DELIVERABLE D1.1 (v1.2)

Draft of Design Concept of the Cognitive Vision (CV) Framework

Final Version

2 May 2002

Authors: *Kingsley Sage, Jonathan Howell, Hilary Buxton, Markus Vincze, Wolfgang Ponweiser, Christof Eberst, Antonis Argyros, Gerald Umgeher, Jiri Matas*

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

This deliverable document provides the first complete overview of the Cognitive Vision (CV) framework. It summarises the work thus far under Task 1.1 (Conception and Interface Definitions). This CV framework is one of the two main deliverables of ActIPret, the second being the development of purposive processing and interpretation techniques according to this framework.

Section 2 describes the basic principles and requirements for CV. These principles represent a generic CV framework and are not limited in scope to ActIPret only.

Section 3 describes the goals, key design decisions and principles for an effective implementation of the general framework, and presents the basic implementation components and how they are managed and selected. Finally, we summarise how this generic CV Framework is intended to realise the challenges presented by the CV approach. Our goal is that this generic framework can be re-used by the wider cognitive vision community.

The specific application of the generic framework within ActIPret is described in Section 4. First the intended operation of the ActIPret Demonstrator (AD) is outlined. Second, we describe the envisaged components and the component structure of the framework for the AD. Third, a short description of each component is given. And finally, options for tools to implement the framework are presented together with an evaluation leading to the selection of the OSCAR tool.

There are three Appendices to this main document:

- **Appendix A:** detailed function specifications for all the components;
- **Appendix B:** IDL (Interface Description Language) definitions that specify the interfaces of each component; and
- **Appendix C:** a glossary of terms used within ActIPret.

Three other deliverables are presented contemporaneously with this one. These 3 other documents describe in more detail work on specific methods to be used within components of the AD framework:

- **D3.1:** describes the methods of establishing relations between one or more objects;
- **D4.1:** describes the approach to recognition of objects; and
- **D6.1:** describes methods of attentive behaviours.

The interfaces definitions for the components described in D3.1, D4.1 and D6.1 are also in Appendix B. This appendix will be a working document throughout the project.

2 Principles and requirements of Cognitive Vision

The development of a systematic methodology for the design, implementation and integration of Cognitive Vision (CV) systems is currently a very important research problem and has its roots in early proposals for active vision. Cognitive vision is concerned with the purpose and behaviour of computer vision systems in the context of their goal-oriented activity (from ECVision). Essential elements of a CV system are:

- **memory:** representations of objects, categories, actions, behaviours, etc.;
- **learning:** of task relevant representations and how to use them;
- **control:** both selective visual processing and active control of viewing geometry; and
- **reasoning:** about representations and events/activity for system decisions/actions.

2.1 Memory

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 ones.

A CV system also needs a consistent method for representing belief. Different pieces of knowledge may arise from different levels of abstraction of a task, but it is necessary to combine these into a single coherent interpretation of a scenario. For example, in a scenario where we are looking for a particular car in a scene (say “blue estate car with a given registration number”), we may first use a generic level of abstraction to look for a particular type of class (say “an estate car”). This level of task abstraction (categorisation) returns a number of different candidates each with a certain level of belief. We might then use a specific representation to find which member of that candidate set is the specific car we are interested in. This process involves taking a prior level of belief and revising it on the basis of a more specific level of task abstraction (identification).

2.2 Learning

A CV system needs to be capable of acquiring and modifying representation schema such as those described in the previous Section based on experience (training). For ActIPret, this may take the form of off-line learning (supervised or unsupervised) or through adaptation (both through reactive planning implemented in the form of task based control structures and through statistical revision of belief) during run-time. The purpose of learning is to maximise the representation schema to be maximally task relevant. For example, for a categorisation task, if we have only ever observed “blue cars” we might suspect that “blueness” is a generic property of cars. The subsequent existence of a single instance of a “red car” causes us to revise our categorisation procedure and deduce that “blue” is a specific property that can apply to a car (relevant for an identification task) but is not a generic property.

Learning in a vision system can be at the level of:

- object models;
- their movements and actions (behavioural models); and
- how to control views and processing in the system.

Objects can be recognised at different levels of abstraction, general categories such as "face" and "car" or more specific categories such as "estate car" or "familiar face". At the most specific level, there is identification where a unique instance such as "my face" or "my car" is recognised. The learning of behavioural or activity models is concerned with acquisition of structure for movements and actions. Activity models can be associated, in general, with "action verbs" such as "grasping", "picking up" and so on. Activity models can be constructed in terms of actions, activities and events.

In addition to object and behaviour recognition, expected behaviour can be used to control further processing in the system through prediction. Many different learning and prediction techniques have been proposed including, for example, symbolic learning using case based reasoning, graphical models for probabilistic reasoning and control, stochastic models for learning and prediction in tracking, deformable models for event analysis and neural network learning in gesture recognition.

2.3 Control

The ActIPret project focuses on understanding activity in dynamic scenes, which leads us to expand on the theme of control in that:

- perception is guided by expectation, i.e. we "see as" in the famous words of Max Clowes; and
- this expectation is purposeful, i.e. we "see for" a particular decision or action.

In particular, this means that understanding visual behaviour must take place not only in the context of what is known about the dynamic scene but also in the context of the observer's task. The first implies the use of conceptual knowledge, i.e. hand built or learned models in a readily accessible form, and the second implies active control of the visual processing, i.e. selective attention of some kind (overt/covert) for real time processing of dynamic scenes.

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.

In vision terms, ActIPret can be viewed as reactive (bottom up) processing limited in scope by task driven (top down) control and constrained by task driven knowledge.

2.4 Reasoning

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. The scenario is demonstrated to ActIPret as exemplars. These exemplars may have end-to-end significance or the expert may choose to structure them in a hierarchical fashion. Where the expert structures the exemplars in a hierarchical fashion, the result is a hierarchical activity plan with embedded learned sequence models (refer to the paper on learning of models). Exemplars of the plan are represented in the conceptual language.

The conceptual language in general consists of:

- the declarative semantics of the task – information about objects referred to during the description of the task and expected behavioural models for these objects;
- sequences of plan concept functions (atomic plan primitives) organised as a mixture of exemplars with end-to-end significance or organised into a hierarchical structure;
- specific behaviour models for activities, actions or events observed in the exemplars and derived from abstract models defined within the activity reasoning engine (see the paper on learning models for more details); and
- a limited amount of domain specific data, outside the scope of generalised models, contained in the activity reasoning engine, necessary for the correct interpretation of the scenario. Examples of such domain knowledge include prior state variables to represent very specific aspects of the state of functions on a CD player.

3 Cognitive Vision framework

Building on the themes presented in Section 2, we now extend the definitions of the core functions of cognitive vision, that is the belief values, attention control and learning, for use within the context of a general Cognitive Vision (CV) framework. We then develop this general framework into a specific one for an ActIPret demonstrator system in Section 4.

3.1 Belief values

The reasoning engine needs to ensure that service requests to lower components are made in the most effective manner to facilitate detection and recognition of behavioural patterns.

Any system of reasoning needs a mechanism to measure its belief in the knowledge that it holds (whether prior or posterior). This mechanism is the belief system and various underlying models exist to support such belief in a useful and natural manner (for example, probability theory, Dempster Shafer theory and fuzzy logic). The belief system defines a system of belief values that can be used to determine truth (whether binary logic e.g. this predicate is true/false with a given belief, or continuous e.g. this predicate is 70% true / false) or relevance (e.g. this rule is appropriate to apply in this case),

For example, the ActIPret reasoning engine component models the validity of any plan hypothesis generated using a probabilistic belief measure. This measure of belief, combined with information on computational costs and other metrics are used to determine the most effective order and distribution of visual processing tasks throughout the lower level components.

For the ActIPret framework overall, the standard measure of belief is a continuous function in the interval [0,1]. Furthermore, the belief function *b* for any given hypothesis h_i in any of the system components falls into one of 3 generic bands:

- $0 \leq b(h_i) < x_1$: very low (“irrelevant”). This level represents a minimum threshold for the component to maintain the hypothesis (for example, as proposed to the reasoning engine by the object relation generator) and to warrant expenditure of computational resources in attentive processing. Any hypothesis whose belief function falls beneath this threshold for any significant period will be removed (garbage collection). Any processing that was being performed by lower level components in support of that hypothesis would usually be terminated.
- $x_1 \leq b(h_i) < x_2$: worth maintaining and investigating further (“promising”). The initial attainment of a belief value in this level corresponds to the notion of detection. When a hypothesis emerges with a belief function that exceeds the minimum threshold, we make service requests of lower level components in order to recover further evidence that either supports or refutes the hypothesis (increasing or decreasing its belief function). The type of service requests (in terms of computational costs and quality of service) will depend on the current utility of the concept for the behavioural task.
- $x_2 \leq b(h_i)$: very high (“confirmed”). This level of belief corresponds to the notion of recognition. At this point there is sufficient evidence to regard the hypothesis as “true” (an established concept). This band has a specific further semantics in the case of the activity reasoning engine where attainment of the band will:
  - $\Rightarrow$ instantiate a concept function c_i from h_i (i.e. after instantiation, system decision means concept is treated as fact -no longer has semantics of a belief value); and/or
  - $\Rightarrow$ cause a change in the internal state of the reasoning engine (i.e. the concept represents a causal change in control policy such as “revoke rule *X*”, action).

Rules within each component define the belief value thresholds x_1 and x_2 for any given hypothesis. The values of x_1 and x_2 may be independent for any given hypothesis function h_i, and could be adaptive.

