

**An approach for designing a real-time intelligent
distributed surveillance system**

By

Maria Valera Espina

A thesis submitted in partial fulfilment of the
requirements for the degree of

**Doctor of Philosophy
May, Year 2006**

Digital Imaging Research Centre
Faculty of Computing, Information Systems and Mathematics
Kingston University

ABSTRACT

The main aim of this PhD is to investigate how a methodology rooted in systems engineering concepts can be established and applied to the design of distributed wide-area visual surveillance systems. Nowadays, the research community in surveillance systems tends to be mostly focused on the computer vision part of these systems, researching and developing more intelligent algorithms. The integration and finally the creation of the system per se, are usually regarded as a secondary priority. We postulate here that until a robust systems-centred, rather than algorithmic-centred approach is used, the realisation of realistic distributed surveillance systems is unlikely to happen.

The future generation of surveillance systems can be categorised, from a system engineering point of view, as concurrent, distributed, embedded, real time systems. An important aspect of these systems is the inherent temporal diversity (heterogeneous timing) that arises from a variety of timing requirements and from the parallelisation and distribution of the processes that compose the system. Embedded, real-time systems are often naturally asynchronous. However, the computer vision part of these surveillance systems is commonly conceived and designed in a sequential and synchronous manner, in many cases using an object-oriented approach. Moreover, to cope with the distributed nature of these systems, technologies such as CORBA are applied. Designing processes in a synchronous manner plus the run-time overheads associated with object oriented implementations may cause communication bottlenecks. Perhaps more importantly, it may produce unpredictable behaviour of some components of the system and hence undetermined performance from a system as a whole. Clearly, this is a major problem on surveillance systems that can often be expected to be safety-critical.

This research has explored the use of an alternative approach to object-orientation for the design and implementation of intelligent distributed surveillance systems. The approach is known as Real-Time Networks (exemplified by system engineering methodologies such as MASCOT and extensions such as DORIS). This approach is based conceptually on conceiving solutions as being naturally concurrent, from the highest level of abstraction, with concurrent activities communicating through well-

defined data-centred mechanisms. The methodology favours a disciplined approach to design, which yields a modular structure that has close correspondence between functional elements in design and constructional elements for system integration. It is such characteristics that we believe will become essential in overcoming the complexities of going from small-scale computer vision prototypes to large-scale working systems.

To justify the selection of this methodology, an overview of different software approach methods that may be used for designing wide-area intelligent surveillance systems is given. This is then, narrowed down to a comparison between Real-Time Networks and Object Orientation. The comparison is followed by an illustration of two different design solutions of an existing real-time distributed surveillance system called ADVISOR. One of the design solutions, based on Object Oriented concepts, uses CORBA as a means for the integration and distribution characteristics of the system. The other design solution, based on Real-Time Networks, uses DORIS methodology as a solution for the design of the system. Once the justification over the selection is done, a novel design of a generic visual surveillance system using the proposed Real-Time Networks method is presented. Finally, the conclusions and future work are explained in the last chapter.

Acknowledgements

I would like to honour many people who have kindly contributed to this PhD study over the course of its development at Kingston University. I would like to express my thorough gratitude to my supervisor, Dr. Sergio A. Velastin for his support and assistance, especially during this last year. I appreciate the discussions, comments and suggestions, which have helped me to present this work. I am also pleased to express my sincere appreciation to Professor Hugo Simpson for his time, the constant supply of documents and for the lively discussions that we have had during this Ph.D. study that have formed the basis of my academic arguments.

I would also like to express my gratitude to: Mr. David Fraser, Professor Anthony Davies and Dr. James Orwell for their valuable comments, suggestions and criticism. I would also like to thank the United Kingdom's Engineering and Physical Sciences Research Council (EPSRC) for the financial support to this Ph.D. study and the researchers from the COHERENT project, especially Dr. Ian Clark. I also would like to express my sincere gratitude to Mrs Bee-Lian Tang, member of staff at Kingston University and Dr. A. Souheil Khaddaj for their help and courtesy.

Finally, I would like to express my deep gratitude to all my friends and family for the support, understanding, comfort and encouragement which they have offered me during the course of this study. They were always there when I needed them, without their support I would have been unable to finish, and so this work is part of them and I cannot end this acknowledgement without mentioning every single one of them. Thanks to every member of my wonderful family and especially to my brother-in-law, José for their invaluable support. I would also like to thank my friends: Oriol, Marta, Susana, Vasco, Joel, Daniel, Robert, Anil, Abi, Yan and the Adamson's family. I would especially like to thank Jelena, Maria, M^a Carmen, Cristina, Annabel and Carme Tello for their help and support during the worst periods of time. I would also like to thank Manel, Paresh and Dr. Boghossian for their constructive advice, comfort and encouragement. Thanks to all of you.

