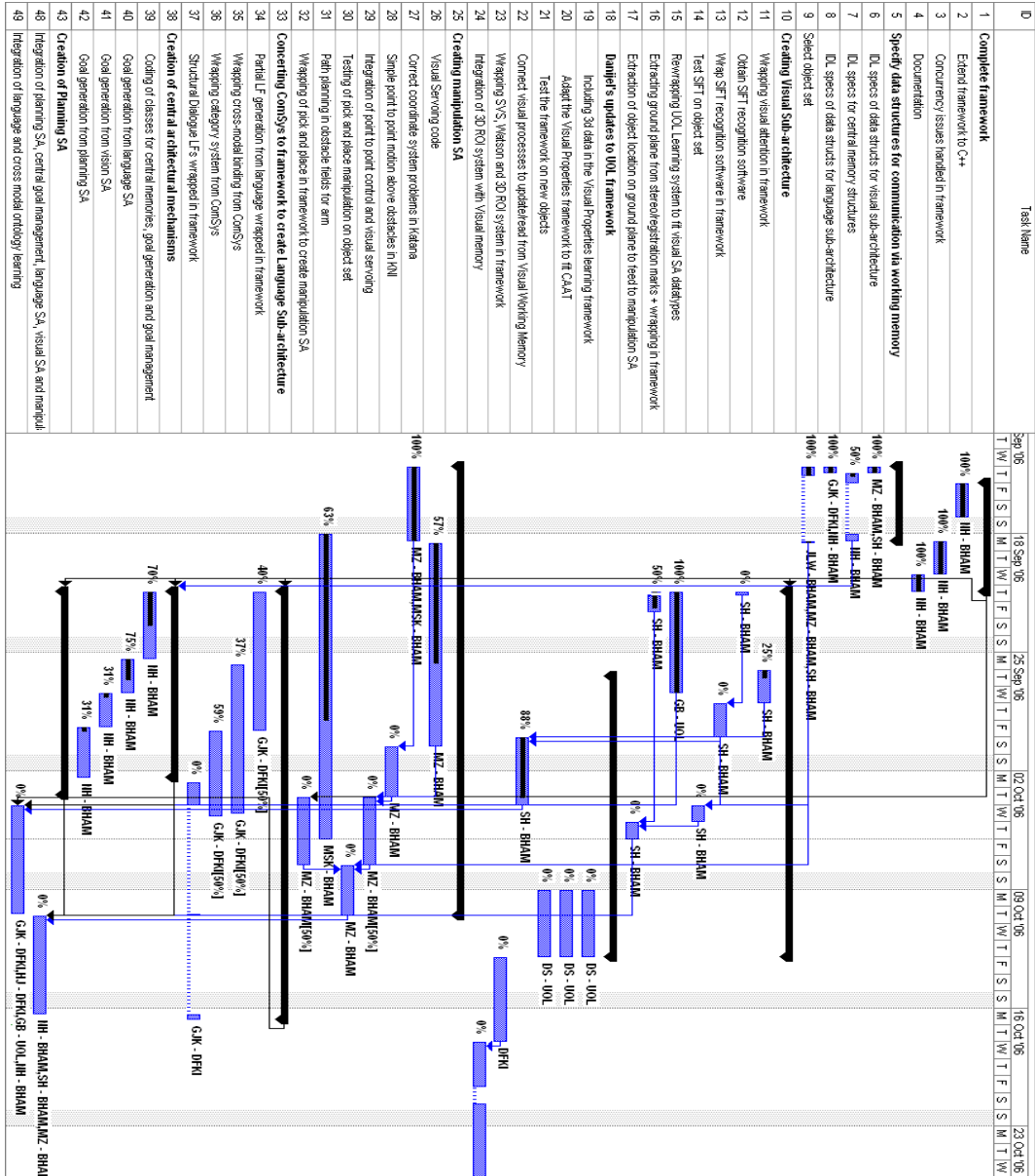


Figure 2: The PlayMate short term integration project plan.

3.2 Appendix 2: PlayMate High Level Design At 1st August 2006

A PlayMate instantiation of the Abstract Architecture Design

Nick Hawes
1/08/06

This section of the document describes a scenario-specific instantiation of the previously presented abstract architecture design that is intended to satisfy a particular (limited) version of the the CoSy Playmate scenario. In this scenario a human is able to command the a robot to move objects around on a table. A schematic overview of the whole architecture design, including the general components, can be seen in Figure 3. At the time of writing only parts of the architecture instantiation have been implemented. This is the reason why some components and sub-architectures are described in greater detail than others.

