Interpreting and Understanding Activities of Expert Operators for Teaching and Education

DELIVERABLE D6.3 (v1.0)

Description of methods to provide investigative behaviours and their evaluation

28 April 2004

Authors: *Gerald Umgeher, Christof Eberst*

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 presents the design of the methods developed within WP6, which deals with attentive and investigative behaviours and their co-ordination. In a system like ActIPret, which has restricted resources, it is essential to control the processing of visual behaviour / services. The most restricted resource in ActIPret system is the view[1] therefore handling the resource "view" is of the main interest in WP6. A fitting view is vital for the vision components to allow robust and improved performance ([1][2][3]). One option how the resource view can be handled in a more intelligent manner is to use active vision systems.

2 Active Vision System

The following section 2.1 will give a short overview of the benefits of an active vision system and which additional afford is necessary to handle such a system. Section 2.2 presents the hard- and software used in the ActIPret system for the active vision system.

2.1 Benefits and drawbacks

An active vision system compared to the classical static approach is able to interact with the environment and can adapt its view point actively to unexpected situations and exceptions. The scene of an active system is less restricted since the camera can adapt its position to provide non-discrete dynamic view points, that satisfy dynamically changing constraints best possible and which can be used to reduce the amount of necessary camera systems. Other important reasons why the active vision approach is used within the ActIPret system is the possibility to fix the attention to important parts of the scene, to follow the movement of objects like the hand and to observe objects in more detail which is quite useful for small objects like the buttons of the CD-Player in the ActIPret scenario. All these advantages are helpful for tracking services (Hand Tracker, Ellipse Tracker) and services like object recognition. Since the active components within the ActIPret-Framework [6] are not statically predefined, but dynamically requested from higher-level components or behaviours, with own cognitive capabilities, realization of these advantages requires to deal with more complex control mechanisms for resource management.

2.2 Active Vision in ActIPret

The ActIPret demonstrator consists of two heterogeneous AMTEC robotic systems [4]. Robot system 1 is a 4 degree of freedom stereo camera head visualized in Figure 1, system 2 a 6-DOF robot illustrated in Figure 2. On each robot a stereo camera pair is mounted, using Sony FireWire cameras (DFW-VL500) 5.5 – 64 mm. The robots are controlled by a PC based industrial controller enhanced with a CORBA-based interface, which provides a simple and efficient command set to command the robot directly out of the ActIPret Framework.

[1] As view we define the assignment of the camera and the control over its orientation that the camera has a requested 3D point in the image centre.

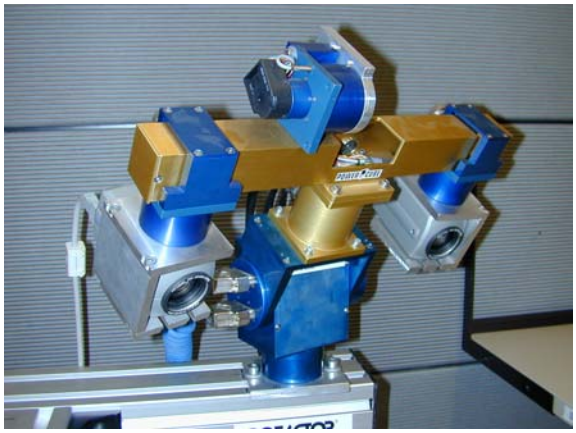


Figure 1: Stereo camera head

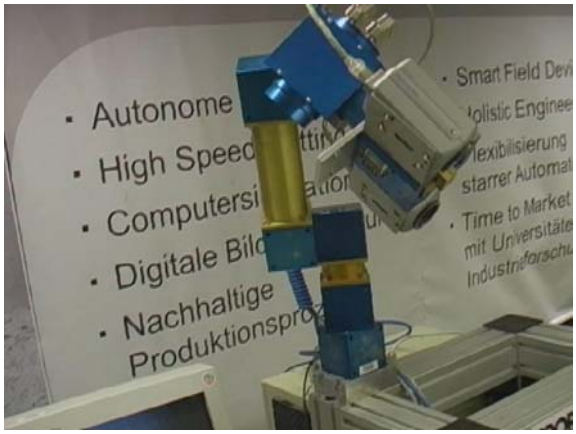


Figure 2: 6-DOF robot system

3 Concept

This section presents the concept of the control mechanism for investigative behaviours / services. Section 3.1 mainly deals with the concept of shared responsibility.

The concept to control of the behaviours and the resources follows the following objectives:

- Enhancing vision components towards active behaviours
    - The character of the behaviour (investigative, attentive) is defined by the vision components alone, independent from the behaviour-control process and the standardized communication patterns in the framework.
    - Support to maximize the efficiency of the cognitive vision system.
    - Support to exploit the cognitive capability of single components and the overall system by distributed control aspects.
- Simplicity, extendibility, adaptability
    - Creation of a behaviour / vision component that requests resources shall be as simple as possible to be programmed.
    - Components will require only restricted “local” knowledge for goal-directed interaction within the ActIPret system and to perform useful behaviours. No single component needs knowledge about large parts of the system. Every component acts only on knowledge that is intuitively liked to its task: E.g. a Vision component has no own knowledge about the overall system, or the robots.

- "Intelligent" behaviour of the overall active system will arise from interaction of these components.

- HW independence and exchangeability. Exchange of HW / SW shall only need adaptations of the directly connected component, with no further consequences for the whole system.

3.1 Shared responsibility

The idea behind the control mechanism is based on a de-central approach in which each involved service has limited specific responsibility, that matches the "local" knowledge inherent to its task.