Belief values may also be eroded as a function of time. This is a useful method for the garbage collection of tentative hypotheses that were short-term transients but for which it was not possible to gather further and sufficient evidence.

3.2 Attention control

A key element of the ActIPret CV framework is the notional separation of "pre-attentive" and "attentive" vision tasks to allow us to focus intensive processing where it is required. This separation underlines the importance of control and reasoning in selecting the focus of attention. We have extended the concept of a binary separation of pre-attentive and attentive vision into a continuum between the two based on measures of computational cost and quality of service (pre-attentive vision corresponds to low computational cost and potentially low quality of service, and attentive vision to high computational cost and quality of service).

Attentive processing is then a continuum of visual services. These services are defined in terms of their computational cost and quality of service. The highest level of control is a function of the activity reasoning engine. This reasoning engine makes task relevant decisions about which visual services should be requested. Using notation from Section 3.1, attentive control for any component arises from the middle band of belief values ($x_1 \leq b(h_i) < x_2$) for any given hypothesis h_i. The control mechanism is a functional loop within a component that attempts to continuously revise the belief in a given hypothesis until it either falls below a threshold x_1 and is discarded, or rises above a threshold x_2.

Other attention control issues that need to be addressed include:

- representation of the rule set within the control policy;
- some hypotheses are not intended to become concept functions, simply to influence selection within local control policy (maybe changing camera viewpoint or tutoring);
- send service requests on a per cycle basis or less frequently, depending on task relevance, computational cost and quality of service parameters; and
- how the values of x_1 and x_2 are set for each member of the hypothesis set.

Services that require particular views (and therefore can influence the motion of the robots) may conflict for resources or may create situations that 'demand' unsafe motion for the robots. Therefore, the service commanding/coordination/control structures must be performed with an intrinsic knowledge of views, kinematics/capabilities of the robots. This requires a reactive co-ordination that activates/de-activates these modules according to the demands of the service requests and this intrinsic knowledge. This coordination, combined with the view-request of the service-modules that specifies views according to their current processing, form the base of the attention control. The services/modules, developed in WP 1-5 are co-ordinated according to their requirements (if any) of quality of service and costs, and the available system resources (mainly robots).

Therefore, highly reactive co-ordination modules interact with these modules at each processing-cycle to 'direct' the camera/robot entities (evaluating, starting and stopping the individual services). In order to improve the overall performance, of the system, the local coordination modules communicate with a central "contract manager" module, which performs task-assignment. For details see Deliverable 6.1.

3.3 Learning of behavioural models

For ActIPret, there are two types of relevant learning that take place in the learning phase:

- learning of general activity models (described by “action verbs”) such as “open the CD player”; and
- learning of general object models for object detection and recognition such as “CD” and “CD player”.

There are several different types of behavioural activity model:

- **Abstract models** (parametric, but without specific values): such as "pushing a button". Abstract models are defined by vision based activity descriptions. For the “pushing a button example” the abstract model might be defined, for example, by requiring minimum distance and persistence parameters.
- **Specific models** (parametric and with instantiated values): derived from the abstract ones. These specific models (or at least their parameters) would be embedded within an instantiated ‘activity plan’. For example, the model for finger pushing the play button on a CD player might be derived of the previous “pushing a button” abstract model by setting a minimum distance of 5cm and a persistence time of 5 seconds.
- **Learned sequences** (part/whole of activity plan): specific activity primitives combined into higher-level (more abstract) activity descriptions. For example, the learned sequence model of "opening the CD player" would consist of alternative sequences of activity primitives based on exemplars.
- **Hand-coded models:** when there is no intention to learn given aspects of the model. The use of Hand-coded models should be a last resort but may be important where we want to capture highly specific activities or events that cannot be determined readily by vision alone (for example, recognising that a music is coming out of a CD player). Hand-coded models can receive external non-vision triggers to determine their state.

3.4 Software framework for Cognitive Vision

One of the goals of ActIPret is to develop a general purpose framework for CV, that can be used not only for the ActIPret demonstrator but also in other CV applications, other vision systems or even vision and robotics systems. In this Section we describe:

- related work relevant for framework design
- the goals for such a software framework;
- the design decisions to realise the framework;
- two particular principles the framework builds on (the ‘service principle’ and the ‘hierarchy principle’);

- the structure of the individual components; and
- the mechanism to manage resources and to select components.

3.4.1 Related work

Throughout the course of computer vision as a research topic, much effort has been applied to the transfer of ideas and results into practical vision systems [1]. Since 1999 the dedicated Conference on Computer Vision Systems (ICVS) has provided a valuable platform for this effort. But still there is a big gap between theoretical algorithms and commercially available vision component products. In the sense of component-based system programming (as seen from a programming perspective [2]) CV systems can also be built around many ‘basic’ vision components like object recognition or active tracking. Hence developments in building general component-based vision systems are equally relevant for the software framework of a CV system.

Besides a large number of application specific implementations of frameworks and architectures, there are generic frameworks designed to distribute and schedule ‘small’ subtasks like binary and basic pixel operations [3,4]. The fixed processing time of such basic operations and the pre-defined pipes and filter dataflow means that a static scheduling can be calculated in advance. Here the main goals are the efficient distribution on different HARDWARE like DSPs or FPGAs and the assurance of real-time constraints.

From a perspective of frameworks and architectural guidelines for vision components of the typical vision component size (more closely related to component-based system programming), the field of autonomous robotic is of relevance. Some of the robotic systems using vision components are described in [5-11]. All of them provide mechanisms to configure a system set-up during compile or start-up time. But no real dynamic component selection and activation is permitted.

Probably the most relevant work is the OROCOS project (*www.orocos.org*) that aims to develop a common basic framework for developing robot control software. OROCOS designs and implements (among many more) some of the principles for robotic control, which are also of relevance to perceptual requirements (e.g. coordination, reactive behaviour [12-13]). For example, we reused some of the basic communication patterns from OROCOS for the IDL-description. However, OROCOS only started in September 2001 and it is not expected that implementation results can be exploited before end of 2002. Related tools to develop frameworks are discussed and evaluated in Section 4.4.

[1] W. Förstner: “10 pros and cons against performance characterization of vision algorithms”; Workshop on "Performance characteristics of vision algorithms", Cambridge, 1996.

[2] C. Szyperski: “Component software”; Addison Wesley; United Kingdom 1999

[3] F. Torres, et al; “Simulation and scheduling of real-time computer vision algorithms”; ICVS 99.

[4] D. Benitez, J. Cabrera: “Reactive computer vision system with reconfigurable architecture”; ICVS 99.

[5] J.S. Albus; 4-D/RCS: “A reference model architecture for Demo III”; IEEE ISIC/CIRA/ISAS Joint Conf.; Sept 1998.

[6] M. Anderson, A. Orebäck, M. Lindstöm, H. I. Christensen: “ISR: An intelligent service robot”; in ``Intelligent Sensor Based Robotics'', Eds: Christensen, Bunke, and Noltemeier, Lecture notes in Artificial Intelligence, Springer Verlag, 1999.

[7] Stefan Blum: „OSCAR - Eine systemarchitektur für den autonomen, mobilen roboter MARVIN“; In Autonome mobile systeme, informatik aktuell, pages 218-230. Springer-Verlag, November 2000.

[8] R. Alami, R. Chatila, S. Fleury, M. Ghallab, F. Ingrand: “An architecture for autonomy”; IJRR Vol. 17, pp 315-337, April 1998.

[9] J. Cabrera-Gamez, A.C. Dominguez-Brito, D. Hernandez-Sosa; “CoolBOT: A component-orientated programming framework for robotics”; Dagstuhl seminar 00421, Springer-Verlag Lecture notes in computer science, 2001.

[10] P. Steinhaus, M. Ehrenmann, R. Dillmann; “MEPHISTO A nodular and extensible path planning system using observation”; ICVS 99.

[11] O. Stasse, Y. Kuniyoshi: ``PredN: “Achieving efficiency and code re-usability in a programming system for complex robotic applications”; ICRA 2000.

[12] R.C. Arkin: “Behaviour based robotics”; The MIT Press 1998.

[13] R. A. Brooks: “A robust layered control system for a mobile robot”; IEEE Journal of robotics and automation, Vol. 2, No. 1, pp.14–23, March 1986.

3.4.2 Goals

This section summarises the important design goals. Besides the needs and principles of cognitive vision (see Section 2), there are several goals that refer to general principles of a framework:

- **Task driven (pro-active)**: as described in Section 2.3 the ActIPret system has to deal with limited resources. Hence the system has to focus its resources (processing power, views etc.) according to task relevancy.
- **Data driven (re-active)**: the system has to react to the sensor input provided by the cameras. The goal is to build a CV system that reacts and responds to human actions. Hence reactions must take place in a time scale appropriate to the natural speed of action of humans
- **Scalability**: should be linear or as close as possible to linear. In practice this means that duplication of any component should result in (as near as possible) duplication of the associated service assuming the existence of similarly duplicated resources.
- **Control**: scalability also implies distributed control throughout the system (a single control component would not scale). Hence, each component has its own "control policy" (albeit at different scales of task relevance) to control the services requested and the results or data obtained.
- **Modularity**: the development effort for CV software can be limited by reusing software segments. Modularity forms the basis for software reuse and a dynamic system structure.
- **Independence of components**: whichever component requests a service should receive the response. Hence every component has the control of the services it provides to other components and the services it requests from other components.

3.4.3 Design decisions

- **The system is distributed*:*** vision and interpretation require huge computations. In order to achieve scalability and parallel development the components should be distributed. Consequently resource management should be distributed too.

