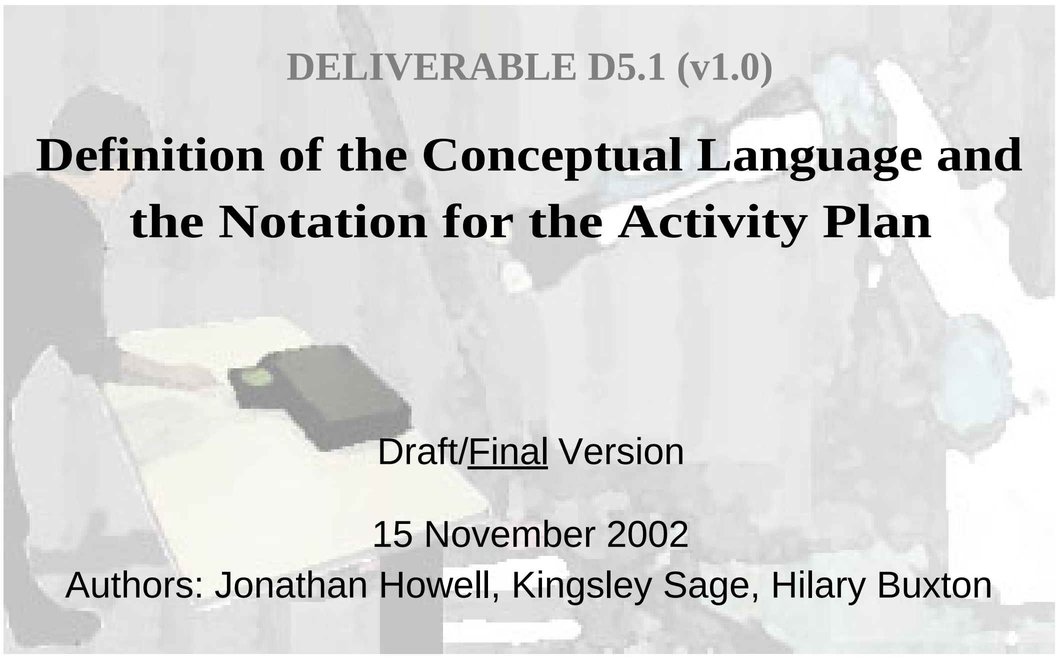


DELIVERABLE D5.1 (v1.0)

Definition of the Conceptual Language and the Notation for the Activity Plan

Draft/Final Version

15 November 2002

Authors: Jonathan Howell, Kingsley Sage, Hilary Buxton

Project acronym:	**ACTIPRET**
Project full title:	**Interpreting and Understanding Activities of Expert Operators for Teaching and Education**
Action Line IV.2.1:	**Real Time Distributed Systems (Cognitive Vision)**
Contract Number:	**IST-2001-32184**

Contents

1 Introduction 3
2 Conceptual Language and Activity Definitions within the ActIPret Demonstrator 3
3 Definition of Conceptual Language 4
3.1 Overview 4
3.2 Formal definition of Conceptual Language 4
3.3 Example Conceptual Language File 4
3.4 From Conceptual Language to Activity Plan 5
4 Activity Definitions 9
5 Relationship to Cognitive Vision Framework 11

1 Introduction

This deliverable document provides details of the Conceptual Language (CL) and activity definitions used by the ActIPret Demonstrator (AD). Section 2 describes how the CL and activity definitions relate to the AD functional components. Section 3 contains a formal abstract definition of the CL together with an example for the initial scenario. Section 4 provides an overview of how activity definitions embedded with the Activity Reasoning Engine (ARE) are used to create plan elements using the CL. Section 5 describes how this work relates to the overall principles and requirements of the Cognitive Vision (CV) framework.