For the case of processing visual behaviours or services, which have to deal with the resource "view", the robots which directs the cameras must be controlled in such a way that the overall system satisfies potentially multiple dynamically varying constraints simultaneously best possible. In the presented distributed approach, each involved component has a limited specific responsibility for controlling the overall cognitive active system (shared responsibility). By interaction of the components, the final control of the robot attention considers:

- Dynamic and reactive service selection by the requesting component. The selection process matches the service description of the providing components, against **task and situation specific** requirements of the requesting component and the system. This functionality is needed to handle the dynamics of the scene and involved objects.
- A **local** highly reactive and rapid response to changes in the scene and requirements of the vision components. Rapid response is achieved by the direct communication between the view requesting component and the view controller as well as the autonomy / responsibility of the view controller for its local operations.
- **Strategy to increase cost and quality of the services towards (local sub-) optima. Self-evaluated cost and quality** features of components are used for local control-decisions that target minimizing cost and maximizing quality of the executed services according to their priority – if given. Cost and quality of services over time is used for rapid local conflict management (potentially up to stopping services which impair other services) and fusion of view requests.
- Scalability for the heterogeneous robot system and **independence** of hardware and software is crucial.
- Allow hand-over of services from one component assigned to one robot to another component activated on a second robot.

Distribution of decentral knowledge and responsibilities:

- Vision / Interpretation components:
    - Know: what output they can provide under what **view conditions** and what input they need. What is their point of interest.

- Know not: What / how many robots and cameras exist and what their status is.
- Decide: Select lower level service and (if active) request views to a given point of interest.
- Decide not: Which robot they are assigned to, which other components are running on the robot, where **the robot must point to**.

- Robot Control components:
  - Know: own motion capabilities and dangerous zones, own status and pose, services to serve and the quality to fulfil requests of individual components, quality and cost for serving a new vision component to be started and different options and combinations.
  - Know not: Other robots in the system, tasks of vision components.
  - Decide: Where to look to satisfy the view request of assigned services simultaneously best possible. Pre-select possible combination of components if a new component shall be started and consequences on cost and quality. “Freeze in” view request of a conflicting component and propose its termination.
  - Decide not: Which component to be started, which vision component to be selected.

- View Contract Manager:
  - Knows: Overview on status of components running on individual robots (including quality and costs features).
  - Knows not: Kinematics, pose or specific features of robots. Actual position of the robots. High-dynamic smaller changes of cost and quality of the components assigned to a robot. Task or properties of vision components.
  - Decides: To which robot to assign a requested component, and potentially in which redundancy (!) and potentially which other service to be terminated. Hand-over procedure of components from one robot to another.
  - Decides not: which vision component-class is selected by the requesting component.

3.2 Behaviour control mechanism

Initial service selection and assignment

If a requesting component needs a specific service to fulfil its task it uses the functionality of the service list, which is part of the ActIPret framework, to get a pre-selection of all services which are potentially useful, according to its requested service properties. The properties of a service to be requested and the initiation of the pre-selection according to these individual service properties are statically or reactive defined inside the component program. The component then selects the best fitting service(s) among the ones pre-selected by the service list.

If the service description of the pre-selected services indicates that the resource “active camera” is a pre-requisite for execution the system automatically delegates the final decision, which service (among the pre-selected ones) to be started on which robot / camera system, to the view contract manager. In this case the view contract manager component is responsible to provide an optimal selection of the view controller for one of the pre-selected services. The view contract manager will propose redundant robot – service assignment if this is supported by the requesting component and possible in the current load-situation. The communication sequence of the initial service selection is illustrated in Figure 3.

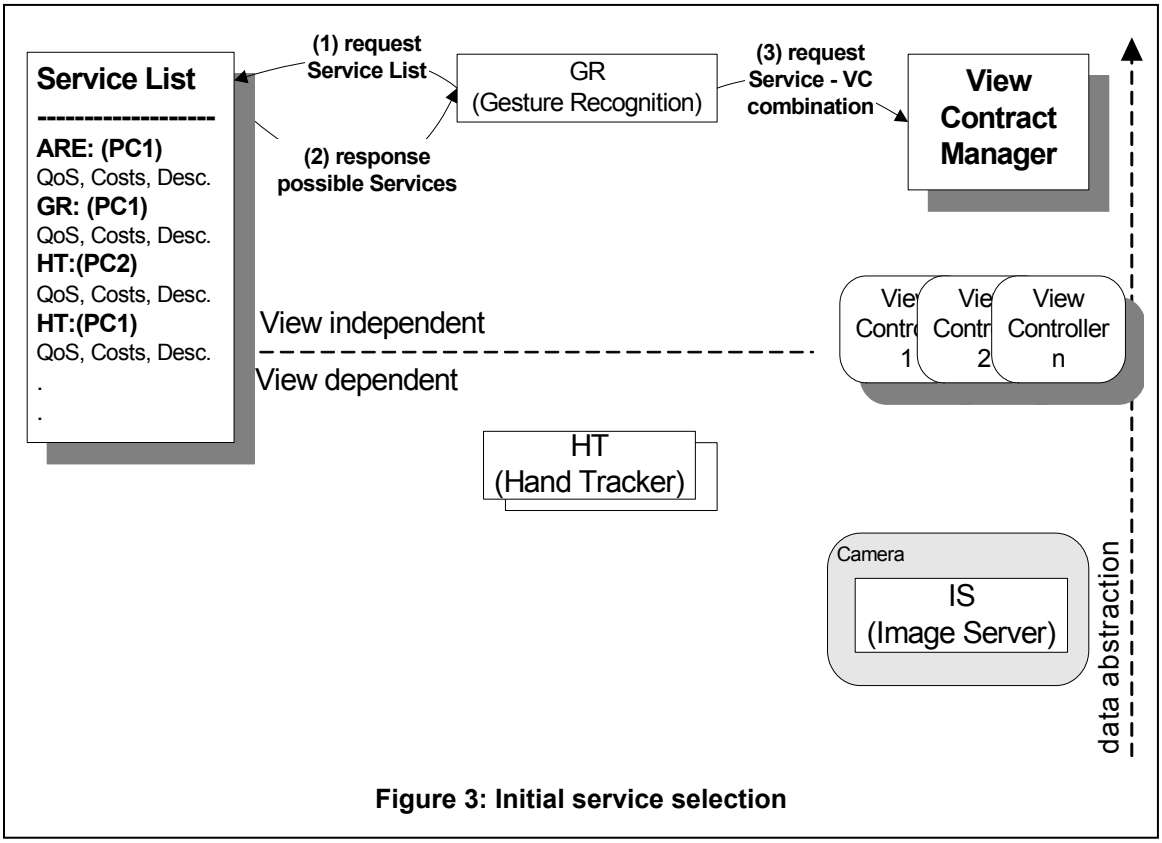


Figure 3: Initial service selection

To improve the selection of the best robot / camera system it is possible to specify a SOI (space of interest), which is used as an initial view request for the selected view controller. With this functionality the camera system focused to a task- and situation specific-view point. This is crucial for single-frame services like the object recognition service, where this functionality is used to direct the camera system to view points which are considered to be of importance by the requesting module, e.g. where hand-activities has been detected.

