

CAVIAR

Context Aware Vision using Image-based Active Recognition



IST- 2001- 37540

CAVIAR : D 30 Report on Federation Assembly

Date: 31 October 2005

Author(s): James L. Crowley, Patrick Reignier, Sebastien Pesnel

Work package WP 5

Document status: Version 1.0

Usage: internal

Keywords: Software Architecture, Processes, Process Federations, Services, Dynamic Service Discovery, BIP.

Abstract

This report describes the BIP software system for assembling CAVIAR perceptual processes into federations. After a brief introduction, chapter 2 defines terminology and reviews the three layers of the CAVIAR software framework. The third chapter proposes a software architectural model for perceptual services based on the PRIMA Robust Tracking Process. The fourth chapter presents BIP, a Basic Interconnection Protocol for Event Flow Services that provides mechanisms for dynamic discovery and automatic configuration of Federations of perceptual processes. The fifth chapter describes the use of BIP for composition of process federations.

DOCUMENT HISTORY

Version	***Date***	***Reason of change***
0.1	september 2005	Initial documentation of BIP - Sebastien Pesnel
0.2	1 October 2005	Produced sections on software framework and process design Patrick Reignier
0.3	22 October 2005	Added discussion on Perceptual Process Model - James Crowley
0.5	28 October 2005	Added abstract and introduction - James L. Crowley
1.0	31 October 2005	Completed first version of report - James Crowley

Table of contents

1. Introduction **1**

2. A Component Based Software Architectures **2**

2.1 Some Terminology 2

2.2 A layered model for software components. 3

2.3 Modules 3

2.4 Perceptual Processes 4

2.5 Process Federations 6

3. An Architectural Model for Perceptual Processes **7**

3.1 A Model for Tracking Based Perceptual Processes 7

3.2 Process Supervisory Control 8

3.2.1 Process Scheduler *8*

3.2.2 Homeostasis and Autonomic Control *9*

3.2.3 Communication between processes *9*

4. BIP, A Basic Interconnection Protocol for Event Flow Services **11**

4.1 Specifications for the Basic Interconnection Protocol (BIP) 11

low latency *11*

reliability *11*

low threshold, high ceiling *12*

function reuse and distribution *12*

4.2 BIP Service overview 12

4.3 Connecting to a process 13

4.4 Communicating with processes 14

4.4.1 Low-level message format *14*

4.4.2 Establishing a link *15*

4.4.3 Communicating over TCP *15*

4.4.4 Communicating over UDP *16*

4.4.5 Communicating over multicast UDP *16*

5. Federation Assembly with BIP. **17**

5.1 An Example of a Process Federation 17

5.1.1 Specifying Process Federations *17*

5.2.3 Monitoring process execution *18*

Bibliography **19**

1. Introduction

This report describes a tool for configuring, launching and monitoring execution of perceptual processes and process federations in the CAVIAR system. We begin by introducing the component-based software architecture proposed for the CAVIAR project. Components are described within three layers: Modules, Processes and Federations. A standard design pattern (or model) is presented for perceptual processes based on augmenting the PRIMA robust tracker with a supervisory component capable of self-monitoring, self-repair and self-description. The self-description abilities of the supervisor make it possible to declare process capabilities to an ontology server, enabling both supervised and automatic assembly of process federations. Chapter 4 describes supervised and automatic software federation assembly using the BIP a tool for describing and assembling distributed perceptual processes. This report builds on and extends work that was reported in deliverable D15 at year 2. Some parts of D15 have been adapted and updated for inclusion in this report for clarity and consistence.

2. A Component Based Software Architectures

This chapter reviews the component based software architecture model proposed for the CAVIAR Project. A layered approach to component assembly is first proposed. Such an approach is designed to favor rapid assembly and experimentation of systems within the CAVIAR project. Auto-descriptive components are then described for three layers.

2.1 Some Terminology

Incorporation of concepts from Software Engineering has introduced some new terminology that has not previously been used in the CAVIAR project. This section defines some of this terminology.

Process: A processing unit that accepts input events and produces output events. It possesses a number of state variables (variable values) and properties (constant values), altogether referred to as parameters. Interaction with a process is achieved by establishing a link with one or several of its channels. [1]

Services: An abstract class of computation offered by processes, defined by transformations on events and streams [2].

Peer: An entity on the network that can communicate with a process (e.g. another process).