2 Conceptual Language and Activity Definitions within the ActIPret Demonstrator

The Conceptual Language (CL) provides all of the domain specific knowledge for a scenario that is required by the ActIPret Demonstrator (AD). It specifies the types of objects that the Activity Reasoning Engine (ARE) will be able to produce instantiated concepts for and the form of the language output of the ARE to the Activity Plan Generator (APG). The CL also specifies the form of the language output from the ARE to any other components e.g. VR reconstruction, if the requirements of that language output are different to that provided to the APG.

Output from the ARE represents the actions events and activities in the scenario as detected by the AD, derived from the control exercised by the ARE over the lower level components. The CL therefore specifies the language form of the native activity plan that is generated by the AD as it observes the scenario. The CL does not, however, specify the control mechanisms that the ARE uses to generate these actions, events and activities. These activity definitions are a function of the ARE.

Activity definitions are domain independent generic activity (vision based) definitions that reside in the ARE that contain information about specific pre-conditions and temporal sequencing within individual activities. For example, an activity definition for ‘pickup’ includes control data about which components should be selected to determine an instance of that activity and what top down control needs to be exercised to detect that activity efficiently.

Activity definitions within the ARE will be re-usable with different scenarios. In order to use the AD with a different scenario, it is necessary only to change the CL input specification.

The CL needs to be able to efficiently specify the proposed manipulation tasks that the ActIPret system is proposing to interpret. The efficiency of this representation will depend on the granularity of the concepts used - too high ('to put a CD into a CD-player, you put a CD into a CD-player') leads to no explanation being available for training new users, too low ('to put a CD into a CD-player, you find a CD, make sure your hand is empty, move your hand near the CD, open your hand, etc') leads to too much explanation being available. The level of granularity of individual statements made in the conceptual language will therefore be related to the required specificity of explanation, i.e. what level of instructions will be most intuitive to the human user.

The initial version of the CL presented here is constrained to independent action types. That is, there is no scope for refinement of any of the action types into more specific forms that may be more appropriate and intuitive for the human expert. A future refinement of the CL will allow for hierarchical relationships between action types. For example, putdown is a generic action type (e.g. X puts Y down on Z) that can encompass a hand placing a CD on a table top, or putting a book on a shelf and so on. A more specific action type such as loading can be derived from putdown where it refers to any hand putting down any CD on any CD player (i.e. specific categorisations exist for each action agent).

3 Definition of Conceptual Language

3.1 Overview

The Conceptual Language (CL) specifies:

- the types of objects that the Activity Reasoning Engine (ARE) will be able to produce concepts for (using <object_type> declarations);
- properties of those objects that will enable the ARE to exercise an efficient control strategy over the lower level components within the ActIPret Demonstrator (AD) (using a <properties_list> embedded within the <object_type> declaration);
- the form of the language output of the ARE to the Activity Plan Generator (APG) (using <action_type> declarations>;
- the form of language output from the ARE to any other components e.g. VR reconstruction if the requirements of that language output are different to that provided to the APG (using <action_type> declarations).

3.2 Formal definition of Conceptual Language

A simple formal definition of the Conceptual Language (CL) is provided in Box 1:

```
start(<ScenarioName>)

# Comment s may appear anywhere in the file
<declarations>

end(<ScenarioName>)

<declarations> =
<object_type1>, …,  <object_typeL>
<action_type1>, … , <object_typeM>
[opt]<subpart_type1>, … , <subpart_typeN>

<object_type> =
       object_type{<object_name>, <object_ref>, <object_model>,
              <properties_list>, [opt] <subparts_list>}

<action_type> =
       action_type{<APG_language_form>, <natural_language_form>}

<subpart_type> =
       subpart_type{<lingustic_description>, <part_name_from_object_model>}
```