Interface of the VCM request

```
virtual void AssignViewController(
    std::string& rHostName, // hostname of requester component
                            // (task management in VCM)
    SOIType & rSOI,         // to control initial view point
    vector<ServiceOffer*> &rProviderOffers,
                            // vector with Service Offers of Provider
    ServicePropertyType & rRequesterProperty);
                            // QoS, priority of service requester
```

View controller selection

To get the necessary information for deciding which choice satisfies **global (**sub-) optimal criterion, i.e. which view controller should be assigned for the requested service, the view contract manager requests bids from the individual view controllers. The bid of each view-controller includes one to several options at which local costs and quality the service can be processed and which local resource conflicts with the currently processed services will arise.

The idea behind the concept of sending several options per bid – selected best by the VC - especially allows near-optimal conflict resolution. While the VC has no information to judge which local conflict causes largest overall impairment of the system performance, different options allow to propose e.g. combinations in which the new service is running in parallel to the older ones assigned to the resource robot – which usually leads to lower quality since the merged view request of several services are a compromise – or to terminate one or more services allowing to provide a view more tuned to the need of the newly start component. The later option will consequently features a predicted better quality and higher cost (need to terminate other components, typically larger motion of the arm from focussing view point of the old components to the one of the component to be newly assigned. Alternatively a bid can include the rejection of starting the component. While there is no promising solution how a local view controller with not knowledge about the total situation and system configuration can decide which are the system wide consequences, the view contract manager can select a global suboptimal solution (selection of a optimal solution would need that all bids include all possible options instead of the ones which are selected by the VC as locally optimal).

For services, which require a specific, situation dependent view point at start-up, the initial specified SOI is considered in the calculation of the costs and quality for the specific view controller.

Examples for costs are:

- Movement necessary to achieve the view point specified by the initial specified SOI.
- Number of already started services on the view controller.
- Conflicts with already started services.

Examples for quality are:

- Distance between the view point specified by the SOI and view point which can be provided.
- Spatial distance between the camera poses that satisfies the focussing on the view points.

The selected services are then started on the assigned robot by the service-requesting component.

The communication sequence of the view controller selection is illustrated in Figure 4.

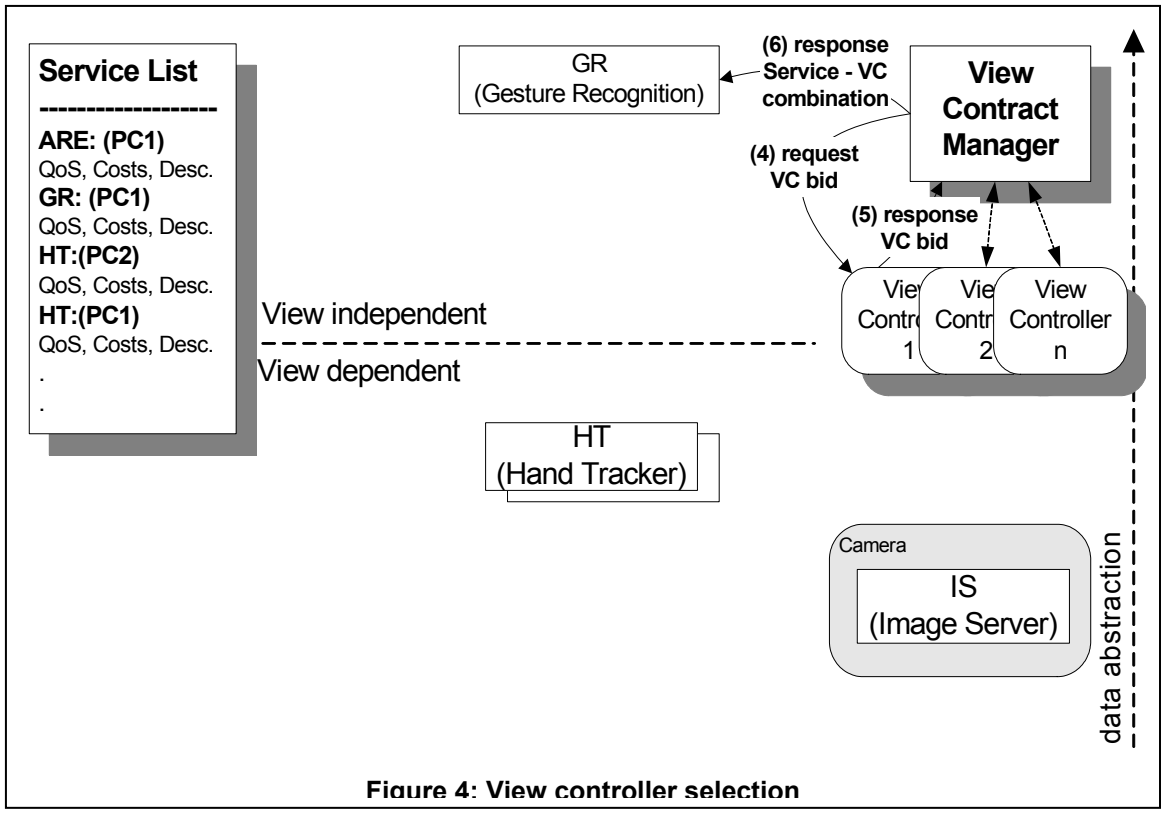


Figure 4: View controller selection

Interface of the VCM response

```
virtual void AssignViewController_Response(
  int & rKeyValue,                  // Internal parameter to access to
                                    // selected VC
  int & rSelectedServiceId,         // Index, specifies service offer
  std::string & rConstraintString); // constraint string for selected
                                    // ViewController
```

Run-time control

Each providing component offers its services to the requesting components. In the special case where the providing component needs a specific view to provide the best possible result the providing component itself is responsible to send the view requests to the view controller. To obtain the information which robot system / view controller should be used for the view requests the system sends automatically additional information via the service request to the providing component. This additional information is used as a constraint string in the service request for the view controllers to get the instance of the view controller assigned by the view contract manager.