Channel A named point of communication between a process and one or several peers. One or more links can be established with a channel. There are three types of channels, depending on the direction of the flow of data: input channels, output channels, and “in-output” (duplex) channels. Note that this is different from system-level sockets, e.g. TCP or UDP sockets.

Link A means of sending messages to a process or receiving messages from a process. A link is comparable to an established socket-based network connection, except that no assumption is made on the underlying transport (TCP, UDP, multicast UDP).

Message An encapsulation of one or several events for transmission over a link.

Event A discrete piece of information that transits between a process and a connected peer.

2.2 A layered model for software components.

In previous reports we have proposed a component-based software architecture for constructing context-aware vision systems. This model builds on recent work on process architectures for machine perception and computer vision [3], [4], as well as on data flow models for software architectures [5]. Component based architectures, as described in Shaw and Garlan [6], are composed of auto-descriptive functional components joined by connectors. Such an architecture is well adapted to interoperability of components, and thus provides a framework by which multiple partners can explore design of specific components without having to rebuild the entire system.

In our architecture we apply the component model to the definition of software at three distinct layers [7], as shown in figure 1. These three layers are the Module layer, the Process layer and Federation layer. The components within each layer are defined in terms of the components in the layer below. Each layer provides the appropriate set of communications protocol and configuration primitives. The following describes the components within each layer.

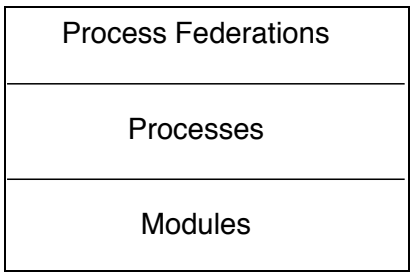


Figure 1. Three Layers in the Component based Software Architecture

2.3 Modules

Modules are auto-descriptive components formally defined as synchronous transformations applied to a certain class of data or events, as illustrated in Figure 2. Modules generally have no state. They are executed by a call to a method (or function or subroutine) accompanied by a vector of parameters. Communication of data and events is generally performed by passing pointers in the parameter vector, in order to avoid unnecessary copying of data. The vector of parameters specifies the data to be processed, and describes how the transform is to be applied. Output is also generally accomplished by writing to a buffer provided by the parameter vector, although it is possible to read or write to a stream or to post send events to an event dispatcher.

Modules return a result that includes a report of the results of processing. Examples of information contained in this report include elapsed execution time, confidence in the result, and any exceptions that were encountered.

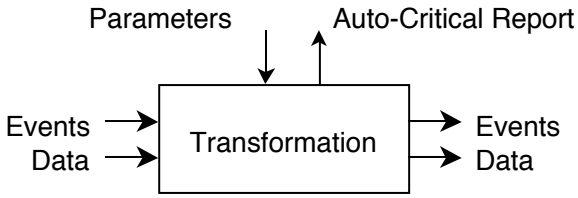


Figure 2. Modules apply a transformation to data or events or events and return a report.

An example of a module is provided by a procedure that transforms RGB color pixels into a scalar value at each pixel that represents the probability that the pixel belongs to a target region. Such a transformation may be defined using as a lookup table representing a ratio of color histograms [8]. The execute command is accompanied by a parameter vector that includes a pointer to the input RGBA image buffer, a pointer for an output image buffer, a pointer to a ROI (Region of interest) data structure that describes where the transformation is to be applied, a pointer to the lookup table, and a step size at which the transform is to be applied within the ROI. Computing time for this process may be reduced by restricting processing to one pixel out of S (S represents a "step size") [9]. A common use for such a module is to detect skin color regions as shown in figure 3.

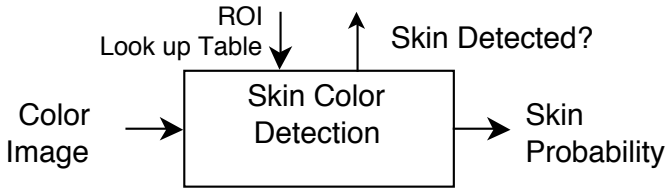


Figure 3. A module for detecting skin pixels

2.4 Perceptual Processes

The second layer in the architecture concerns perceptual processes. Using the terminology defined above, processes provide perceptual services. Perceptual processes typically communicate via synchronous data streams and asynchronous events through Channels to Peers or to a process supervisor.