- **The components of the framework run asynchronously*:*** the effort required to synchronise a distributed system is too big related to the simplification reached in the integration process. In particular, synchronisation is pointless for components having different temporal behaviours such as tracking and recognition.
- **Data consistency*:*** control over service responses from other components requires assuring that the data received is made consistent. For example, if two detectors on different cameras are asked to find a hand, the responses must be fused in the component that requested these two services.
- **Number of resources:** the number of resource types in a generic system can be very high. However for CV systems this number is of the order of 3. For example, in the ActIPret demonstrator there is the CPU time and the realisable view together with the image delivery. Hence, for CV systems the number of resources a component can request is small.

3.4.4 Design principles

The framework is built according to two major design principles derived from the list of goals and based on the design decisions.

The 'service principle'

Our first goal for the CV framework is to build a task driven system and so every distributed component is task driven. From the point of view of a component, it has to provide one or more interfaces to make its functionality accessible. Using this interface other components can initiate one or more tasks. Components can be regarded as black boxes and so according to the goal of component independence these interfaces have to describe all the abilities of the component. These interfaces are called ***services*** and the component acts as the ***service provider***.

Any component can use the services of other components. In this case the component acts as the ***service requester***. So every component 'presents' its abilities by providing services and every component makes use of the abilities of other components by requesting services. These services form the complete interface for a component and hide the implementation.

A special component, called the ***service list,*** provides the mechanism for any components to find out about the services that other components provide. This mechanism operates as follows: Every component registers its services in the service list. Using the service list as a 'yellow pages' directory for services, a component can find on-line the services it requires and can then establish communications with the relevant component(s).

In order to provide a simple service selection the ***service list entry*** consists of the service name, some service specific properties (describing the service in more detail) and an ***abstract description*** of the abilities of the component. This abstract description contains two main descriptors:

- **Quality of Service (QoS*)***: describes performance (confidence, accuracy, speed etc.) a service can actually (when registering/updating the entry in the service list) deliver; and
- **Costs**: abstract descriptions of the effort required to establish a service. This is used to quantise the narrowness of a resource and the conflicts (incompatibility) of resource requests. For example, the effort to transfer a running component from one PC to another PC (i.e. to get more CPU time on that PC) is embodied by migration costs.

Several other descriptors are possible, for example, constraints related to a service. However these two descriptors (QoS and cost) are the minimum and will be used and implemented for all components.

To cope with limited resources (see Section 3.4.6 on resource management below) it is necessary to introduce a **priority** descriptor to quantify the need of the component requesting the service: This is set by the component requesting the service to inform resolution of conflicts between service requests.

These three **abstract descriptors** (QoS, costs, priority) provide a mechanism to support a distributed system and a dynamic system set-up. They support the overall goals of scalability and modularity and should provide a reliable system that is able to react online to the failure of a component or to the introduction of a new component.

The 'hierarchy principle'

Fully agent-based systems do not use a hierarchy. All agents negotiate until a configuration is reached. This is thought too time consuming for a reactive system. To reduce administrative communication overheads during decision-making (service selection) a strict hierarchical structure of components is used. Hence every link between components has a higher-level component and a lower-level component. The higher-level component always orders the task (service) the lower-level component has to process (provide/deliver).

3.4.5 Component structure

Using these two principles a component can be separated into three parts. In the main part of the component the real functionality is placed. This part is encapsulated by two interfaces. The top interface provides the services of the component itself and the bottom interface requests services from other components. The example in Figure 1 shows an object recognition component that provides three specific object recognition services: identification, detection and search of/for objects. The ***service provider interface*** configures the component for the requested task and is responsible for the communication with the requesting component. The ***service requester interface*** is responsible for the selection, configuration and management of services requested by the main part of the component. All communications to lower level components are realised through this interface.

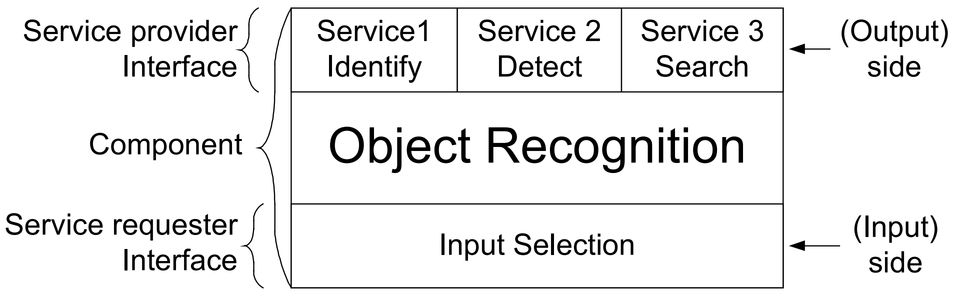

Figure 1: Component structure.

The framework makes use of two types of component:

- ***Main Components (MC):*** are the workhorses of the system. All necessary visual computation and task control (excluding the effects of resource constraints) decisions of the system are placed inside MCs. Typical MCs for a CV system include: object recogniser, feature tracker and reasoning engines. MCs register their services in the service list and are able to select services of other components.

- ***Controller Components (CC):*** represent resources inside the system. It is not important if the real actuator driver is part of the CC or if the CC is just a gateway to the real driver. Typical CCs for a CV system include are view controller (for establishing camera poses, change focus, cutter etc.) or CPU controller.

3.4.6 Resource management and service selection

Use of the ‘service principle’ presents a major challenge in that vision systems have to deal with restricted resources. There is always a limited amount of CPU power, only a limited number of cameras and the free working space of the robots used to get good viewpoints is constrained. Therefore access to resources must be managed to solve conflicting resource needs.

In order to optimise resource allocation, it is necessary to manage the components that raise the resource requests. Otherwise components might be activated that have little or no probability of obtaining the necessary resources. For example, to start a component on a PC with nearly zero available CPU time makes little sense. Hence every component (or, more precisely, its abstract representation the service) has to be managed, because of the restricted and conflicting resources it requires.

Using the component example in Figure 1 the method of selecting a service is the critical issue. Service selection serves the purpose of selecting resources efficiently. Therefore it has the goal of implementing and resolving resource management. The mechanism introduced here is distributed and based on the CORBA trader service (or a derived method), defined by the OMG (Object Management Group *www.omg.org*). Figure 2 gives an example sequence for the set-up of a component connection.

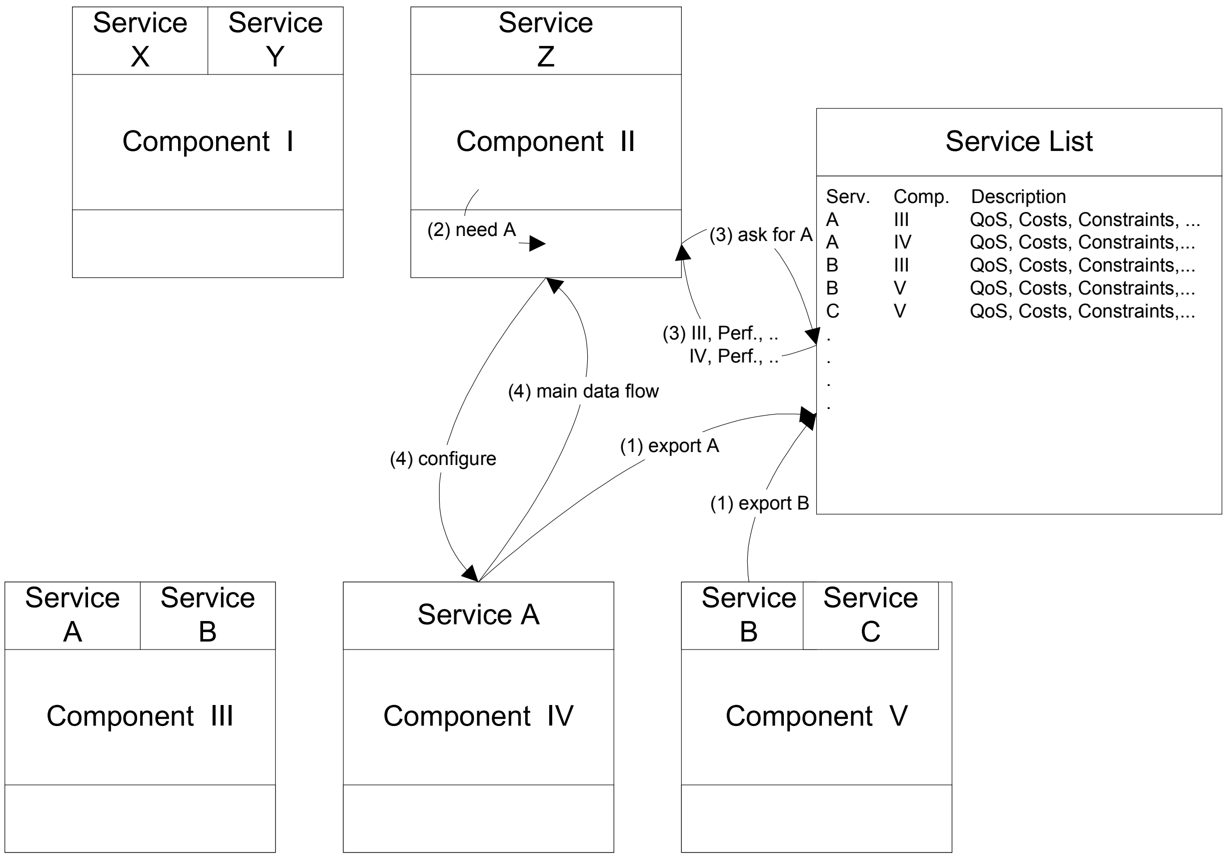

Figure 2: Selection and configuration of a service. Numbers (i) correspond to the item numbers in the text. Not all messages/arrows are shown.