The view request is – independent of the robot on which it is performed – specified as space of interest in 3D world coordinates. The requested space of interest specifies the view point and the optimal orientation from which to observe the object, plus the allowed parameter tolerances. The view controller checks the feasibility of the requests of the given robot, i.e. if it can be reached considering the robot's dexterous workspace. The view controller component also checks for inconsistencies with view requests from other services, which are active on this specific robot. According to these conditions the view controller recalculates local cost and quality for each service combination. In case of larger changes, it sends an update to the view contract manager, which can re-assign the service or initiate stopping services. In case of a conflict, the view controller can react rapidly by suppressing the requests of one service and by initiating the termination of the service (). For all non-conflicting, active services of the robot, the view controller merges the requests and calculates a new reachable view trajectory and commands the robot.

The run-time control procedure is visualized in Figure 5.

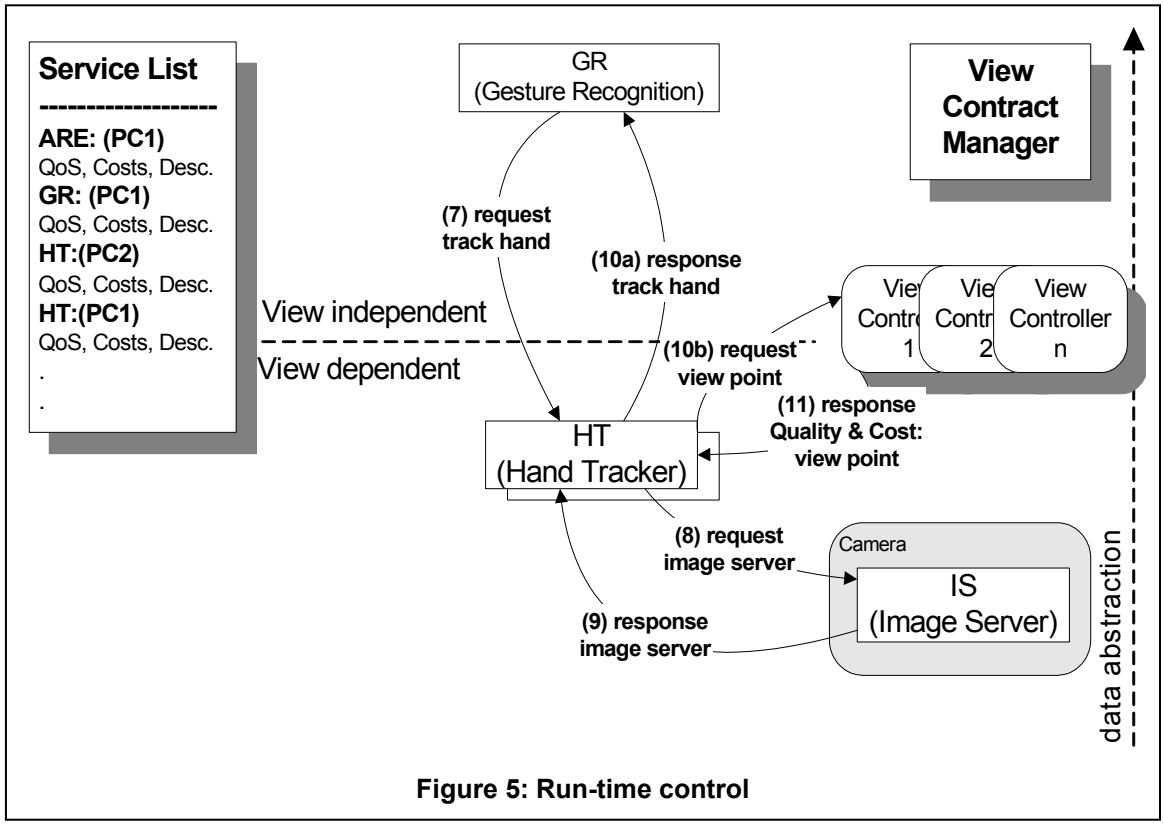


Figure 5: Run-time control

Interface of the view request

```
virtual void RequestView(
  int & rKeyValue,       // unique ID number of view request
  SOISeqType & rSOIVec); // SOI where component want to watch

virtual void RequestView_Response(
  double & rCost, // Calculated cost for the provided view point
  double & rQoV); // Calculated quality of provided view point
```

The actual vision-based behaviour arises from the interaction of the view requesting component, the view controller, the component that requested the vision component, from the view contract manager component and the interactions which are not-statically defined. The character of such a behaviour is clearly specified by the (investigative, attentive) character of the component that is connected to the view controller. The performance of the behaviour varies over time and is influenced by the sum of components which are assigned to the view controller and send view requests to it.

4 Implementing behaviour based control aspects in distributed components according to the concept

The coordination procedure described in chapter 3 with all its elaborated and complex control mechanisms is independent from the components and their character. Therefore it is hidden from the programmer of the components, releasing him from the difficult tasks and additional constraints. Chapter 4 presents the usage of the view contract manager and view controller from the component programmer point of view for “realizing” a desired robotic behaviour.

4.1 Overview

The role of the view controller (VC) is to allow a service which needs a specific view to process its task to achieve this view. The role of the view contract manager (VCM) is to choose the best fitting view controller for a view dependent service.

The first step needed is to request the best fitting services from the view contract manager. The view contract manager generates a constraint string and a key value for the service and sends this information in the response message.

The view dependent service is using this information (constraint string and key value) to establish the communication to the view controller. The simple communication sequence is visualised in Figure 6.