Perceptual processes are autonomous assemblies of modules executed in a cyclic manner by a process supervisor. The process supervisor interprets commands and parameters, supervises the execution of the transformation components, and responds to queries with a description of the current state and capabilities of the process, and provides event management.

The auto-critical report from modules allows a process supervisor to adapt the execution schedule for the next cycle so as to maintain a specified quality of service, such as execution time or number of targets tracked.

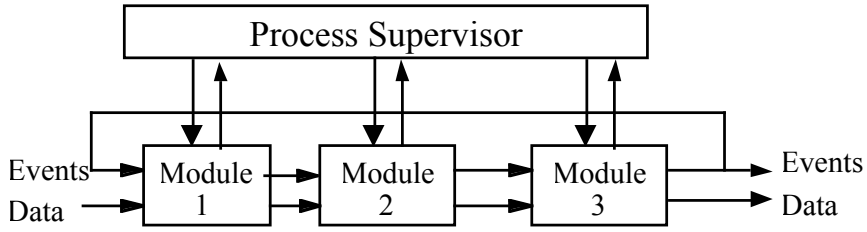


Figure 4. A perceptual process integrates a set of modules to transform data streams or events into data streams or events.

The supervisory component of a process provides five fundamental functions: execution scheduling, parameter regulation, event handling, command interpretation, and reflexive description. The supervisor acts as a scheduler, invoking execution of modules in a synchronous manner. The supervisor regulates module parameters based on the execution results. The supervisor provides a subscribe and publish service for events. The supervisor can act as a programmable interpreter, receiving snippets of code script that determine the composition and nature of the process execution cycle and the manner in which the process reacts to events. Finally, the supervisor responds to external queries with a description of the current state and capabilities. We formalize these abilities as the autonomic properties of self-monitoring, auto-regulation, auto-description and auto-criticism.

A self-monitoring process maintains a model of its own behavior in order to estimate the confidence for its outputs. Monitoring allows a process to detect and adapt to changing observational circumstances by reconfiguring its component modules and operating parameters. Techniques for acquiring an operating model, for monitoring operating conditions, and for repairing operational problems are described in Year 3 deliverable D29 Report on Error Recovery.

A process is auto-regulated when processing is monitored and controlled so as to maintain a certain quality of service. For example, processing time and precision are two important state variables for a tracking process. These two parameters may be traded off against each other. The process controllers may be instructed to give priority to either the processing rate or precision. The choice of priority is determined by the federation tool or by the federation supervisor. Techniques for monitoring output quality and for regulating internal parameters have been described in year 2 deliverable D 12 Report on Automatic Recognition Process Optimization.

An auto-descriptive process provides a symbolic description of its capabilities and state. The description of the capabilities may include either the basic command set of the controller or a set of services that the controller may provide to an ontology server for use in automatics federation assembly. This report is concerned with such methods.

2.5 Process Federations

A process federation [4] is a collection of independent processes that cooperate to perform a task. We have designed a middle-ware environment that allows us to dynamically launch and connect processes on different machines. This environment, called BIP, provides an XML based interface that allows processes to declare input command messages, output data structures, as well as current operational state. BIP is accompanied by BipTool, a software tool that allows inspection and supervision for federation assembly. BipTool allows a programmer to monitor the configuration and execution of federations that have been automatically assembled.

BipTool provides tools for configuring a process by sending snippets of control script to be interpreted by the process supervisory controller. Each control script defines a command that can be executed by a message from the PFT or from federation supervisory process. Processes may be interrogated to determine their current state and the current set of commands from either the PFT or the Federation Supervisor.

As a simple example of a federation of perceptual processes, consider a system that integrates targets from multiple cameras to provide 3-D target tracking, as shown in figure 5. The system uses an entity and detection tracking process that can use background difference subtraction and motion to detect and track blobs in an image stream. The set of tracked entities is sent to a composition process that tracks targets in 3D scene coordinates.