In the subsequent description of the architecture instantiation, all of the goal-driven components can only act on information if they suggest an information-processing goal and have it adopted by either the global goal manager, or as is more likely, their sub-architecture goal manager.

4 Visual Sub-Architecture

In the architecture, the visual sub-architecture is chiefly concerned with providing descriptions of objects to allow linguistic references to be resolved, and extracting 3D information about the scene to facilitate manipulative actions. The descriptions will include the visual attributes of objects (such as shape, colour and size), and these will be used, along with spatial information, to disambiguate objects when they are involved in a manipulation command. The design of the sub-architecture is pictured in Figure 4.

The visual sub-architecture is driven by a data-driven visual attention component which suggests regions-of-interest (ROIs) to be processed based on change, motion and salience. These ROIs will then be processed to obtain information on any objects they contain. This processing involves fast SIFT recognition to get initial results, then 3D segmentation if manipulation is required.

4.1 Data-Driven Component: ROI Proposal

The regions-of-interest proposal component suggests ROIs for processing based on visual change or motion in the input video stream, or on context-sensitive visual salience [3].

4.2 Goal-Driven Component: 3D Segmentor

This component produces a 3D point cloud of the current scene, then segments surfaces from this. The purpose of this is to obtain a representation of the shape and pose of the objects in the scene so that their position can be described, and that they can be manipulated by the robot.

4.3 Goal-Driven Component: Object Features

Given a particular object, part of object, or surface, this component returns a list of the visual properties of this entity. This component will be based on an incremental visual attribute learning system.