The following items describes the sequence for registering a service in the service list and selecting the service(s) requested (item numbers correspond to numbers in Figure 2):

1. During the component start-up, every component registers its services in a global service list. In CORBA terms this is the export of a service offer (this start-up does not have to be at the same time as the system start-up!) A service list entry consists of the service name, the ID of the component providing the service and an abstract description of the service (i.e. quality of service QoS, costs, constraints, etc.). This entry is updated only if major changes of the component description occur.
2. If a component needs a service, it asks its service requester interface to establish a connection. From this point in time the main component part, where the real functionality is placed, is decoupled from the mechanism to set-up the connection. All selection, configuration and communication is done by the service request interface (see Figure 1).
3. The service requester interface starts the search for a service in the service list (it asks for service "A" in Figure 2). In CORBA terms this is known as an import. The component description (QoS, costs, etc.) makes it possible to select the relevant subset of all components providing the service "A". The service list returns the list of corresponding service offers (in our example, for components III and IV).
4. The service requester interface selects one of the service offers received and initialises an OSCAR link to the selected component (in fact, to its service provider interface). This completes the process to find the service requested.

The component that delivers the service is responsible for generating and updating these values in the service list. Especially, QoS and costs depend on available resources. The necessary information on resources required to calculate QoS and costs is received from the relevant controller components (CC). This process is referred to as resource management.

3.4.7 Management of models

This Section applies to task based behavioural models at the highest level of abstraction such as those described in Section 3.3, but also to the use of models more generally (e.g. object models) in computations throughout all of the ActIPret components.

Different components will need access to different model types in order to perform their characteristic operations. Each component only supports a limited number of model types. A *Support Vector Machine* (SVM) based recogniser will need a SVM model; a *Principal Components Analysis* (PCA) based recogniser will need a PCA model; a *Hidden Markov Model* (HMM) based behaviour analyser; and the ACIN detect and tracking component requires a wire frame model. The component selection mechanism of the service principle has to ensure that components selected for a task can access models of the object of a supported model type.

Consider the situation for object recognition: The component selection is model type specific e.g. PCA model for the PCA recogniser. The model type is itself object specific e.g. if we have a PCA model of a CD player. However, the object is not known before the real service request is generated. Hence from the software framework point of view, model data behaves like a varying resource.

One brute force method to solve this problem would be to ask every component if it can detect (recognise, track, etc.) the object. Most would answer that they cannot detect the object because of missing model data of the type required.

Hence we suggest implementing the following method for managing model data (i.e managing the model data constraint for selecting components):

1. There are one or more **model server(s)**. Model server(s) contain either the model data themselves (especially for common model types) or have reference entries to model data that is component specific and hence placed in the component itself (e.g. SVM model data is placed in the SVM recogniser component).
2. Every model data dependent component adds the supported model data types to the service description in the **service list**.
3. At the first step of component selection, the service requester evaluates the available model data types for the object to be computed. It receives this information from the model server.
4. These model data types are then forwarded to the service list as part of the service request.
5. At the service list this data is compared to the entries and appropriate selections can be made.

This method is asymmetric compared to other selection constraints like the view or the CPU time, because the service requester has to take account of requirements from the service provider. But in principle the view can be handled by the same method. The service requester could ask all available cameras for the potential to realise a specific view and then forward this information to the service list, where every entry contains the cameras they can use. The main differences between the model data and the view are:

- the number of cameras in the system is probably higher than model server(s) (just one for the initial demonstrator). Hence the communication overhead is higher; and
- the view that should be realised depends on the implemented method of the selected component.

This method meets the goals for the software framework for CV. The restricted resources are still divided according to the actual tasks (according to task relevancy) in an adequate timeframe (not to lose important data - data driven). The method scales linearly (although the number of model servers is assumed to be small) (scalability) and supports dynamic and modular composition (modularity) of independent components (independence of components).

3.5 Realisation of principles and requirements of Cognitive Vision

This Section is intended to show how the generic framework solution presented realises the principles and requirement for a CV system (goal-orientation and control, reasoning, memory, learning) described in Sections 2. The AD framework proposed in Section 3.4 is a tool to implement these principles and requirements.

The **goal-orientated** nature of CV is realised through the use of components that can dynamically select the components that are best suited to solve the immediate task. Whilst it seems that the major **task-based control** resides in upper level synthesis and reasoning, each component also provides task-based control for lower level processes. This control extends from selection of tracking or recognition right down to the selection of the most relevant views, cues or features.

Task-directed control in each component is most powerful if each component also contains capabilities to **reason** about its local tasks and can therefore select appropriate processes. The belief values form the basis of the method to provide formal values for the reasoning process. This formalism enables handling of the results of external processing (from a service that has been requested) and of internal processing within the component itself (although we are aware that the nature of the belief values might differ substantially depending on the underlying methods used to obtain the belief values). The service principle realises distributed reasoning, such that knowledge available in each component can be best exploited in the current context.

It should now be clear that **memory** (in particular what could be called short-term memory) also resides in each component, since it's related to control and reasoning. Short-term memory denotes time varying data, e.g., the results of previous processing and reasoning steps, updates of states and parameters. To ensure that data coming from internal and external processes is consistent, it has to reside within the component. On the other hand, there are models in long-term memory (that can have different forms of representations), which persist over time, such as a model of an object or an activity. It is the ModelServer that ensures consistency of these models.

The final aspect of CV considered here is **learning**. As will be seen later in Section 4, a separate "learning phase" is devoted to initialising the system components. Initially learning will take place at a component per component basis, e.g. learning to detect hand motions, learning of belief networks for activity interpretation or learning representations for recognition. This learning can be done fully off-line (the component alone) as well as with several components providing the input for learning. For example, hand tracking providing the input for learning to discriminate hand gestures. It is finally foreseen that during the phase of interpreting new activities, the system could also learn, though in this case we should refer instead to 'adaptation' to a specific user.

Table 1 provides a summary of the essential items for realising the principles and requirements of CV:

Principle	**Realisation**
Memory	Long term memory: ModelServer (consistency of time invariant data) Short term memory: distributed to all components (consistency of actual data)
Learning	Distributed to all components: • Initial learning phase (off-line, before system operates on its task) to learn activity as well as object models • On-line learning phase (limited learning capabilities) Extension: learning between components
Control	Task-driven to focus processing and exploit resources efficiently Data-driven to obtain reactive system Reason about control using memory
Reasoning	Formalised services (QoS, costs) report belief values for reasoning Distributed to use local knowledge and to scale with complexity

Table 1: Realisation of principles and requirements of Cognitive Vision.

4 ActIPret demonstrator

The objective of ActIPret is to develop a vision methodology that interprets and records the activities of people handling tools. The tasks considered are observable by video streams. Focus is on active observation and interpretation of activities, on parsing the sequences into constituent behaviour elements, and on extracting the essential activities and their functional dependence. By providing this functionality ActIPret will enable observation of experts executing intricate tasks such as repairing machines and maintaining plants.

The expert activities are interpreted and stored using natural language orientated expressions (encoded in a conceptual language) in an activity plan. The activity plan is an indexed manual in the form of 3-D reconstructed scenes, which can be replayed at any time and location to many users using, e.g., Augmented Reality equipment. Due to the interpretive level of the system, ActIPret can provide the trainee with feedback when repeating the operation (in simulation or reality), which results in a superior training effect compared to repetition without feedback.

A demonstrator will be built to test the generic framework on this scenario. This specific example of a framework implementation is referred to as the ActIPret demonstrator (AD).

4.1 How the user interacts with the ActIPret demonstrator

4.1.1 AD functional phases

There are 3 substantive functional phases for the AD:

- **Learning phase:** in this phase the AD relies on a teacher (as distinct from the expert) to provide supervised learning examples of activity and object models. For activity models, the system needs to derive specific models relevant to the scenario that the expert will demonstrate from the generic models built into the reasoning engine. For object models, the lower level vision components need to establish parameters for detection and recognition tasks based on supervised learning examples.
- **Expert phase:** in this phase the AD is shown the sequencing of plan primitives that make up scenario exemplars. This process is synonymous with activity plan synthesis. The learning phase must have already been carried out before we can use the expert phase. Expert phase learning may be incremental (take place in several sessions giving rise to multiple scenario exemplars) although we assume that for any scenario that the resulting activity plan contains no exemplar sequences prior to entering the expert mode for the first time. There are 2 elements to the expert phase:
  - ⇒ extraction of the linear exemplars; and
  - ⇒ (as a further refinement) organisation and hierarchical grouping of parts of the linear exemplars to highlight common elements of plan structure. This could be as simple as combining all of the linear exemplars or could include more sophisticated plan editing.

  Once all expert phase learning is complete, the activity plan is fully synthesised (represented in the conceptual language) and is stored in a static database.
- **Tutor phase:** in this phase the naïve user (the trainee) attempts to perform the sequencing of plan primitives. This phase is synonymous with both activity plan synthesis and comparison. The expert phase must have already been carried out before we can use the tutor phase. As a result, we have the activity plan as generated during the expert mode stored in a static database. This is the reference activity plan. There are then 2 elements to the tutor phase:

⇒ explanation of the content of the static database to the trainee. This can be achieved either through natural language based descriptions or through VR reconstruction; and

⇒ plan synthesis that takes place to a separate transient workspace plan that is then stepwise compared with the exemplars in the static database. Variations between the temporary workspace plan and the static database represent errors in the trainee's performance.