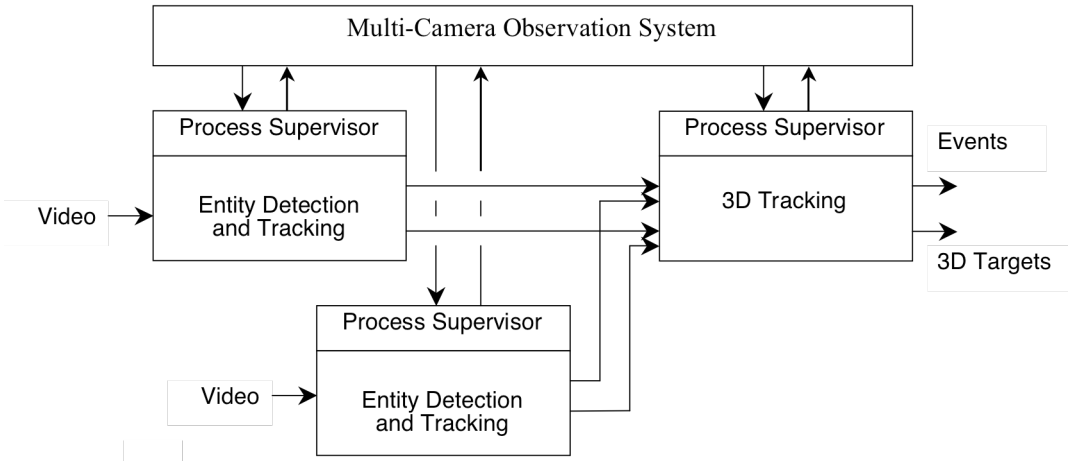


Figure 5. A simple process federation composed of an entity detection process, a composition process and a meta-supervisor.

3. An Architectural Model for Perceptual Processes

Perceptual processes observe a scene in order to measure properties, detect and recognize entities, or detect events. Perceptual processes based on tracking constitute an important class of processes with many applications. Such processes integrate information over time, typically through calculations based on statistical estimation. Tracking processes may be designed for nearly any detection or recognition method, and provide important benefits in terms of robustness of detection and focus of processing resources. In this section, we propose a general architectural model for tracking-based perceptual processes.

3.1 A Model for Tracking Based Perceptual Processes

Our general model for tracking based perceptual processes is shown in figure 6.

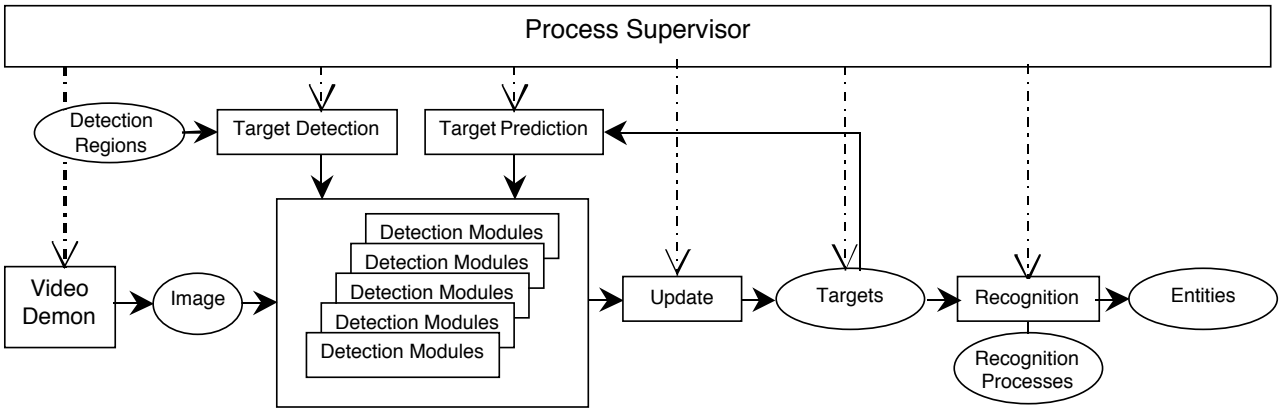


Figure 6. An architectural model for tracking based perceptual processes.

Tracking is a cyclic process of recursive estimation. A well-known framework for such estimation is the Kalman filter. A complete description of the Kalman filter [10] is beyond this paper. A general discussion of the use of the Kalman filter for sensor fusion is given in [11]. The use of the Kalman filter for tracking faces is described in [12].

Tracking is classically composed of four phases: Predict, observe, detect, and update. The prediction phase updates the previously estimated attributes for a set of entities to a value predicted for a specified time. The observation phase applies the prediction to the current data to update the state of each target. The detect phase detects new targets. The update phase updates the list of targets to account for new and lost targets.