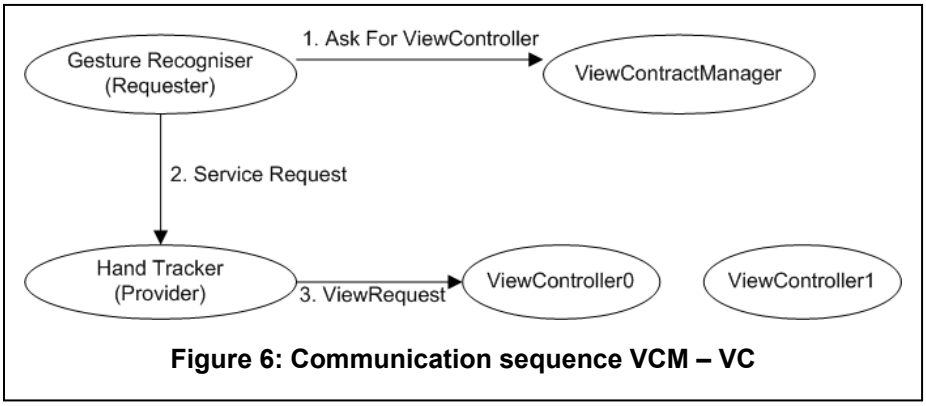


Figure 6: Communication sequence VCM – VC

All these functionalities and interfaces needed are hidden form the programmer by using a special inheritance structure visualised in **Error! Reference source not found.**:

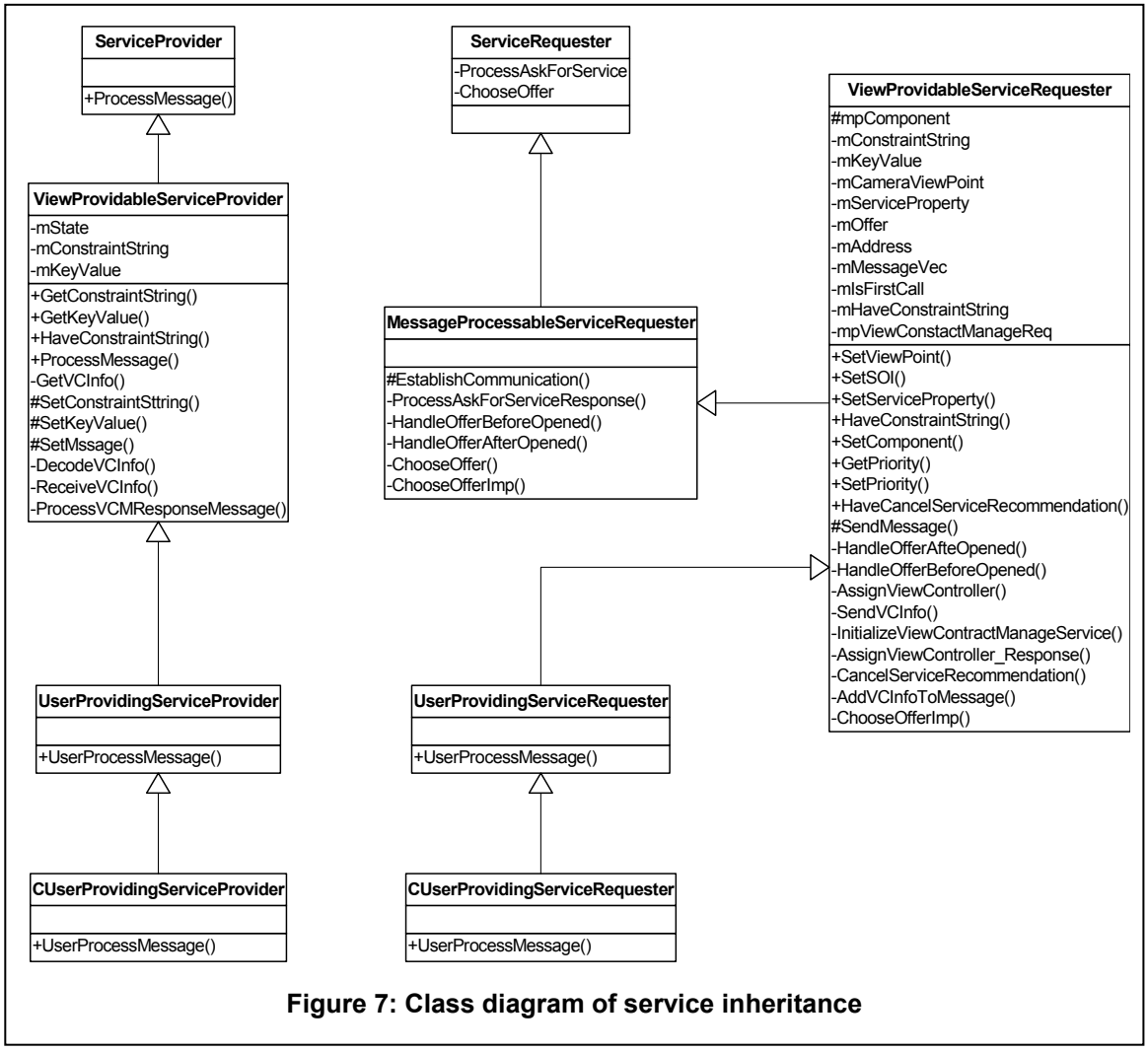


Figure 7: Class diagram of service inheritance

Example source code for the TrackObject service: communication between Gesture Recogniser and Hand Tracker:

```
#include "ViewProvidableServiceProvider.h"
// ServiceProvider
class TrackObject_Provider : public ViewProvidableServiceProvider
{
…
};


#include "ViewProvidableServiceRequester.h"
// ServiceRequester
class TrackObject_Requester : public ViewProvidableServiceRequester
{
…
};
```

4.2 Setting up the connection to the View Contract Manager

In order to get access to the View Contract Manager the `ServiceRequester` class has to derive the `ViewProvidableServiceRequester` class. The initialization of the requester instance is the same as for the normal service request. To initialize the View Contract Manager communication the component programmer has to call the `SetComponent()` function. If SOI (space of interest) is available it is necessary to use the `SetSOI()` function to initialize the View Controller with this position. The response of the View Contract Manager request is hidden from the component programmer and the returned information (constraint string and the KeyValue) is transferred automatically to the service provider.