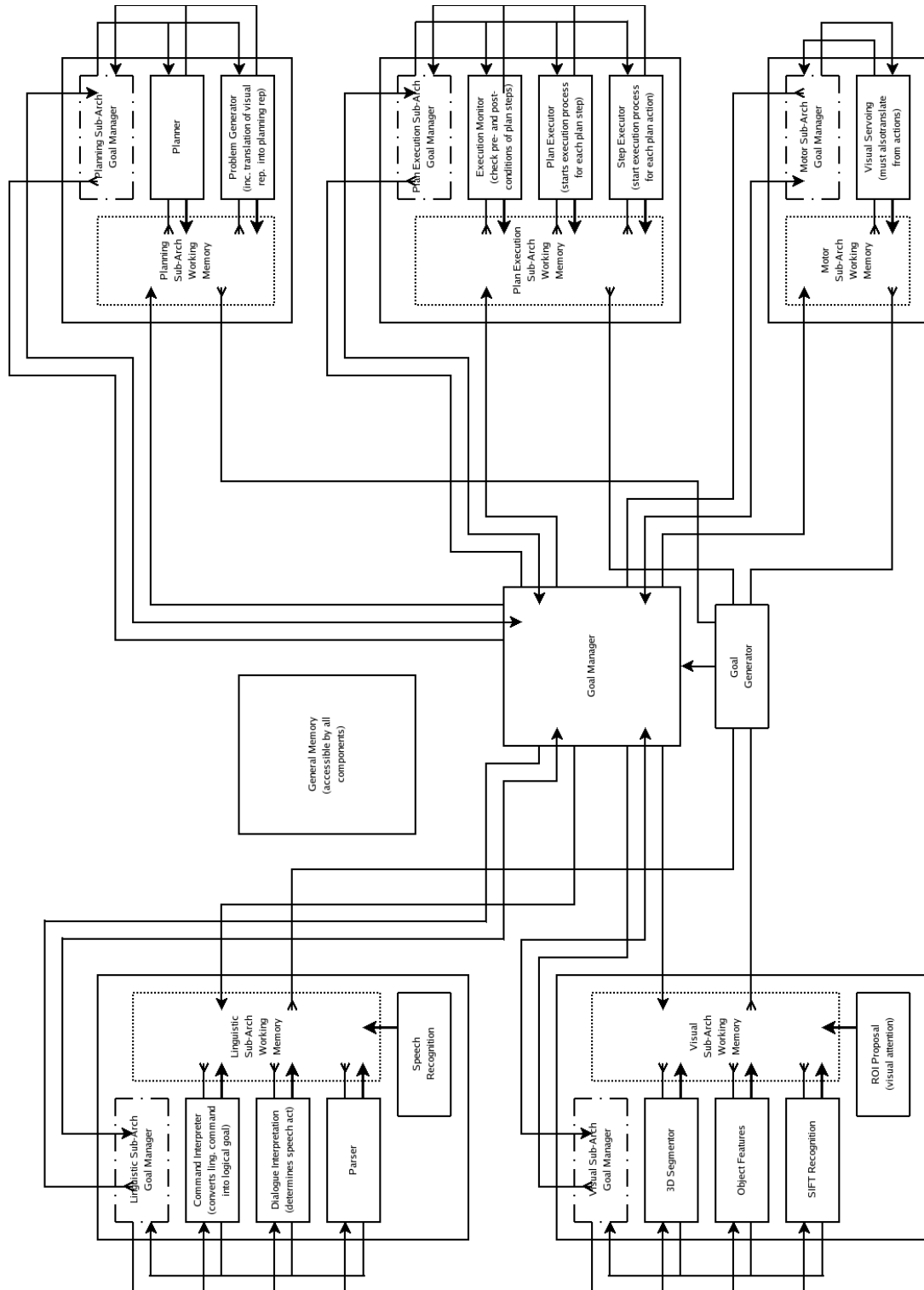


Figure 3: Architecture Overview

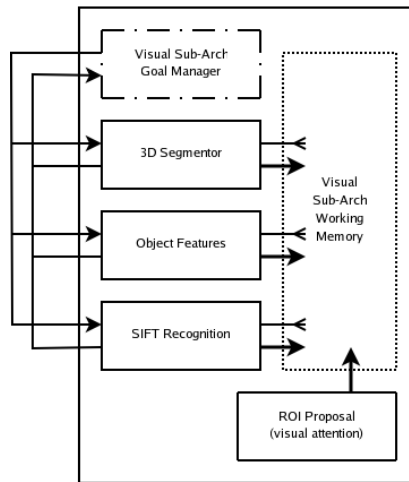


Figure 4: Visual Sub-Architecture Diagram

The learning process will initially be performed separately to the object manipulation scenario, but will be integrated at a later stage.

4.4 Goal-Driven Component: SIFT Recognition

This component runs a SIFT recognition system on a given ROI and returns any previous learned labels associated with recognised SIFT vectors [6]. This component is intended to provide information about the ROI in a rapid manner to help determine whether more expensive processes should be run.

5 Planning Sub-Architecture

The main purpose of the planning sub-architecture is to produce plans that the system can execute in the world. To do this it has a component that translates the central representation used by the system into the domain used by the planner, and a component that actually creates the plan. The planning sub-architecture is not strictly tied to the current design scenario, as most of the scenario-specific knowledge is tied to the problem generator and the planning domain, rather than the planner itself. The design of the sub-architecture is pictured in Figure 5.

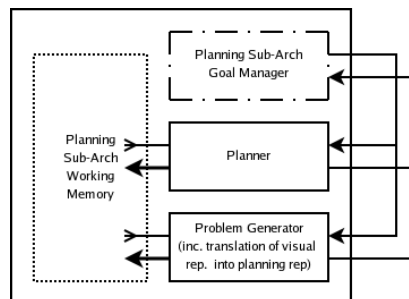


Figure 5: Planning Sub-Architecture Diagram

5.1 Goal-Driven Component: Problem Generator

Given a representation of the current world in the form used in the general memory, and representation of a goal state (e.g. to put all the square objects on the left of the table) as produced by interpreting a human's command (see Section 8.4), this component will generate a planning problem in the domain used by the planner. The planning language used is MAPL [1].

5.2 Goal-Driven Component: Produce Plan for Problem

This component produces a plan from the problem definition produced by the problem generator component. To do this the component first compiles the MAPL problem description into a PDDL [2] description which is then used as input to the FF planner [4].

6 Execution Sub-Architecture

The purpose of the execution sub-architecture 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. At the start and end of each plan step, an additional monitoring component checks whether the plan is still valid given the current world state. The design of the sub-architecture is pictured in Figure 5.

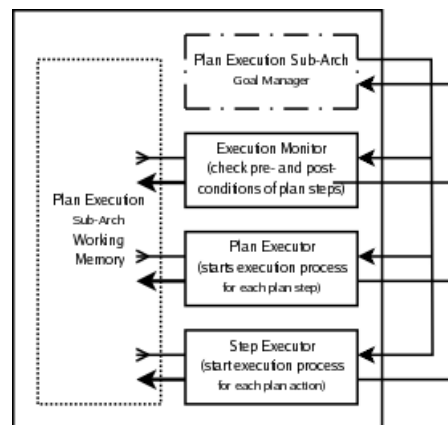


Figure 6: Execution Sub-Architecture Diagram

6.1 Goal-Driven Component: Execute Plan

This component takes a plan produced by the planner component in the planning sub-architecture and starts the process of plan execution. This component has responsibility for starting the execution of each individual plan step. This is done by writing the step into the sub-architecture working memory when the previous step has been completed successfully.