To this set of phases, our model adds a recognition phase, an auto-regulation phase, and a communication phase. In the recognition phases, the tracker interprets recognition methods that have been downloaded to the process by a configuration tool. These methods are bits of code that may be expressed in scheme, CLIPS or C++. They are interpreted by a RAVI interpreter [13] and may result in the generation of events or the output to a stream. The auto-regulation phase determines the quality of service metric, such as total cycle time and adapts the list of targets as well as the target parameters to maintain a desired quality. During the communication phase, the supervisor responds to requests from other processes. These requests may ask for descriptions of process state, or capabilities, or may provide specification of new recognition methods.

3.2 Process Supervisory Control

The supervisory component of a process provides five fundamental functions: command interpretation, execution scheduling, event handling, parameter regulation, and reflexive description. The supervisor acts as a programmable interpreter, receiving snippets of code script that determine the composition and nature of the process execution cycle and the manner in which the process reacts to events. The supervisor acts as a scheduler, invoking execution of modules in a synchronous manner. The supervisor handles event dispatching to other processes, and reacts to events from other processes. The supervisor regulates module parameters based on the execution results. Auto-critical reports from modules permit the supervisor to dynamically adapt processing. Finally, the supervisor responds to external queries with a description of the current state and capabilities.

3.2.1 Process Scheduler

The process supervisor maintains a schedule of modules to be executed. The scheduler can interrupt processing after each phases to receive and react to events. Typically this schedule will be composed of phases, with the module calls within each phases determined by a list of data elements. Our tracking architecture uses a schedule composed of the following six phases:

Module Execution Schedule for the Tracker:

1. GetNextImage();
2. For each current target: Predict target location and update target description;
3. For each detection region: If New Target Detected then add target to current target list;
4. For each recognition method in list: execute method();
5. Evaluate quality of result and adapt module schedule and parameters;
6. Interpret messages.

Each target and each detection region contains a specification for the module to be applied, the region over which to apply the module, and the step size to apply processing. Recognition methods are interpreted snippets of code that can generate events or write data to streams. These methods may be downloaded to a robust tracker as part of the configuration process to give a tracking process a specific functionality.

Quality of service metrics such as cycle time, number of targets can be maintained by dropping targets based on a priority assignment or by reducing resolution for processing of some targets (for example based on size).

3.2.2 Homeostasis and Autonomic Control

Homeostasis or "autonomic regulation of internal state" is a fundamental property for robust operation in an uncontrolled environment. A process is auto-regulated when processing is monitored and controlled so as to maintain a certain quality of service. For example, processing time and precision are two important state variables for a tracking process. These two may be traded off against each other. The process supervisor maintains homeostasis by adapting module parameters using the auto-critical reports.

An auto-descriptive controller can provide a symbolic description of its capabilities and state. The description of the capabilities includes both the basic command set of the controller and a set of services that the controller may provide to a more abstract supervisor. Such descriptions are useful for both manual and automatic composition of federations of processes.

During initialization, processes publish a description of their basic functionality and data types in an ontology server, provided by the BIP tool described below. During execution, processes can respond to requests for information about current state, with information such as number and confidence of currently observed targets, current response time, or other quality of service information.

3.2.3 Communication between processes

Three classes of channels exist for communication between processes: events, streams and requests. Events are asynchronous symbolic messages that are communicated through a publish and subscribe mechanism provided by the Federation Supervisor. Streams provided serial high

bandwidth data between two processes. Requests are asynchronous messages that ask for the current values of some process variables.

4. BIP, A Basic Interconnection Protocol for Event Flow Services

BIP stands for Basic Interaction Protocol. BIP is designed to allow low-latency dynamic discovery and assembly of processes for assembling federations of perceptual processes. Perceptual processes and components advertise their abilities to provide services by publishing ontological descriptions in a Multi-cast Dynamic Name Service (DNS) Service Discovery (SD). BIP uses the Apple Rendezvous (DNS-SD) open standard to allow processes to describe and publish their perceptual abilities as software services.

In addition to facilities for publishing and discovery of services offered by perceptual processes, BIP provides primitives for systems level communication and control of software federations based on procedures for event flow services. BIP services exist for both JAVA and C++ and may be used to assemble federations of processes running under Linux, Win32 and Mac OSX.

Processes advertise their abilities by publishing a 64-byte event containing an IP address and port. Perceptual services and data structures are described using an XML protocol. An ontology server is provided organized as an inheritance hierarchy for software and perceptual service classes. This ontology can be searched using XPath to discover compatible services.

This section describes BIP and its use for assembling federations of processes for the CAVIAR project.