Example source code for the Gesture Recogniser:

```
// Initialization of service requester instance
HandInfo_TrackObject_Requester *tracker =
  new HandInfo_TrackObject_Requester(this);

// Initialization of ViewContractManager communication
tracker->SetComponent( GetComponent() );

// If SOI is known, this information is used to initialize
// ViewController position
//tracker->SetSOI( SOI );

// same way as normal service registration
AddRequester(tracker, "SupportedDataModel == 'colour'");
```

4.3 Setting up the connection to the View Controller

In order to get access to the View Controller the `ServiceProvider` class has to derive the `ViewProvidableServiceProvider` class. Like in the normal service setup the component programmer has to create an instance of the requester interface. In addition to this the programmer has to use the constraint string and the KeyValue sent by the service requester.

Example source code for the Hand Tracker:

```
// Get additional information to identify the View Controller
SetMessage( pMsg );
std::string constraintString = GetConstraintString();
int keyValue = GetKeyValue();

// initialization of requester instance
mpViewController = new CViewControl_Requester() ;

// service registration
pComponent->AddRequester(mpViewController, constraintString );

// Use additional information to identify ViewContoller
mpViewController->SetKeyValue(keyValue);
```

4.4 Sending the View Request

Now the view dependent service can send its view requests by calling the `RequestView()` function.

Example source code for the Hand Tracker:

```
// view request data type
AIP::SOIType viewpoint;

// specification of view point
viewPoint.Origin.Position = ObjHypList[p].Pose.Pose.Position;

// request of view point
mpViewController->RequestView(viewPoint);
```

4.5 View dependent components in ActIPret

Within the ActIPret system several different components are employed to interpret the activities of human experts. Some of them are responsible to observe / interpret scenes and to calculate the pose of objects or operators in 3D world coordinates. To solve this task they need a specific view, which is best fitting for the object they have to process. All these components are therefore potential candidates to send a view request to the view controller. In ActIPret these components are:

Table 1: List of components using View Controller

Component Name	**Provided services**
Hand Detector & Tracker (HT)	HandPose, TrackHand
Object Detector & Tracker (ODT)	TrackObject
Object Recogniser (OR)	RecogniseObject

Before the communication to one of these components is established the View Contract Manager has to allocate the best fitting Service – View Controller combination. Therefore there are some components in the ActIPret system, which are responsible to request this combination for the View Contract Manager. In ActIPret these components are:

Table 2: List of components using View Contract Manager

Component Name	**Requested services**
Gesture Recogniser	TrackHand
Object Relation Generator	HandPose, TrackObject, RecogniseObject

A graph with all the components of the ActIPret system and the description of there communication links is presented in Figure 8. The components which are responsible to get the best fitting Service – View Controller combination are marked with a green square. The components using the View Controller to direct the robot / camera system to the best view point are marked with a blue square.