The need for VR reconstruction within the tutor mode further creates a need for a more "complete" conceptual language.

These 3 phases are visually summarised in Figure 3.

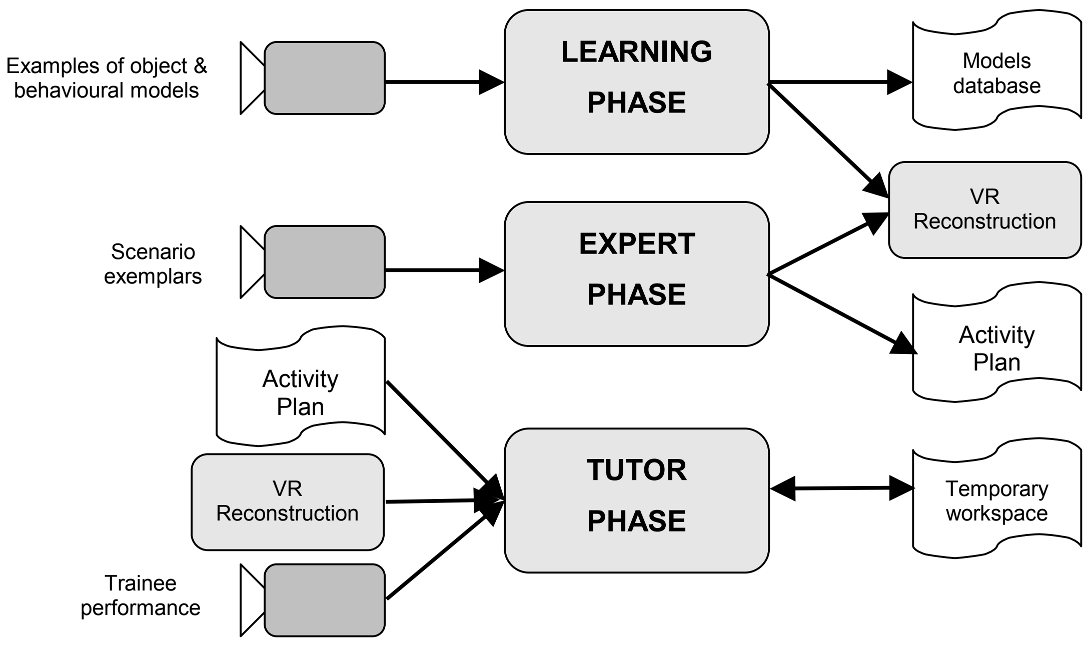

Figure 3: This diagram shows each of the 3 functional phases with their corresponding high level inputs and outputs

4.1.2 AD conceptual language

The activity plan is written in a conceptual language. A very simple activity plan might be represented as follows (using C++ derived notation – the conceptual language is not yet developed fully):

```
// ---------------------------------------------------------------------
// Using C++ notation (refer also to the paper on learning models …)

linear_exemplar(1) {
       button_press(button0,cdplayer0);
       button_press(button1,cdplayer0);
       pick_up(cd0,nondef);
       put_down(cd0,cdplayer0);
       button_press1(button1,cdplayer0);
       button_press2(button2,cdplayer0);
       }

button_press(button1, cdplayer0) : public button_press(b,object) {
       // button_press(b,object) would be defined within the reasoning engine
       minimum_distance = 10.0;
       persistence_time = 5;
```

```
        }
// -------------------------------------------------------------------
```

4.1.3 Our approach to building the first AD

The initial thrust of the development work of the project will be in the iterative development of the expert phase. Development starts with very simple activities beginning from the basic set of objects for the CD scenario and the action verbs (grasp CD, move hand, CD-player,…) as suggested by COGS.

It is our goal that objects are shown and learned before being used in activity interpretation. It is assumed that the intention of the expert is known. Hence objects and basic activities involved are known at the beginning.

4.2 AD System Structure

Figure 4 shows the overall structure of the ActIPret Demonstrator (AD). Three different types of process can be detected. The relationships of these processes mirror the hierarchical principle introduced in the previous section. The higher the data abstraction of the process output, the higher the level of the corresponding component in the AD diagram.

The components are arranged in three main layers, *Synthesis, Pre-Reasoning* and *Pre-Attention and Attention*. Each layer corresponds to a different level of data abstraction. Pre-Attention & Attention groups all functions that produce as output object poses. Pre-Reasoning computes temporal and spatial relations for one or two objects. Synthesis finally oversees the complete scene and produces the activity plan, the final abstract description of the activity. The components alone as well as together implement the need of a CV system for belief values, attention control and learning.

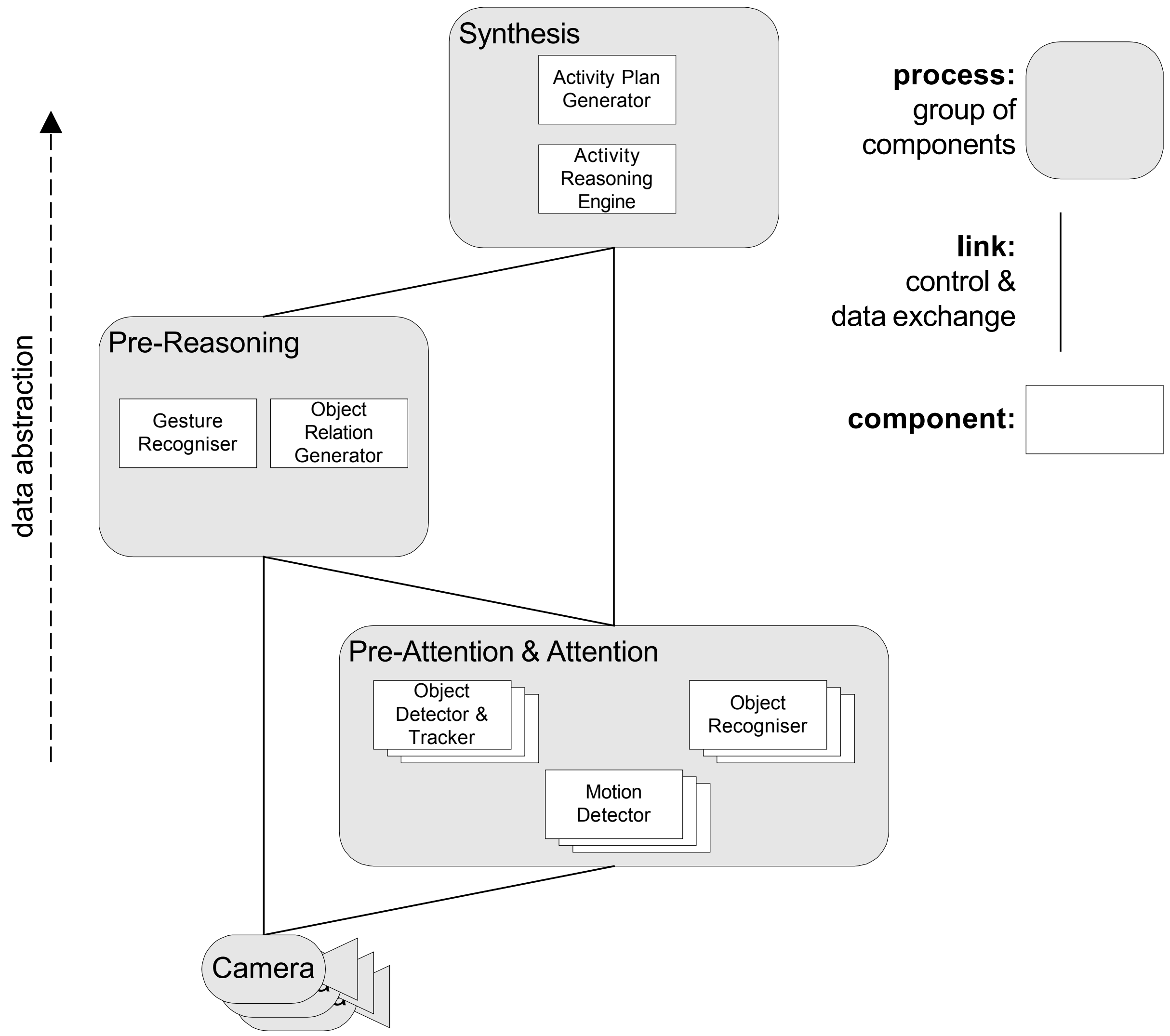
Synthesis
Activity Plan Generator
Activity Reasoning Engine
process: group of components
link: control & data exchange
component:
data abstraction
Pre-Reasoning
Gesture Recogniser
Object Relation Generator
Pre-Attention & Attention
Object Detector & Tracker
Object Recogniser
Motion Detector
Camera

Figure 4: Abstract view of the ActIPret Demonstrator system structure

4.2.1 Example set-up for expert mode

An example system set-up of the ActIPret demonstrator during the expert mode could look like Figure 5. All real components are assigned to an abstract process.

- Main Components (MC) can be further subdivided into:
  - ⇒ Servers that only provide services;
  - ⇒ User(s) that only require(s) services;
  - ⇒ Components that provide and require services; and
  - ⇒ Components that provide and require services and request a view (marked with a chequered square in Figure 4).
- Controller Components (CC) that try to service as many resource requests as possible; and
- Service Lists where all available services are registered.

As already mentioned in Section 3.4.6, service selection of components with varying resource requests can result in significant communication overheads. This is the case particularly with the ‘realisable view’ and therefore the corresponding robot pose is extremely variable and its behaviour is particularly environment dependent. If a component requests a view directed to track the top of an object (e.g. the CD) every change of position of this object can cause huge changes in the position of the robot. To reduce the communication overhead when using realisable view, the service list is separated into two different parts:

- the view independent ***service list*** where all components are registered that don’t require the resource ‘realisable view’; and
- the ***service list with view controller*** where all components are registered that require the resource ‘realisable view’.