4.1 Specifications for the Basic Interconnection Protocol (BIP)

BIP is designed to meet a number of requirements encountered when constructing real time perceptual systems: low latency communication, reliability, function reuse and distribution.

low latency

Real time systems place a constraint on overall latency. For example, steering a camera requires latency on the order of one frame time (40 ms) to maximize usability and avoid instability. BIP is designed to provide very low latency communications.

reliability

Overall system reliability for a process federation depends critically on the communications reliability between processes.

low threshold, high ceiling

Easy integration of BIP services into heterogeneous systems is a requirement. Minimal implementations (under 100 lines of code in any language) should always possible, in order to allow for easy interoperability.

function reuse and distribution

Subsystems may need to be used by multiple other subsystems, possibly running on different machines (for real-time requirements, or for geographic reasons). For instance, a video capture service might be used concurrently by a surveillance monitor located on a centralized server, and a local event detection system.

These requirements have a number of consequences that influence how BIP is designed:

- it is presumed that the systems communicating using BIP are independent processes;
- no assumption is made on the peer operating systems, languages, or method of communication (machine-local or distant);
- peer services must be dynamically discoverable in order to alleviate the need for address-based and port-based setup;
- multiple transports must be available to cope with the multiple latency and reliability needs;
- the protocol must contain a minimal portion that is sufficient to make services useful, while allowing simple programs like telnet to interoperate with any BIP implementation.

The minimal BIP protocol consists in an exchange of ASCII messages with 1-line message headers over a TCP socket. This allows for minimal interoperability with a very low effort. Implementers of BIP can enrich the interconnection by adding support for UDP communications, multicast UDP, DNS-based dynamic discovery, and finally service control through an XML control protocol.

4.2 BIP Service overview

BIP allows processes to advertise their services using the DNS-SD [14] convention over Multicast DNS [15]. They use the local.domain, TCP transport, and the process name. Thus fully qualified names for BIP processes names must be as follows:

```
<name>._
bip._tcp.local.
```

where <name> is chosen by the process instanciator. The TCP port number advertised in the DNS-SD SRV record is the port used for control channel to the process.

The DNS-SD TXT record fields listed below must or may be present for each process. They must not change during the lifetime of a process.

inputs (**required**)

A comma-separated list of inputs names. Each name must only contain roman letters, underscores and digits. This field can be empty.

outputs (**required**)

Same syntax as *inputs*. Describes the process outputs. Cannot be empty.

inoutputs (**required**)

Same syntax as *inputs*. Describes the process' channels used for transactions (excluding format negotiations).

class (**optional**)

The class of process. This will be standardized in a future revision of `BIP`. Classes starting with a period are reserved.

owner (**optional**)

A federation that owns the process. This can be used to limit connections to a cluster of processes, e.g. in the case of concurrently running systems that uses the same process.

Additionally, there must exist a field for each channel name in *inputs*, *outputs*, and *inoutputs*. The field key is the channel name, and the field value is a slash-separated list of 1, 2 or 3 ports. The first port (mandatory) is the channel's TCP port, as a decimal number. The second port (optional) is the channel's UDP port, as a decimal number. The third port (optional) represents the channel's multicast group, formatted an 8-digit hexadecimal number.

4.3 Connecting to a process

There are three types of channels within a process:

- the *control channel* communicates events that allow a peer to inspect a process and control its state variables;
- *input channels* transmit events to one of a process' inputs;
- *output channels* transmit events from one of a process' outputs.

A process accepts a TCP link on one TCP port for control links, and an additional TCP port for each input and output. For inputs, it also accepts messages over UDP if a TCP link has been established

(see below). For outputs, it can emit over UDP if requested. Data that transits over any link with process obeys the message format described below. For the control link, data inside messages obeys specifications given below.

Processes may define any number of parameters. All processes must define at least the following so-called “universal” parameters.

status (**integer**)

Possible values are

. listcnt1

- the process is not running;
- the process was asked to run, but is waiting for one or more inputs to be connected;
- the process is running.

lock (**integer**)

Possible values are

- zero; or
- the ID of a peer connected to the control port.

If lock is non-zero, the process will not modify any parameter upon reception of a control query that originates from a peer whose ID is not equal to the value of this parameter. A control answer event will still be emitted. This parameter is automatically reset to zero if the corresponding peer disconnects.

4.4 Communicating with processes

The communication protocol must not impose any software requirements on the peers that connect to a process. This leads to the following communication protocol.