Box 1: Simple formal definition of the Conceptual Language

3.3 Example Conceptual Language File

An example Conceptual Language definition file for the CD scenario is presented in Box 2. This file provides specific information about the types of objects for the scenario and their properties, and the generic activities that can be performed on them:

```
start(playcd)

object_type{“Hand”, 0,  handmodel,
       properties={hand, container,singular,!pickupable,!subparts}}

object_type{“CD”, 1, cdmodel,
       properties={!hand,!container,!singular,pickupable,!subparts}}

object_type{“CD-Player”, 2, cdplayermodel,
       properties={!hand,container,singular,!pickupable,subparts},
       subparts={power_button,play_button,eject_button  })

object_type{“Desk”, 3, nondef, properties={}}

subpart_type{ “Power Button”, power_button}
subpart_type{“Play Button”, play_button}
subpart_type{ “Eject Button”, eject_button}

# In this example there is only one hand

action_type{(BUTTON_PRESS,X,Y,Z):
       “Use Hand to press the Z on the Y.”
       }

action_type{( HAND_PICKUP,X,Y,Z):
       “Use Hand to pick up the Y from the Z.”,
       }

action_type{(HAND_PUTDOWN,X,Y,Z):
       “Use Hand to put the Y down on the Z.”
       }

end(playcd)
```
Box 2: Example Conceptual Language definition file

3.4 From Conceptual Language to Activity Plan

3.4.1 Structure of the Activity Plan

The observed scenario is represented by the ActIPret system in the Activity Plan Generator (APG) that provides all relevant information about how the task is performed. In its native form (i.e. when considered as distinct entity divorced from the Conceptual Language CL) the APG contains two distinct sections:

- The first represents the declarative semantics of the task – information about objects referred to during the description of the task and optionally, specifying behavioural models for these objects. These declarative semantics are taken directly from the CL that was used to generate the APG.
- The second contains individual statements, the form of which is specified by the CL, describing the specific sub-tasks required to fulfil the overall task and their temporal sequencing.

This second part can be considered at 3 levels:

- **The atomic plan primitive:** this is an instantiated concept function generated by the ARE. The atomic plan primitive is derived from of a combination of vision primitives (actions, activities or events) defined within the ARE. The primitive cannot be decomposed further at the planner level.
- **The intermediate instruction:** this represents an instantiated non-empty set of atomic plan primitives which is a significant part of a scenario, such as a single concept, e.g. 'pick up an object from somewhere' or 'press a button on an object', or a

combination e.g. 'Put the CD in the CD-player' (requires pick-up and put down). The sequence of instructions which represent a particular scenario is termed an activity plan.

- **The scenario:** this is an instantiated non-empty set of intermediate instructions and provides a general description of the task, such as 'play a CD on a CD-player' or 'make a cup of tea'. This level could be used to create an index of all known scenarios each one of which a user could select in order to find tutorial directions.

We can visualise a single scenario exemplar with end-to-end significance as a linear finite state automaton:

Key:

Triangles represent plan start and end points
Black circles represent plan atomic primitives

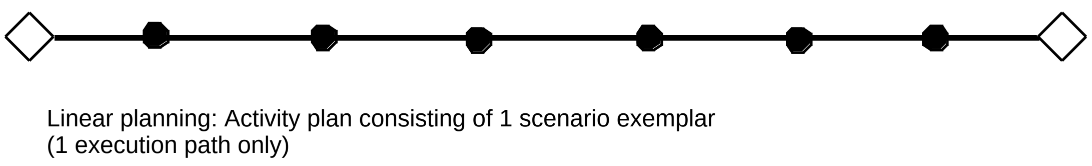

Linear planning: Activity plan consisting of 1 scenario exemplar
(1 execution path only)

Figure 1: Single scenario exemplar represented as a Finite State Machine.

We can visualise a conceptual hierarchy with intermediate plan concepts formed from ordered sequences of atomic plan primitives as a graph. Acquisition of the scenario exemplars could now occur in smaller sections as defined by the intermediate plan concepts:

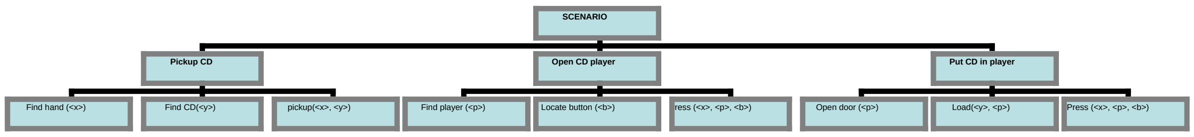
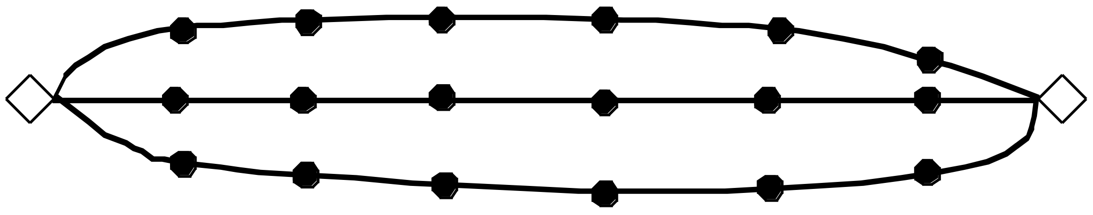


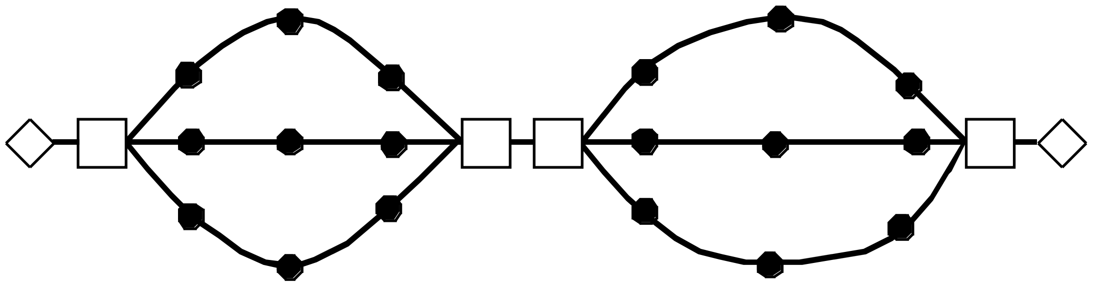
Figure 2: Hierarchical representation of plan concepts

The expert demonstrates each intermediate plan concept independently. This could be achieved either as part of the learning phase (the notion of a learned sequence) or through more complex flow control within the expert mode. Finally, the expert demonstrates/specifies the intermediate concept sequencing to complete the plan. Now in tutor mode it is possible to recognise an end-to-end exemplar that is composed of any combination of the sub-structure ordered sequences. The hierarchical approach can be represented graphically as finite state automata:

Key:
Triangles represent plan start and end points
Black circles represent atomic plan primitives
Square represent intermediate concept start and end points



Linear planning: Activity plan consisting of 3 scenario exemplars



Hierarchical planning: Activity plan consisting of 2 intermediate plan concepts each with 3 exemplars.

Figure 3: More complex types of planning represented as Finite State Machines

3.4.2 Types of atomic plan primitive

Body movements

For the proposed manipulation tasks, the important information about the user's activity will be related to what they are doing or are intending to do with objects in the scene, e.g. picking up, putting down, rotating an object, pressing buttons, etc. Other movements, such walking, rotation of body to face a new direction, movement of hand or body to align towards an object, are predictive of an activity but not a task-relevant activity in themselves. For instance, the action 'open the door' may require the expert to move to and align their body with the door before actually holding and turning the door handle, but how much movement is required for that alignment will be related to where they start from, not to the task itself.

Non-alignment movements, such as standing up or sitting down are similarly predictive of an activity, rather than a task-relevant activity.