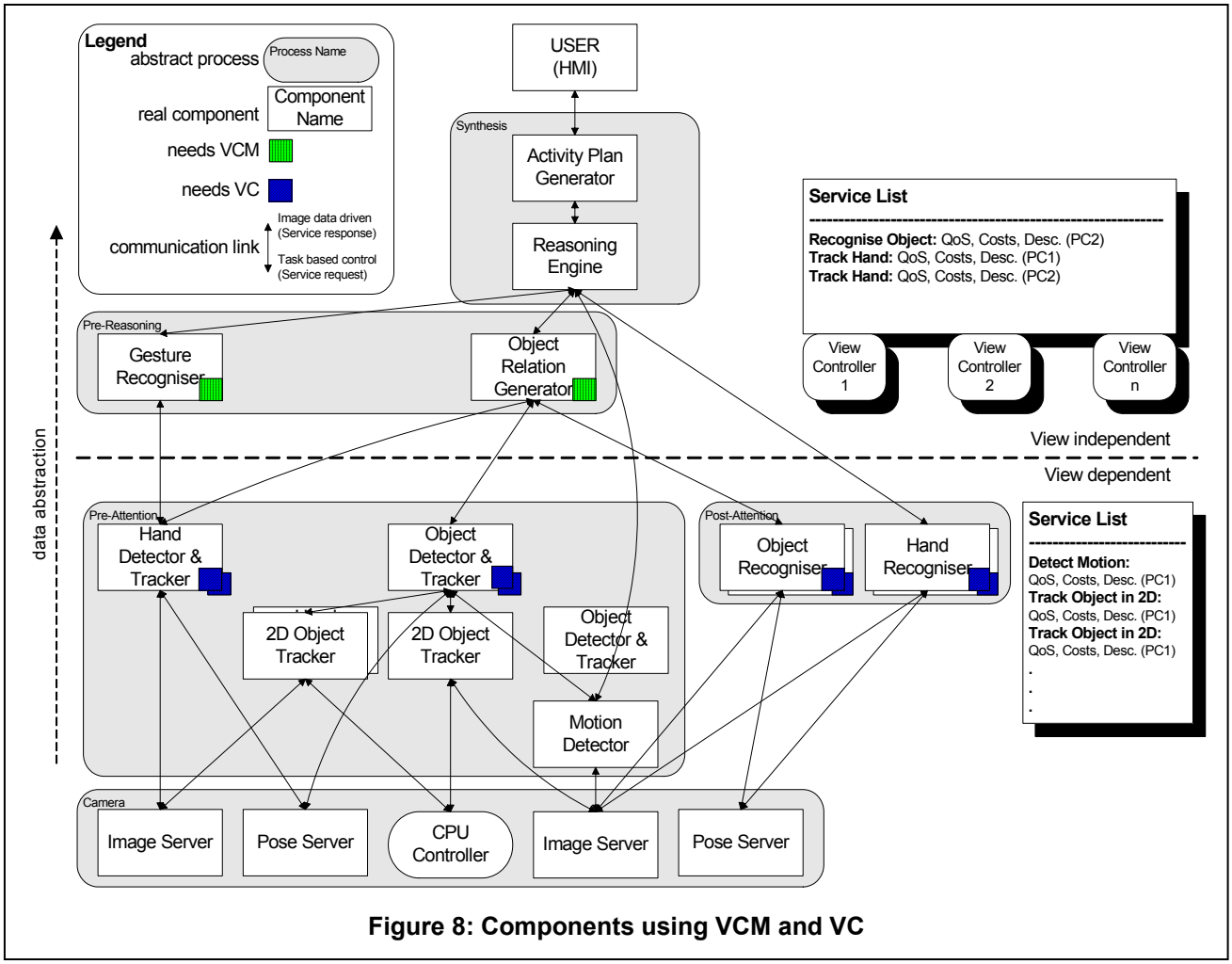


Figure 8: Components using VCM and VC

5 Experimental evaluation

Figure 9 shows the set-up in the laboratory at PROFACTOR. The stereo head in the top left is used subsequently to demonstrate active tracking of the operator hand. The stereo system shown in the top right is mounted on a robot arm.

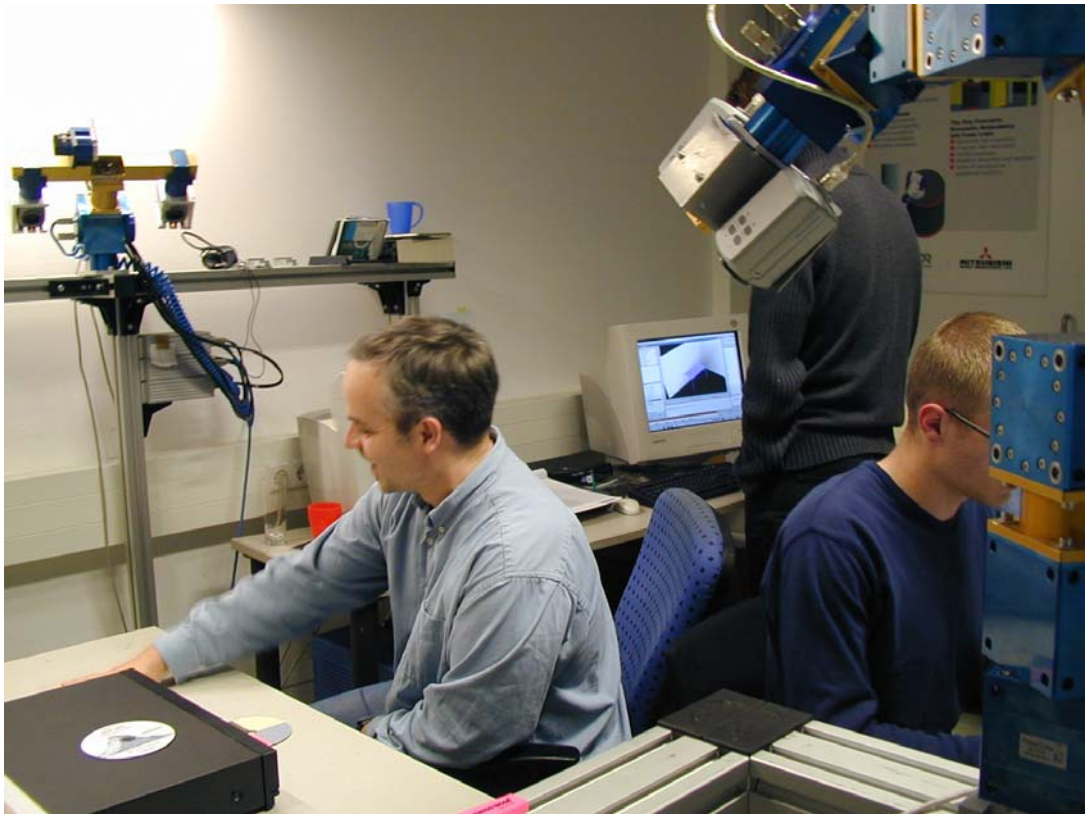
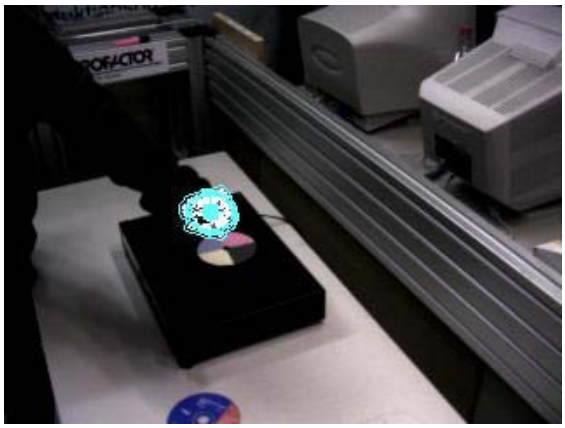


Figure 9: The ActIPret demonstration in the laboratory at PROFACTOR. Top left is the active fixed stereo system, on the top right the stereo system on the robot arm is seen. The screen in the background shows the GUI of the framework while Jon is placing the CD in the player.

Figure 10 shows an example stereo pair of images with the output of the hand tracker component. The detected hand is shown in cyan and other hypotheses are represented by other colours.