4.4.1 Low-level message format

Messages are the lowest-level of the protocol. Messages transit over a network connection to a process. They contain a fixed-size (34 bytes) header and a variable-size payload. In order to prevent identifier collisions, a process requires a random number generator. In UNIX platforms, the random number generators in the standard C library should be seeded, for example, with the process startup time.

The format of message contents is not specified for input, inoutput, or output links to a process. However, it is specified for the control connections (see the sections on inspection and control below). It is strongly advised to keep the format of message contents for a particular process as simple as possible, for the sake of inter-operability with other systems and processes. In particular, process designers should:

- avoid using high-level languages except where relevant: XML is pertinent for large tree-like structures, not for transmitting two integers;
- transmit plain ASCII text instead of binary when possible: it will be easier to trace the protocol for debugging purposes;
- use `<cr><lf>` as a line delimiter even inside messages.

4.4.2 Establishing a link

To establish a link to a process, a peer must:

- Open a TCP connection to one of the process's ports (either the control port or one of the input, inoutput, or output ports);
- Send an empty message to the process;
- Receive an empty message from the process.

These two messages allow the peers to identify each other, thank the peer ID provided with each message. The order of steps 2 and 3 is not specified.

A process must accept only one *control* link at a time. A process may choose to only accept one link to any *input*, *output* or *inoutput* channel at a time.

4.4.3 Communicating over TCP

Communicating over TCP is possible for any process link. Once a link has been established as described above, communication is achieved by sending and receiving any number of successive messages. It is recommended that processes that require low-latency communication over TCP send and receive messages (e.g. empty messages) at all times. This is to ensure that the TCP window stays as open as possible, and that occasional "bursts" of data are not limited by the endpoints' TCP stacks. The actual minimal data throughput to maintain depends on you local implementation(s) of TCP; it is however recommended that you maintain an overall average throughput of half your maximal throughput between the two peers.

4.4.4 Communicating over UDP

Communicating over UDP is possible for process input, output and inoutput channels. To communicate with a process over UDP, a peer must first establish a (TCP) link as described above. For *input* and *inoutput* links, the peer may then send messages in UDP datagrams to the UDP port that corresponds to an input immediately after the link has been established. For *output* links, the peer must send and empty message over UDP to the UDP port that corresponds to the output. The process may then send outgoing to the linked peer either to the UDP port that emitted the message, either through the TCP connection. Each UDP datagram must contain exactly one message. A process must ignore incoming UDP messages from a peer if no link is established.

4.4.5 Communicating over multicast UDP

The above paragraph (Communicating over UDP) remains valid for communication over multicast UDP. No protocol is specified when choosing a multicast group. It is recommended to select a group at random.

5. Federation Assembly with BIP.

A process federation is a system of independent cooperating processes. A process federation provides a convenient mechanism to extract and integrate information from a network of sensors and cameras, without the need to communicate large volumes of high-bandwidth data. Federations can also be used to distribute processing over a network of computing devices in an ad-hoc manner in response to changes in operating context.

Deliverable D35 gives details of how to use the BIPTool environment to configure and monitor processes and process federations. This section illustrates the functions that can be provided by this tool.

5.1 An Example of a Process Federation

A simple example of a federation of perceptual processes is provided by a system that uses a distributed network of cameras to maintain a model of the current situation in a parking lot, as shown in figure 7. Targets detected within each robust tracking process are reported to an entity composition process. The composition process assembles targets into composite entities and maintains a global history of the target evolution and trajectory. This system has been used to track vehicles and pedestrians within the INRIA CAVIAR parking lot test-bed.

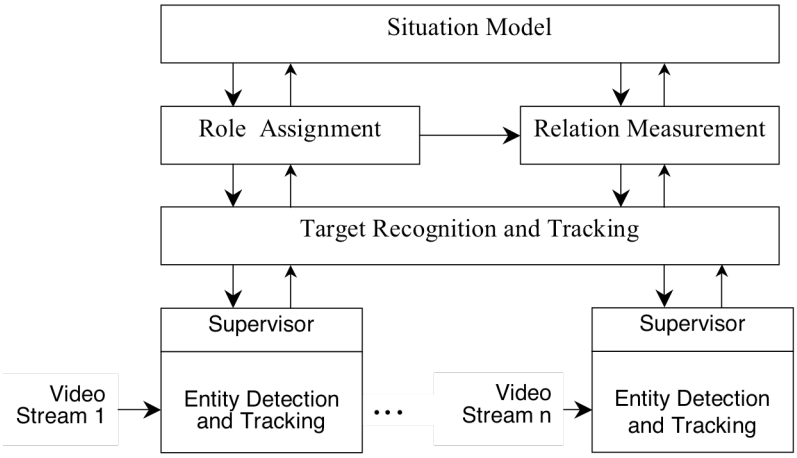


Figure 7. A process federation for situation modeling using a distributed camera network