All view dependent components are placed in one specific layer as below. This simplifies the service selection and robot assignment process for this type of service request.

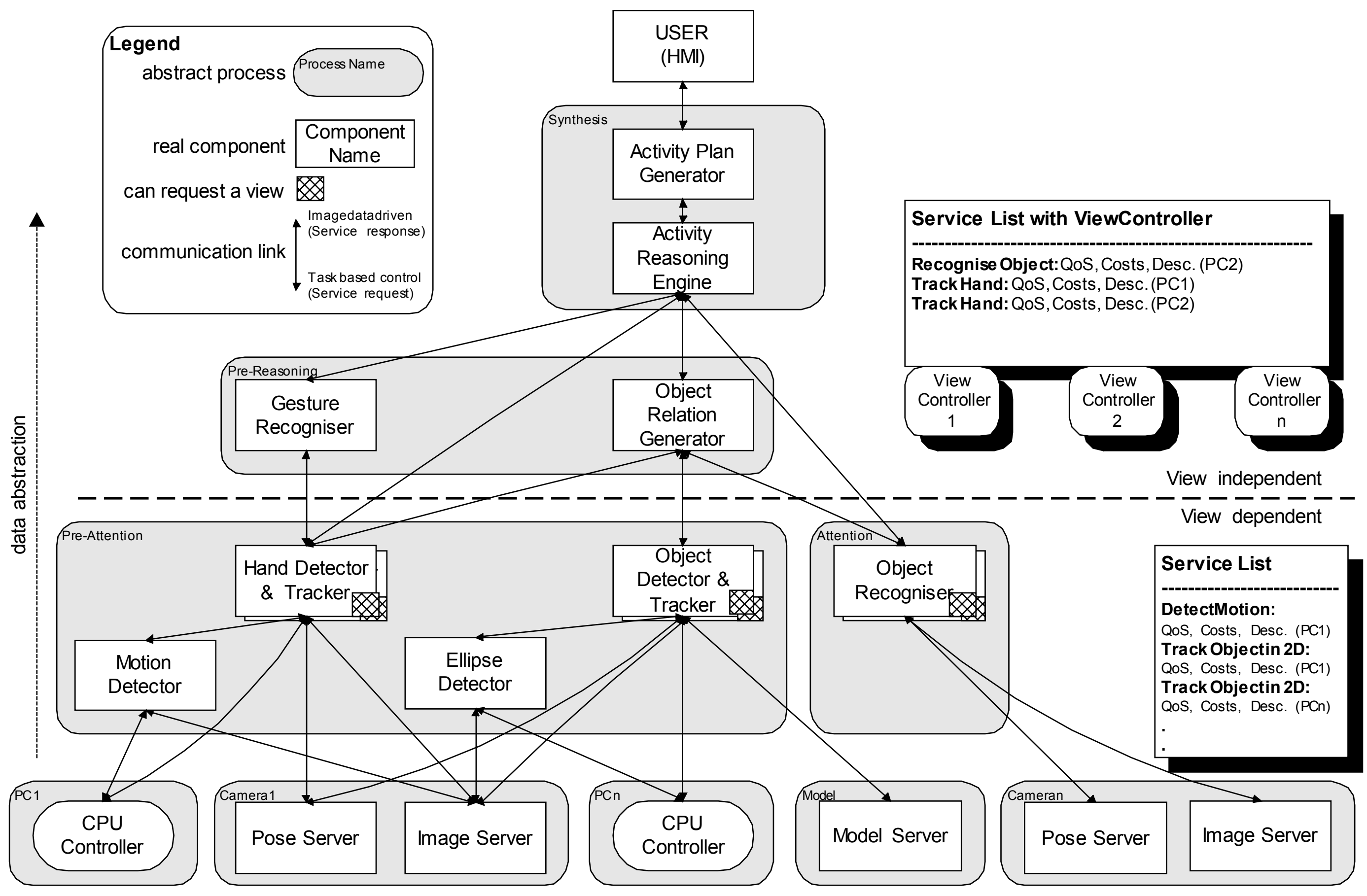

Figure 5: Example set-up

The example set-up for the ActIPret demonstrator in Figure 5 realises all the principles and implementation rules introduced in the previous sections. Since the service principle is related to the dynamic set-up of the system, it is invisible. Nevertheless, the two different service lists can be seen on the right side.

4.3 Description of components

This Section contains a short overview of each functional component of the AD. A fuller account of each of these components can be found in Appendix A with IDL in Appendix B.

4.3.1 Service list (ACIN)

The Service List is a database where components register their abilities and the properties of these abilities. These abilities are called ***services*** and form an abstract interface description of the components. All components and their services depend on the resources available.

Resources are restricted. There is always a limited amount of CPU power, only a limited number of cameras and the free working space of the robots used to get good viewpoints is limited. Therefore access to resources must be managed to solve conflicting resource needs.

The following three values are used to parameterise an abstract service description:

- **Quality of Service (QoS):** describes the performance (confidence, accuracy, speed, etc.) a service can actually deliver.
- **Costs:** are the abstract descriptions of the effort (resource requirement) of the establishment of a service.
- **Priority:** is set by the resource-requesting component to give a hint on how to solve conflicts of resource requests.

4.3.2 User HMI (COGS)

This module has overall control of the ActIPret system, and switches it between the major phases of operation:

- **Learning phase**: individual models are trained on databases of generic information.
- **Expert phase**: the entire system is run in a 'record-only' mode analysing the activities of an expert while demonstrating some specific part/whole of scenario and providing explicit start and end signals. Activity plans from each demonstration of the scenario are combined into a composite representation of the scenario.
- **Tutor phase**: the system either 'explains' how a scenario is carried out to a learner via some VR representation or 'watches' the learner attempt to carry out the scenario, either giving a binary 'correct/false' interpretation, or attempting to give guidance to the user when errors or confusion occur.

4.3.3 Activity planner (COGS)

In expert mode, the Activity Planner is concerned with the synthesis of an activity plan in a static database for the scenario as demonstrated by the expert. The scenario is demonstrated to ActIPret as exemplars. These exemplars may have end-to-end significance or the expert may choose to structure them in a hierarchical fashion. Where the expert structures the exemplars in a hierarchical fashion, the result is a hierarchical activity plan with embedded learned sequence models. Exemplars of the plan are represented in the conceptual language. This conceptual language will be defined formally elsewhere.

The conceptual language in general consists of:

- **Sequences** of plan concept functions (atomic plan primitives) organised as a mixture of exemplars with end-to-end significance or organised into a hierarchical structure;

- **Specific behaviour models** for activities, actions or events observed in the exemplars and derived from abstract models defined in the activity reasoning engine;
- **Limited domain specific data** outside the scope of existing models, necessary for the correct interpretation of the scenario. Such domain knowledge may include state variables to represent specific aspects of the functions on a CD player.

In tutor mode, VR reconstruction of the scenario using the activity plan will provide initial instruction for the trainee. The activity planner is then used to generate a new temporary workspace activity plan for stepwise comparison with the static activity plan produced in expert mode. Comparison of the static and temporary plans will be used to determine whether the trainee has made any errors. VR reconstruction will then be used to explain any discrepancies to the trainee.

4.3.4 Activity reasoning engine (COGS)

The Activity Reasoning Engine can be represented at a high level as shown in Figure 6.

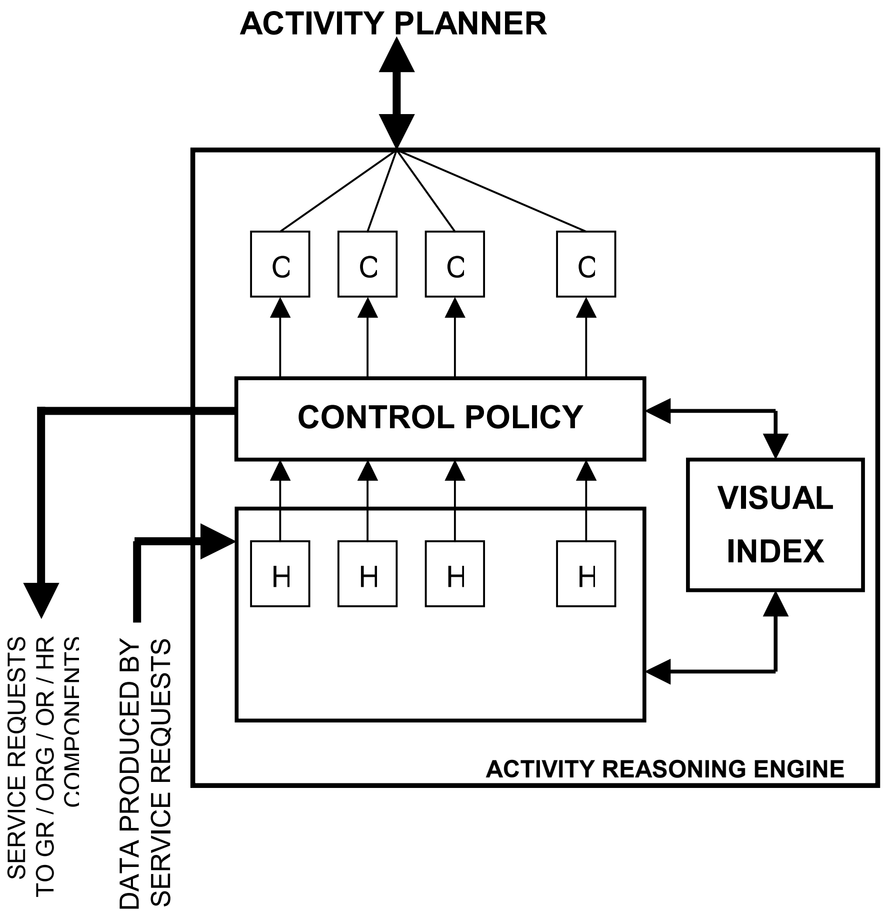
ACTIVITY PLANNER
C
C
C
C
CONTROL POLICY
VISUAL INDEX
H
H
H
H
ACTIVITY REASONING ENGINE
SERVICE REQUESTS TO GR / ORG / OR / HR COMPONENTS
DATA PRODUCED BY SERVICE REQUESTS

Figure 6: High level view of activity reasoning engine.