6.2 Goal-Driven Component: Execute Plan Step

This component starts the execution of a particular step of the plan. This process involves translating operators used in the planning domain into actions (which could include information-processing actions internal to the architecture) that are executable by system. 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. Actions are written into the working memory of the sub-architecture, where the motive manager (external to the sub-architecture) will use them to propose a goal to have the action executed by the appropriate sub-architecture.

6.3 Goal-Driven Component: Execution Monitor

The role of the execution monitor component is to verify whether a future plan step can still be executed, and whether a completed plan step has had the desired effect. It does this by comparing the preconditions or effects (depending on whether it is checking a future or past plan step respectively) of the step to the system's current model of the world. It is triggered when a proposed or completed plan step appears in the sub-architecture working memory. The result of the monitoring process will determine whether the plan step execution component proceeds to execute the proposed step, and whether the plan execution component proceeds to execute the next step in the plan.

7 Motor Sub-Architecture

The purpose of the motor sub-architecture is to execute motor commands produced by other components in the architecture. In this scenario the only motor commands are commands to perform manipulation actions on objects in the world. Future iterations of the architecture could contain components to move a robot around or move other limbs (such as a pan-tilt unit). The design of the sub-architecture is pictured in Figure 7.

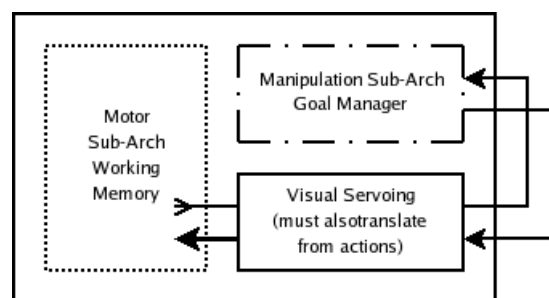


Figure 7: Motor Sub-Architecture Diagram

7.1 Goal-Driven Component: Visual Servoing

This component has a library of basic manipulation commands that correspond to the actions it will need to execute to achieve its goals. As well as input describing the action to perform and its parameters, the visual servoing component must communicate closely with the visual sub-architecture to determine the rough positions of any objects it must act on, and then to get visual feedback during the

process of visual servoing. This necessitates a specialised shared working memory connection which is described in Section 9.1.

8 Linguistic Sub-Architecture

In this design the role of the linguistic sub-architecture is purely to interpret action commands given by a human to the system. The initial scenario does not require the system to give feedback to the human, so that is why the sub-architecture represents an input-only pipeline. The design of the sub-architecture is pictured in Figure 8. It is based on an initial place-holder system developed for previous work, and is not related to the CoSy ComSys [5].

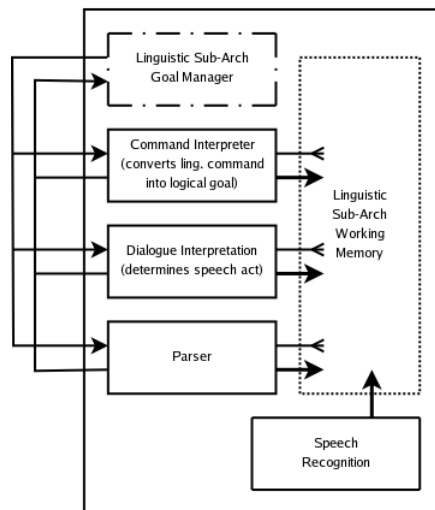


Figure 8: Linguistic Sub-Architecture Diagram

8.1 Data-Driven Component: Speech Recognition

The speech recognition component provides linguistic input to the linguistic sub-architecture. It continually monitors the world for speech and attempts to recognise utterances. Any recognised utterances are added into the linguistic working memory,

8.2 Goal-Driven Component: Parser

If an utterance is present in the working memory this component parses it to extract a logical form representing its meaning.

8.3 Goal-Driven Component: Dialogue Interpretation

Once an utterance has been parsed, the dialogue interpretation component attempts to understand why it utterance was made. In this scenario, the component only looks for utterances that represent manipulation commands. If this is the case, then the result of the initial parse is written back to working memory.