Manipulation of static 'non-pickupable' objects

Since static objects cannot be moved, the types of manipulation possible with them will be concerned with changing some internal state of theirs, e.g. 'Pressing the 'on' button', 'Open the CD-player'. Actions leading to state change which have no visual evidence (e.g. pressing a button) will need to be inferred (if a finger hovers for more than a second very close to a button, we will assume it has pressed that button) or signalled via some non-visual route (e.g. CD-player directly telling the system what state it is in. Unfortunately, the recognition of the change of internal state will usually require object-specific knowledge (e.g. the 'on' button is at the top left corner of the front panel of the CD-player, when the CD-player is opened, a platter will appear from the front panel). In some cases, this state change will not even be evident visually (e.g. CD-player 'play' button pressed starts the music without a visual change) and will require non-visual cues. An example of a static manipulation action would be :

```
press_button(button, object)
```

Box 3: Example static manipulation action

It is implied that this is done by a hand, the two parameters referring to the specific button pressed, and the object on which the button is found (the location and functions of buttons will need to be specified in the learning phase).

Manipulation of moveable 'pickupable' objects

Generally, the manipulation of moveable objects will involve changes in the objects location, not its internal state (exception e.g. putting tea in a teapot, turning a door handle and pulling the door). Therefore the major actions for the 'put CD into CD-player' scenario would be picking up a CD and putting it down again. An example of moveable manipulation actions would be:

```
pick_up(object, location)
put_down(object, location)
```

Box 4: Example moveable manipulation actions

For both of these, it is implied that this is done by a hand, the two parameters referring to the object being picked up or put down, and the location where this is done (this can either be some recognised area or object, or '`nondef`' which is a task-irrelevant area of the scene, e.g. anywhere on a table).

3.4.3 Construction of the Activity Plan

The Activity Plan Generator (APG) for a specific scenario is constructed during the learning and expert phases, and interrogated during the tutor phase. The learning phase determines the objects relevant to the scenario and their behavioural models. The expert mode extracts linear sequences of atomic plan primitives from each demonstration of the task by the expert, and combines these either incrementally or at the end of the expert phase into a composite representation which reflects common patterns of actions or required sequences of steps for the scenario. Intermediate instructions can also be determined by the expert as this stage, in order to clarify the important steps in the scenario.

3.4.4 Example Native Activity Plan

This shows a possible native activity plan written in the Conceptual Language:

```
#include “playCD.def”

button_press( button0, cdplayer0 )
button_press( button1, cdplayer0 )
pick_up( cd0, nondef )
put_down( cd0, cdplayer0 )
button_press( button1, cdplayer0 )
button_press( button2, cdplayer0 )
```

Box 5: Activity Plan written in the Conceptual Language

This is a record of the concepts encountered by the AD during an expert mode phase using the “playCD.def” definition file. In tutor mode, this can be used to create natural language instruction for the task or to run a VR demonstration.

```
Use Hand to press the Power Button on the CD-player.
Use Hand to press the Eject Button on the CD-player.
Use Hand to pickup the CD from the desk.
Use Hand to put the CD down on the CD-player.
Use Hand to press the Eject Button on the CD-player.
Use Hand to press the Play Button on the CD-player.
```