The activity reasoning engine contains four separate elements:

- **Control policy:** determines at the most abstract level the task control strategy for the whole ActIPret system. When ActIPret is first started, this policy will be generic e.g. "find possible hand objects". This policy will evolve as task relevant vision data is produced. For example, if a hypothesis has emerged that a candidate hand object is moving with a consistent trajectory, then the control policy should then locate candidate objects along that trajectory that the hand might pick up.

- **Visual index:** is a database of task relevant objects. The definition of task relevance will depend on the context set by the control policy.
- **Hypothesis functions *H*:** provide posterior probability estimates for a set of plan hypotheses. For example, using the previous example, the probability h_i is given by h_i = grasped-by(<object1>, <object2>). There are two distinct types of plan hypothesis function: those that map object tuples to deictic relationships that may lead to instantiated plan concept functions and those that map individual objects that are required for informing the control policy.
- **Hypothesis manager:** see Appendices for more details.
- **Concept functions *C*:** represent instantiated instances of activities, actions or events.

4.3.5 Gesture recogniser (COGS)

The Gesture Recogniser (GR) module is contained in the view-independent area of the ActIPret system and will use a Time-Delay Radial Basis Function (TDRBF) network to categorise characteristic purposive 3-D hand trajectories (gestures) over time. During the learning phase, the network learns the appearance of the general categories of these hand trajectories from a database of example data.

The tasks concern manipulation of objects by hands in 3-D space. Therefore the major purposive hand gestures (or sub-activities) that need to be recognised by this module are:

- A hand moving away from the torso
- A hand moving towards the torso

On initialisation by a service request from the activity reasoning engine, these can be recognised by the GR via a TDRBF network using either 3-D hand trajectory information (relative to the torso centroid) or 6-D hand/pose trajectory information. Information is passed back so that complex activities (such as the hand grasping an object) can be recognised via additional attentive queries, such as `Is the hand empty or holding something?'

Where more than one hand is present in the scene, several parallel service requests can be made, one for each hand.

4.3.6 Object relationship generator (ACIN)

The component Object Relationship Generator controls the generation and maintenance of spatio-temporal relationships between two objects. It determines relationships between the objects (hand, CD, player, button) in 3-D at one instance or over several cycles. The relationships can be quantitative (frame relative to frame) or qualitative (proximity, above, left, moves along, etc.).

Specific functions

- to confirm/create specific deictic relationship queries. This uses specific references to objects for answering a specific (single shot) query, for example *proximity<Obj1, Obj2>*, where *Obj1/2* is a reference to a given object;
- to create deictic relationships. This uses other components to obtain potential object candidates to establish the relationship. The relationship determines levels of task-related interest of combinations, e.g., proximity. The evaluation of the relationship

may be recreated each cycle or updated. Priorities for this would be determined by the Control Policy, e.g., *'find objects on hand trajectory'* (with priority *p*); and

- to update current deictic relationships. This newly evaluates a previously created relationship each cycle by using the updates from other components such as tracking.

Specific relationships

The specific relationships derived by the Object Relationship Generator are simple, task-related operators and are in two forms (see deliverable 3.1 for more details): *Basic relationships* are generally behaviours of a single object or simple relations between two objects. *Aggregate relationships* are relations between two (or more) objects and in the spatio-temporal domain. Of particular interest for the ActIPret system will be:

- **Purposive behaviour trajectory:** a single object trajectory is of interest for the synthesis process, if the object (hand) makes a purposive behaviour trajectory, that is a trajectory that is task relevant. An example is a hand moving towards an object. The goal is to detect the purposive trajectory as early as possible to be able to focus processing to the area indicated by the direction of the trajectory. An example is grasping, where processing should be focused on the objects at the estimated grasp location. Methods to detect a purposive behaviour trajectory can be based on Kalman filters (as proposed by ACIN) or a time-delay Radial Basis Function (RBF) network (as proposed by COGS).
- **Distance between two objects (mutual proximity):** the exact distance between two objects depends on the representation of the objects used. The simplest representation uses the reference coordinate frames. In this case the distance is the Euclidian distance between the origins of the two object coordinate frames. If the objects are represented with a hull, the distance can be defined as the closest point between the two hulls.
- **Find objects near to each other**: this service provides a pre-attentive predictive cue, based on the assumption that the closer two objects are to each other, the more likely they are to have some task-relatedness.
- **Object near trajectory of object:** this service provides a pre-attentive predictive cue, based on the assumption that the closer an object is to a hand's trajectory, the more likely it is to be manipulated by that hand.

4.3.7 Service list with view controller (Profactor)

The Service List with View Controller is very similar to the normal Service List, which is specified by ACIN. This process takes over additional functionality to assign a specific robot to a requested service.

This component contains the View Controllers, which are responsible to operate the view requests of the services. It contains functionality to merge the requested view points and initiates the movement of the robots. For details see Deliverable D6.1.

4.3.8 Hand detector and tracker (FORTH)

The functionality of this service is to detect and track human hands present in the scene viewed by the ActIPret stereoscopic vision system. This functionality will be based on:

- **Colour information:** skin colour models will be used as a cue indicating the presence of hands in the viewed scene. Several models are currently under review.

- **Motion information:** since hands participate in activities and actions, a skin-coloured region in the image becomes more salient if it moves. Motion is measured with respect to a world-centred coordinate system, which actually coincides with the coordinate system in which camera positions are measured and reported.
- **Depth information:** depth information is necessary for reporting 3-D position of hands in the image. The "locality of reference principle" (in simpler words, the continuity of the 3-D locations of a hand as a function of time) will be used to resolve ambiguities.
- **Tracking mechanisms**: the previous modules provide visual cues to tracking mechanisms that will track the various hand hypotheses in the scene. Kalman filtering is a classical tracking mechanism. Additionally, current investigations include particle filters and more specifically, tracking based on the condensation algorithm.

The above-mentioned techniques are accompanied with techniques for computing camera positions that are beneficial for observing the scene and tracking the hands present in the scene. For the moment, two criteria are considered:

- The most salient hand hypothesis should remain in the field of view of both cameras.
- The viewpoint should be such that the most salient hand moves in an almost front-parallel plane (appears to be moving laterally).

4.3.9 Object detector and tracker (ACIN)

The functionality of this component is model-based detection and tracking of objects. Tracking means to regularly determine the object location (and orientation) in 3-D in soft real-time (latency not longer than several frame cycles). 3-D pose is extracted from one or more 2D camera views. The model is either a wire-frame representation using vertices, lines, ellipses and regions (for man-made objects like a CD player) or just colour, colour histograms or texture (e.g. for regions). Detection is one possible initialisation of tracking and can be based on colour, colour histograms or texture derived from the model. Using data of extern recognition is another way for initialisation.

The principal output of this service is 3-D pose data (position and orientation) relative to a world coordinate frame at each time step (i.e. an update every 33ms), together with an uncertainty value for pose accuracy (standard deviation) and a confidence value for object tracking.

Additional output could be the re-projection of the object into images, image location of (found/projected) features, the location and confidence values of individual features in images in 2D (e.g. to inform other trackers about possible occlusions - see section 3.), and possibly the transformation to other coordinate systems (e.g. camera, head).

4.3.10 Object recogniser (CMP)

This component uses an example based object representation to learn and recognise objects in a scenario. See also Deliverable D4.1.

4.3.11 Hand recogniser (FORTH)

To be defined if more than hand detection is required.

4.3.12 Motion detector (FORTH)

The Motion Detector module finds moving objects in the scene, while the observer is also moving. Motion is measured with respect to a world-centred coordinate system, which actually coincides with the coordinate system in which camera positions are measured and reported. Since the observer also moves, the solution of the problem relies on knowledge of the depth of the scene. Provided that depth is known (stereoscopic processing) and camera egomotion is also known, a motion field can be predicted, assuming that the 3-D world remains rigid. The actual flow is then compared to the predicted flow and significant deviations between these two flows are attributed to independently moving objects.

4.3.13 Ellipse detector (CMP + ACIN)

Detects ellipses in an intensity (colour) image. Ellipses can be partially occluded. Ellipses are found by grouping of convex edge strings.

4.3.14 Image server (camera) (Profactor)

The Image Server is responsible for providing the images of cameras, which are mounted on a robot. Since it is not a good idea to send whole images with TCP/IP, we will use shared memory to transmit the data.

4.3.15 Pose server (Profactor)

The Pose Server is responsible for providing the current position and orientation of the Robot-TCP (Tool Centre Point). The services also provide the current information of the camera angles, which are mounted on the robot.

4.3.16 CPU controller (Profactor)

The CPU Controller is a service, which determines the available CPU-Time of a specific computer. The information is used to find a computer with the lowest CPU usage for a requested service.