8.4 Goal-Driven Component: Command Interpreter

If a command appears in working memory, this component attempts to convert the command into a MAPL representation that can later be used as input to the planner. This component is placed in the linguistic sub-architecture because it operates on the (specialised) output of the parser to produce a representation that can be understood by other sub-architectures (i.e. a more abstract logical form).

9 Shared Working Memories

This section describes the specialised shared working memory connections that exist in the architecture.

9.1 Visual Servoing Working Memory

The visual servoing component requires access to parts of the internal representation used by the visual sub-architecture in order to perform accurately. In particular it needs to access detailed quantitative information about the position of object surfaces in space, and rapid visual feedback of the effects of moving the system's effector in space. Because of the level of detail of the former, and the speed required for the latter, the architecture requires a shared working memory between the two involved sub-architectures.

Whilst visual-servoing is taking place, the visual sub-architecture writes its output to this shared working memory, rather than its own sub-architecture-specific working memory. This allows the components in the motor sub-architecture fast, direct access to the information produced by visual components without going through a central mechanism.

10 General Architecture Components

The following sections describe the roles of the general architecture components in the scenario. It is evident from these descriptions that the components require problem-specific knowledge to perform effectively, although there are also potentially general-purpose mechanisms at work too (e.g. the goal manager examining goal-subgoal relationships). Initially the problem specific knowledge will be given to these components before run time, but there is nothing in the abstract architecture design to stop this information from changing at run time as the whole system gains experience.

10.1 The Motive Manager

In this scenario, the motive manager monitors the sub-architecture working memories for the following conditions:

- A MAPL description of an action command appearing in the linguistic sub-architecture working memory. If this condition occurs, it copies the description to general memory and proposes a task goal for the system to follow the command.
- A completed plan appearing in the planning sub-architecture working memory. If this condition occurs, it copies the completed plan to general memory and proposes a task goal for the system to execute the plan.

- An action description appearing in the plan execution sub-architecture. If this condition occurs, it copies the description to general memory and proposes a task goal for the system to execute the action.

10.2 The Global Goal Manager

In this scenario, the global goal manager monitors performs the following goal management tasks:

- If a goal is proposed to follow an action command it accepts it if no other command is being followed, and if no physical action has been undertaken if another action command is currently being followed. To follow an action command it must write the MAPL description of the command to working memory in the planning sub-architecture.
- If a goal is proposed to execute a plan it accepts it if it is currently pursuing a goal to follow a command, and executing the plan is subgoal of following the command. To execute a plan it must write the plan to working memory in the plan execution sub-architecture.
- If a goal is proposed to execute an action it accepts it if it is currently pursuing a goal to execute a plan and executing the action is a subgoal of executing the plan. To execute an action it must write the action to working memory in the motor sub-architecture.

The goal manager must also manage monitoring the success and failure of some of the processing that is occurring in the sub-architectures. This is particularly critical for plan execution as it is important that execution stops when either the motor sub-architecture fails to execute an action, or if the execution monitor in the planning sub-architecture detects that the plan is no longer executable.

References

- [1] Michael Brenner. Planning for multiagent environments: From individual perceptions to coordinated execution. In *ICAPS-05 Workshop on Multiagent Planning and Scheduling*, 2005.
- [2] M. Fox and D. Long. PDDL 2.1: an extension to PDDL for expressing temporal planning domains. *JAIR*, 20:61–124, 2003.
- [3] Nick Hawes and Jeremy Wyatt. Towards context-sensitive visual attention. In *Proceedings of the Second International Cognitive Vision Workshop (ICVW06)*, Graz, Austria, May 2006.
- [4] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *JAIR*, 14, 2001.
- [5] Geert-Jan M. Kruijff, John D. Kelleher, and Nick Hawes. Information fusion for visual reference resolution in dynamic situated dialogue. In Elisabeth Andre, Laila Dybkjaer, Wolfgang Minker, Heiko Neumann, and Michael Weber, editors, *Perception and Interactive Technologies: International Tutorial and Research Workshop, PIT 2006*, volume 4021 of *Lecture Notes in Computer Science*, pages 117 – 128, Kloster Irsee, Germany, June 2006. Springer Berlin / Heidelberg.
- [6] D. G. Lowe. Object recognition from local scale-invariant features. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 1150–1157 vol.2, 1999.