5.1.1 Specifying Process Federations

Process federations are composed in order to provide observational services. An observational service is defined by

1) A specification of the classes of entities and actions that are to be detected, recognized and tracked,

2) A specification of the predicate functions (relations) that are to be evaluated on the entities

3) A specification of for a network of situations that define the activities to be observed.

4) An initialization script for launching and configuring a process federation.

Specification of the entities and actions translates into a requirement of perceptual services, to be provided by perceptual processes. The initialization script uses information about available camera and software processes to launch and configure a set of perceptual processes. Configuration can include loading snippets of interpretation code that can be used drive interpretation by a process. Each process registers its available functions and data types in the BIP ontology server.

The situation model initiates process federation by launching and configuring the role assignment and relational measurement processes. These, in turn, detect and recruit the available observational processes by establishing links.

These descriptions can be named and saved using an XML format in a process federation directory. A list of previously specified federations is obtained by importing the XML descriptions from this directory. Selecting such a description triggers instantiation and default configuration for this federation.

5.1.3 Monitoring process execution

The BIP tool provides a basic output viewer that allows configuration and execution of processes to be monitored, as well as graphical display windows, to monitor both input data and the current situation. These are adapted from the PFT described in the year 2 deliverable D15.

Bibliography

[1] J. L. Crowley and H. I. Christensen, Vision as Process, Springer Verlag, Heidelberg, 1993.

[2] J Coutaz, J. L. Crowley, S. Dobson, and D. Garlan, "Context is Key", Communications of the ACM, Special issue on the Disappearing Computer, Vol 48, No 3, pp 49-53 March 2005.

[3] Software Process Modeling and Technology, edited by A. Finkelstein, J. Kramer and B. Nuseibeh, Research Studies Press, John Wiley and Sons Inc, 1994.

[4] J. Estublier, P. Y. Cunin, N. Belkhatir, "Architectures for Process Support Interoperability", ICSP5, Chicago, 15-17 June, 1997.

[5] J. L. Crowley, "Integration and Control of Reactive Visual Processes", Robotics and Autonomous Systems, Vol 15, No. 1, december 1995.

[6] M. Shaw and D. Garlan, Software Architecture: Perspectives on an Emerging Disciplines, Prentice Hall, 1996.

[7] J. L. Crowley, "Context Driven Observation of Human Activity", European Symposium on Ambient Intelligence, Amsterdam, 3-5 November 2003.

[8] K. Schwerdt and J. L. Crowley, "Robust Face Tracking using Color", 4th IEEE International Conference on Automatic Face and Gesture Recognition", Grenoble, France, March 2000.

[9] J. Piater and J. Crowley, "Event-based Activity Analysis in Live Video using a Generic Object Tracker", Performance Evaluation for Tracking and Surveillance, PETS-2002, Copenhagen, June 2002.

[10] R. Kalman, "A new approach to Linear Filtering and Prediction Problems", Transactions of the ASME, Series D. J. Basic Eng., Vol 82, 1960.

[11] J. L. Crowley and Y. Demazeau, “Principles and Techniques for Sensor Data Fusion“, Signal Processing, Vol 32 Nos 1-2, p5-27, May 1993.

[12] J. L. Crowley and F. Berard, "Multi-Modal Tracking of Faces for Video Communications", IEEE Conference on Computer Vision and Pattern Recognition, CVPR '97, St. Juan, Puerto Rico, June 1997.

[13] A. Lux, "The Imalab Method for Vision Systems", International Conference on Vision Systems, ICVS-03, Graz, april 2003.

[14] Multicast DNS: performing peer-to-peer DNS queries over IP Multicast. Described at http://www.multicastdns.org/.

[15] DNS-Based Service Discovery: browsing and discovery of services using DNS queries.Described at http://www/dns-sd.org/

[16] J. L. Crowley, J. Coutaz, G. Rey and P. Reignier, "Perceptual Components for Context Aware Computing", UBICOMP 2002, International Conference on Ubiquitous Computing, Goteborg, Sweden, September 2002.