4.3.17 Scene modelling for visualisation (CMP)

The three-dimensional scene will be described using the VRML standard and presented using third-party VRML browser/viewer. The objects can be parameterised (scale, colour, surface texture, etc.). Humanoid model should be H-Anim 1.1 compliant and it can be parameterised, too. Data describing motion can be converted to the VRML interpolator format and used in VRML scene.

4.4 Framework tool selection

A very first version of the ActIPret framework must be presented after year 1. For successful integration and testing this early version of the framework, establishing the component communication as soon as possible is imperative. A first evaluation did show that the ActIPret Framework must base on an existing component-based software-tool, which is on a higher abstraction level than middleware such as RPC [5], ACE [7, 8], or CORBA [9].

Middleware: Considering safety of investment in the future and the existing interfaces, CORBA of the OMG [10] is superior to ACE and RPC.

4.4.1 The tools for the framework

Evaluated Framework-tools / Framework concepts are: RCS, NASREM [1], SAPHIRA [2], BERRA [3], DAMN [4], AYLLU [4], MOBILITY [11], OROCOS–Patterns [13], SMARTSOFT [12], OSCAR [14]. The evaluation was done according to availability, modern design, clear component design and debugging support.

A preselection phase did show that the most suitable versions are MOBILITY, SMARTSOFT and OSCAR. Up to now there is no implementation of the OROCOS concept available. Because it is an interesting concept of an impressive and powerful consortium the design pattern of OROCOS was also part of the evaluation.

As best fitting middleware, we identified CORBA. These 4 where evaluated in more detail according to the following topics:

Availability:
1. Sources Available?
2. Available Licences?
3. Documentation quality?

Easy to use
4. Required knowledge in Middleware communication
5. Support of the used Hardware
6. Debugging opportunities

Safety of investment
7. Will the system be maintained and improved in the future? How many engineers are working on it?
8. How long does the system exist
9. Will adaptation made in ActIPret migrate into the system or will every release cause new incompatibilities?
10. How widely is the system used now and in the future (estimated)
11. Is the system based on CORBA?
12. Prior experience with the system?
13. Open Source?

Functionality supported
14. Vision modules
15. Robotic Modules
16. Variety of supported functionality
17. Suitability of functionality

Ranking: 1 good – 5 poor, points that are of specific relevance are marked

🟥 Reason for rejection

🟨 Strong contra-point

🟩 Positive Point

#	SMARTSOFT	MOBILITY	OROCOS	OSCAR
1	🟨 3 (the system is partially available and the developer has promised send a free version. But not received yet)	🟥 5 (The system is not available for tests)	🟥 5 (The system is now in the definition phase and not available in the near future)	1 (The system is already available)

2	3 (system in under the GPL licence)	5 (it is a commercial product and the licence is not available now)	1 (system will be under the GPL licence)	1 (free licence available, will be made open source)
3	1 (well documented)	2 (documentation focussing on RWI robots)	- (documentation of the communication pattern available)	5 (limited documentation in German only)
4	2 (knowledge of the underlying middleware ACE is not necessary)	3 (only some knowledge about the naming service of CORBA is necessary)	(The underlying middleware is not known yet)	2 (knowledge of the underlying middleware CORBA is not known. Only IDL knowledge is necessary)
5	3 (no support for the hardware, but no barrier to implement it)	3 (no support for the hardware, but no barrier to implement it)	(Till now no support for hardware)	2 (support for the AMTEC Hardware already available)
6	2 (the system provides debugging functionality)	2 (the system provides debugging functionality)	(no debugging functionality defined till now)	2 (the system provides debugging functionality)
7	3 (the team of the developers is very small)	2 (the team of the developers is small. The web presence is decreasing since bought by iRobot)	1 (the consortium is very impressive)	3 (very small team)
8	1 (the system exists for 5-6 years)	1(the system exists for 4-5 years)	5 (the system is not existing yet)	1(the system exists for 4-5 years)
9	5 (the migration of adaptations is very unlikely)	5 (the migration of adaptations is very unlikely)	4 (extension of patterns unlikely but possible)	2 (the migration after testing is promised by the developer)
10	3 (there are estimated 7 institutes they are using the system)	2 (the system widely distributed with RWI robots)	1 (Many European laboratories participate in the discussions and design)	4 (there are 4 institutes so far they are using the system)
11	4 (the system is ACE-Based)	2 (the system is based on TAO "The Ace Orb")	2 (there are independent patterns so far)	1 (the system is based on ORBacus 4.x, which is a free available CORBA implementation for no commercial use)
12	5 (there is no evaluation version available up to now)	2 (FORTH has experience with the system. But there is no evaluation version available)	(the system is not existing yet)	1 (PROFACTOR and ACIN have experience with the system)
13	3 (in principle are parts open source. Evaluation version not yet available)	5 (no version available)	1 (the plan is to make the system open source)	2 (the sources are available at PROFACTOR. There are plans to make it open source)
14	3 (there is a wide support of vision modules)	3 (support for all kinds of sensors)	2 (support for vision modules is estimated)	2 (optimised on multi-cue systems with ring-buffer concepts)
15	3 (the good patterns are usable for robotic modules. Currently only mobile robots are tested)	3 (robotic functionality is mostly supported for RWI robots)	1 (very well elaborated patterns for robotic modules)	4 (the supported functionality is mostly for mobile robots. The usability for manipulators is tested)
16	1 (a wide variety of functionality is provided)	2 (a wide variety of functionality is provided)	1 (it is planed to provide a wide variety of functionality)	4 (mostly functionality for communication is provided)
17	2 (The communication functionality is very useful)	3 (communication functionality is not very well supported)	1 (it is planed the provide very useful functionality for robotic systems)	2 (The easy communication functionality is very useful)

Table 2

Conclusion: OROCOS and MOBILITY are not available (within the early period of WP1, or at all) and therefore cannot be considered. However, due to the expected impact OROCOS should be observed. SMARTSOFT is based on ACE, and availability is very limited.

OSCAR is with limitations usable, easy to use, based on CORBA, and will incorporate Cognitive-Vision-Framework's specific parts developed in ActIPret after testing. Therefore, OSCAR is chosen. Free licenses of OSCAR are available for the ActIPret and OSCAR will be made open source next. For safety of future investments, the following improvements are suggested:

1. Increased efforts for documentation
2. Interfacing to OROCOS Framework as soon as available

4.4.2 Hardware

The Demonstrator will be build up based on 4 AMTEC Manipulators or Pan-Tilt Units with 3-6 DOF. The manipulators will be controlled via a CORBA interface developed at PROFACTOR which communicates with a Real-Time Robot-Controller.

The manipulators carry stereo-pairs or triples of ieee1394 cameras. SONY DWF500L zoom-lens cameras will be used for the manipulators, Basler A320f cameras with fixed-focus lenses are considered for the pan-tilt units.

Scenarios, example videos (Profactor):

Two example videos for the scenario insertion of a CD in a player have been logged so far. The videos are taken with verging, but non-moving Pan-Tilt-Unit for varying view on the CD player and operator with changing starting scenario. Next example videos will be made with moving camera-system as soon as a first framework-version is available that allows synchronised data logging of video-data and camera poses.

4.4.3 References

1. R. Lumia, J. Fiala, A. Wavering, "The NASREM Robot Control System Standard," Robotics & Computer-Integrated Manufacturing, Vol 6, N 4, 1989, pp 303-308.
2. Konolige, K. and K. Myers. "The Saphira Architecture for Autonomous Mobile Robots," www.ai.sri.com/~konolige/saphira/
3. M. Lindström, A. Orebäck, und H. Christensen, "Berra: A research architecture for service robots," in Intl. Conf. on Robotics and Automation (Khatib, ed.), vol. 4, (San Francisco), pp. 3278-3283, IEEE, Mai 2000.
4. J. Rosenblatt, DAMN: A Distributed Architecture for Mobile Navigation, doctoral dissertation, tech. report CMU-RI-TR-97-01, Robotics Institute, Carnegie Mellon University, January 1997.
5. B. B. Werger, "Ayllu: Distributed port-arbitrated behavior-based control," in Proceedings of Distributed Autonomous Robot Systems 2000, Springer Verlag, (Knoxville, TN), October 2000.
6. Sun Microsystems, Inc., "RPC: Remote Procedure Call Protocol Specification," Version 2, RFC 1057, June 1988.
7. D. C. Schmidt, "ACE: an Object-Oriented Framework for Developing Distributed Applications," in Proceedings of the 6th USENIX C++ Technical Conference, (Cambridge, Massachusetts), USENIX Association, April 1994.

8. Schmidt, D., "An Architectural Overview of the ACE Framework: A Case Study of Successful Platform Systems Software Reuse", USENIX Login Magazine, Tools Special Issue, November 1998.
9. Object Management Group, “The Common Object Request Broker: Architecture and Specification,” Revision 2.6, December 2001.
10. The Object Management Group home page. http://www.omg.com/
11. iRobot Corporation: Real World Interface. http://www.irobot.com/rwi/p10.asp
12. C. Schlegel, R. Wörz, “The Software Framework SmartSoft for Implementing Sensorimotor Systems,” IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS '99 1610-1616, Kyongju, Korea, October 1999.
13. C. Schlegel, “Communication Patterns for OROCOS, Hints, Remarks, Specification,” Draft Version 0.11, January 2002.
14. S. Blum, “OSCAR - Eine Systemarchitektur für den autonomen, mobilen Roboter MARVIN,” In *Autonome Mobile Systeme*, Informatik aktuell, pages 218-230. Springer-Verlag, November 2000.

---END OF DOCUMENT -----