Box 6: Example natural language output from Activity Planner

4 Activity Definitions

The activity definitions contain the domain independent code for specific activities. These are executed in the Activity Reasoning Engine (ARE) for the creation of hypotheses and concepts. Once a hypothesis is initiated, they control the collection of supporting evidence and the signalling of confirmed concepts that are recorded in the activity plan.

```
// Actipret action pseudo-code definition (domain independent)
// ============================================================
//
// hypothesis for action: pick_up
//
// input: object ref_object - marker for an object that has
//        ‘hand’ properties (allows for multiple hands)
// output: object i - marker for object successfully picked up by ref_object
//                      (null if fails)

object i pick_up(object ref_object)
{
        //preconditions: ref_object must have 'hand' properties:

         ref_object_type =
               visual_index(MAX_BELIEF,object_model, ref_object);
        if (properties_of(ref_object_type) includes ‘!hand’)
               then return null;

        //and must observe 'move_out' gesture:
        if ( visual_index(GESTURE, ref_object) != 'move_out' )
               then return null;

        //and hand must not be holding something:
        if ( visual_index( EMPTY, ref_object ) == false )
               then return null;


        //monitor ref_object position to predict
         // suitable (pickupable) objects:
        X=[];    //start with empty set of candidate objects X

        //---------------------------------------------------
        //loop until hand moves back in or hand becomes full:
        //(needs to be threaded so as not to halt other processes)
        {
                //stop updating if hand so close it occludes object
                if ( visual_index( MIN_DISTANCE, X, ref_object) > 5cm )
                {
                 // identify updated set of candidate objects
                 X=visual_index(FIND_PICKUPABLE_ON_TRAJECTORY,ref_object );
                }
        }
        until ( visual_index( GESTURE, ref_object ) == 'move_in' )
                        or ( visual_index( EMPTY, ref_object ) == false);
        //---------------------------------------------------

        //make sure ref_object is now full:
        if ( visual_index( EMPTY, ref_object ) == true ) then return null;

        //check for a single object no longer where it used to be
        i = visual_index( CONFIRM_LOCATION, X );
                  //(returns set of markers no longer in previous location)

        //check that i not 0 or > 1
        if ( members(i) > 1 ) or ( members(i) < 1 ) then return null;

        // at this stage i = object picked up by ref_object

        //get where i was picked up from ('nondef' if not task-relevant):
        location loc = visual_index( PREVIOUS_LOCATION, i );

        //send message to visual index: i and loc are now task significant
        visual_index( MAKE_PERMANENT, i );

        if (loc != 'nondef') then visual_index( MAKE_PERMANENT, loc );

        //send message to control policy that hand_pickup action perceived
        control_policy( HAND_PICKUP, ref_object, i, loc );
                        //controls conversion into concept, activity plan

        return i;
}
```

Box 7: Example of a domain independent vision based definition of action type

5 Relationship to Cognitive Vision Framework

The Conceptual Language (CL) and Activity Reasoning Engine (ARE) support the principles of Cognitive Vision as initially defined in Deliverable 1.1.

The specific aspects of Cognitive Vision addressed in this deliverable are:

- Control
  - ⇒ Resource management within the collection of supporting evidence and the setting of priorities within the Control Policy.
  - ⇒ In control structure terms, ActIPret is task driven. At the most abstract level, we use rules to guide our expectation associated with the current vision task. These rules are embodied in the form of a control policy. At the different levels of the system appropriate knowledge arising from the control policy will constrain the processing. Our expectations are defined in the context of the chosen scenario (encapsulated in our Conceptual Language) so our control policy can capture this sense of purpose.
- Reasoning
  - ⇒ The Conceptual Language scenario definition files give task-specific cues for the appropriate interpretation of the domain independent activity definitions.
  - ⇒ A CV system must be capable of using the knowledge encapsulated using its representation schema to guide the processing based on expectation (control) and to provide the user with task relevant explanations of derived inferences.
  - ⇒ More specifically, the building of sequences of actions, activities, events and object and behavioural models using the various representation schema within ActIPret is synonymous with the synthesis of an activity plan (“activity planning”) for the scenario as demonstrated by the expert.
- Memory
  - ⇒ The activity definitions provide a framework for the creation and maintenance of hypotheses over time.
  - ⇒ A CV system needs to be capable of representing and storing (cf. memory) data about task relevant objects and behaviours. Such representation schema need to reflect the requirements of the vision task. Traditional approaches to representation tend to focus on hand-crafted schema that have little generality outside of their specific application. Representations for CV applications reflect whether a task is concerned with categorisation (requiring generalised representations capable of supporting decisions such as “some type of car”) or recognition and identification (capable of deciding “this particular instance of car”). Different levels of task abstraction require different types of representation schema and a CV system needs to be able to choose the most appropriate for any particular task. Representations may be organised into hierarchical systems such that specific schema can be derived from more generic
- Learning
  - ⇒ At this stage much of the model information is provide manually in the notional off line learning phase. The learned object models are currently contained in the scenario definition files. An expansion of learning mechanisms will be addressed in task 5.3. This will likely incorporate hierarchical CL action types as described at the end of section 2.