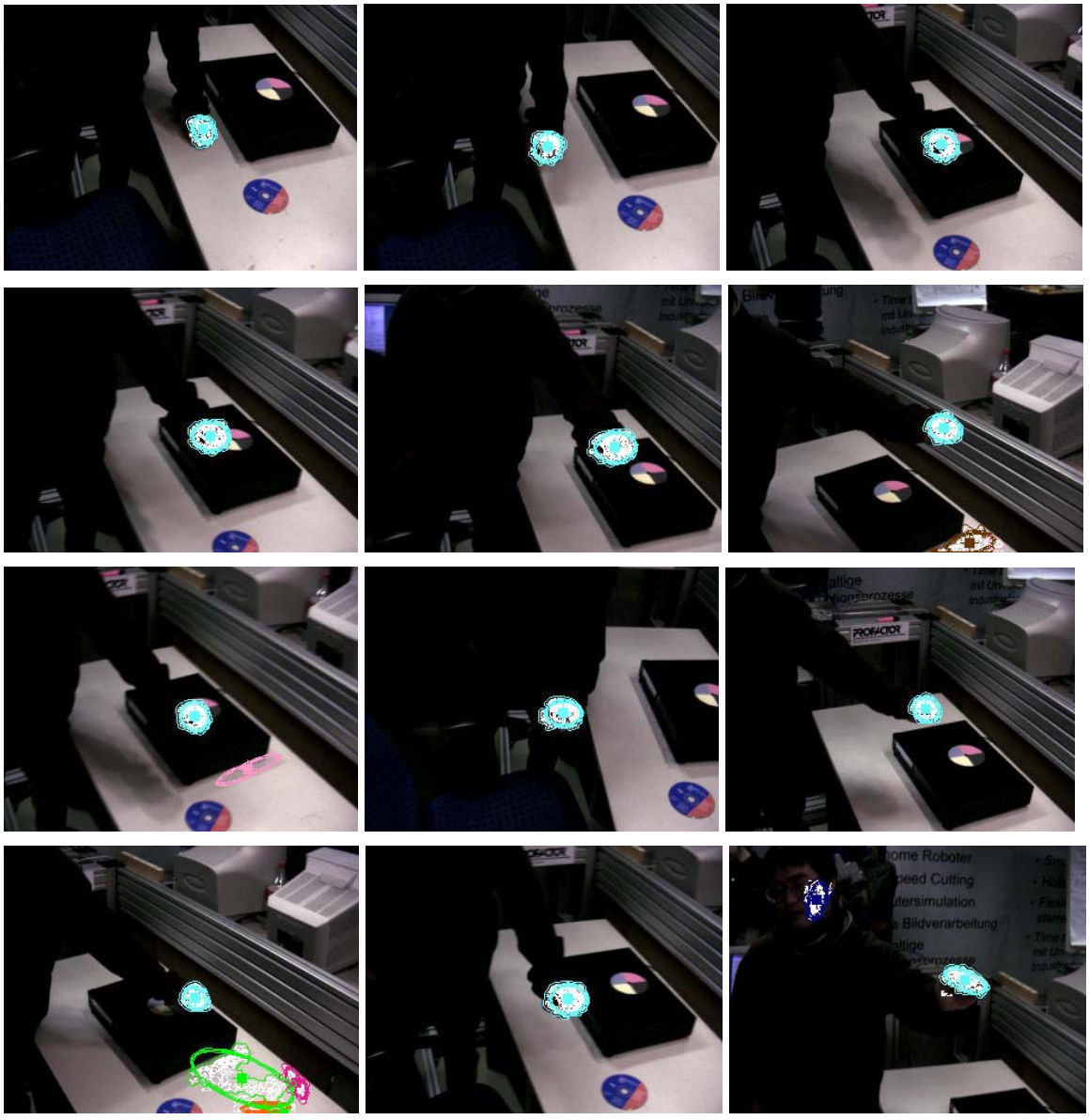
Left Image



Right Image

Figure 10: Example stereo pair from the Hand Tracker (HT) component.

Figure 11 shows a sequence of images from the stereo head during actively following the motion of the hand. The correct hand hypothesis is represented in cyan. Although there are other hypotheses form colour detection, tracking constraints keep following the correct hand throughout the complete sequence. The images are shown in steps of 667 ms.



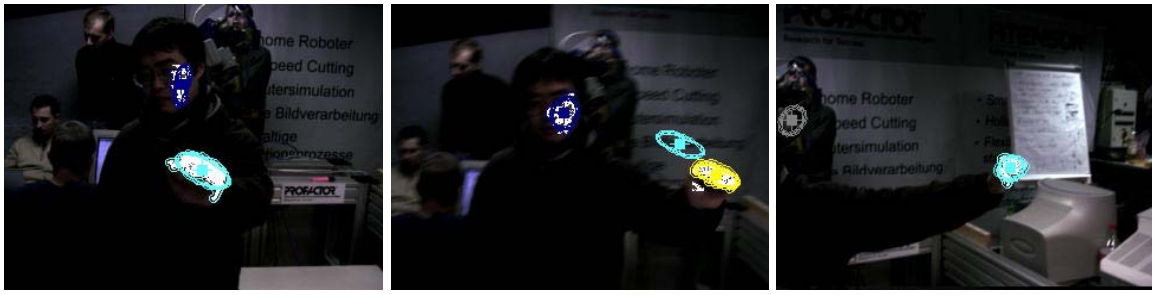


Figure 11: Actively following the hand motion

6 References

[1] C. Capurro, F. Panerai and G. Sandini, “Dynamic Vergence, In IROS96, pages 1241-1249

[2] J. A. Fayman, O. Sudarsky, and E. Rivlin, “Zoom Tracking”, In Proc. IEEE Int. Conf. On Robotics and Automation (ICRA `98), pages 2783 – 2789, May 1998.

[3] T. Lindeberg, K. Brunnström and J.O. Eklundh. “Active Detection and Classifiaction of Junctions by Foveating with a Head-Eye System Guided by Scale-Space Primal Sketch”. In ECCV 92, page 701-709, May 1992

[4] http://www.amtec-robotics.com/

[5] M. Takagi, C. Eberst, G. Umgeher: Control of Redundant Attentive and Investigative Behaviors in an Active Cognitive Vision System, 4th Asia-Europe Congress on Mechatronics, Saitama, Japan, September 2003.

[6] W. Ponweiser, G. Umgeher, M. Vincze: A Reusable Dynamic Framework for Cognitive Vision Systems, Workshop on Computer Vision System Control Architectures (VSCA 2003), Graz, Austria, March 2003.

[7] M. Takagi, C. Eberst, G. Umgeher: Control of Redundant Attentive and Investigative Behaviors in an Active Cognitive Vision System, Workshop on Computer Vision System Control Architectures (VSCA 2003), Graz, Austria, March 2003.