*To my loving parents, sister and brother
To my country, Catalunya*

Table of contents

1	Introduction.....	1
1.1	Motivation.....	1
1.2	Context of the research	3
1.2.1	Distributed systems.....	3
1.2.2	Real-Time systems.....	9
1.2.3	Asynchronous and Synchronous systems	10
1.3	Aim and original contributions	15
1.4	Structure of the thesis	16
2	The state of art of Intelligent Surveillance Systems	18
2.1	Introduction.....	18
2.2	Vision Systems	18
2.3	Evolution of Intelligent Surveillance Systems.....	19
2.4	Requirements of an Intelligent Surveillance System.....	22
2.4.1	Surveillance system requirements for transit applications	23
2.4.2	Surveillance system requirements for port applications	24
2.4.3	Metro and Railway Stations surveillance system requirements	25
2.5	State-of-art in the design of visual surveillance systems	27
2.5.1	Processing components in surveillance systems.....	27
2.6	Examples of surveillance systems	33
2.6.1	Commercial surveillance system for outdoor applications.....	33
2.6.2	Surveillance systems for parking lots applications.....	34
2.6.3	Surveillance systems for traffic control application	36
2.6.4	Surveillance system for port applications.....	37
2.6.5	Surveillance systems for public transport applications.....	38
2.6.6	Multi-camera surveillance system	40
2.6.7	Co-operative camera systems	41
2.7	Distribution and communication	44
2.8	Summary	46
3	Design methodologies for real-time distributed intelligent surveillance systems	49
3.1	Introduction.....	49
3.2	Design methods.....	50
3.2.1	Design methods in surveillance systems	51
3.2.2	Classification by structural principles.....	52
3.2.3	Current research in design methods.....	53
3.2.4	General criteria for comparing design methods.....	55
3.3	Traditional design methods for real-time systems	55
3.3.1	The Yourdon Structure Method (YSM).....	56
3.3.2	Jackson System Development (JSD)	57
3.3.3	NRL	58
3.3.4	ADARTS	60
3.4	Survey of some important OO design methods	61
3.4.1	ROOM	62
3.4.2	BOOCH	64
3.4.3	Rumbaugh (OMT)	66
3.4.4	HOOD.....	67
3.4.5	UML (Unified Modelling Language)	70

3.5	Real Time Networks (RTN)	75
3.5.1	MASCOT- RTN principles.....	76
3.5.2	The MASCOT network design.....	77
3.5.3	MASCOT communication model.....	80
3.6	DORIS- further extension of RTN principles.....	84
3.6.1	Definition: COntrolled Requirements Expression (CORE).....	86
3.6.2	Design: MASCOT	87
3.6.3	Implementation: DIA.....	89
3.6.4	Communication mechanisms.....	89
3.7	Comparison between the OO and the MASCOT/DORIS approaches	95
3.7.1	The difference of abstract model between the two approaches	96
3.7.2	Communications	97
3.7.3	Concurrency and Information hiding.....	102
3.7.4	Inheritance	104
3.7.5	Polymorphism and dynamic binding	106
3.7.6	Performance.....	107
3.8	Summary.....	109
4	Case study: ADVISOR	112
4.1	Introduction.....	112
4.2	CORBA (Common Object Request Broker Architecture)	114
4.2.1	CORBA components	115
4.2.2	CORBA features	117
4.3	ADVISOR.....	120
4.3.1	Specifications of the ADVISOR system.....	120
4.3.2	Specifications that ADVISOR Prototype did not accomplish	122
4.3.3	ADVISOR system architecture design	122
4.3.4	Data types in the system and communication between modules....	130
4.4	The CORBA architecture design implemented in ADVISOR Prototype using DORIS graphical notation.....	132
4.4.1	ORB and COM subsystem.....	136
4.5	The ADVISOR Prototype architecture using CORBA platform technology in DORIS notation	139
4.5.1	The decomposition of ADVISOR architecture design	140
4.5.2	First level of the Architecture design: ADVISORSYSTEM	146
4.5.3	Second level of the Architecture design: HCICENTRAL.....	147
4.5.4	Second level of the Architecture design: LOCALDATAPROCESSING.....	148
4.5.5	Third level of the Architecture design: CORBA_SUBSYSTEM, HUMANINTERFACE and ASUSUBSYSTEM.....	151
4.6	The ADVISOR Prototype architecture using the DORIS method and its concepts	163
4.6.1	First level of the architecture design: ADVISORSYSTEM	164
4.6.2	Second level of the architecture design: COMMUNICATION	165
4.6.3	Second level of the architecture design: HCICENTRAL.....	166
4.6.4	Second level of the architecture design: HCINODE	167
4.6.5	Third level of the architecture design: CROWD_MONITOR.....	168
4.7	Comparison between the two architectures	169
4.7.1	Communication techniques.....	170
4.7.2	Concurrency and distribution.....	175
4.7.3	Run-time	177

4.7.4	Development aspects	179
4.8	Summary	180
5	Design of a Real-Time Distributed Surveillance System with multiple cameras	182
5.1	Introduction.....	182
5.2	First level of the system design.....	183
5.2.1	Functional definition of the system	183
5.2.2	RTN components used in the design	188
5.3	Functional description of different parts of the design	190
5.3.1	Classification of data used in the system	191
5.3.2	Monitoring part of the design	202
5.3.3	Data processing part of the design	207
5.3.4	Feedback part of the design	209
5.3.5	Control part of the design	212
5.4	Design of the system.....	219
5.4.1	Partitioning.....	222
5.5	Network Design of the system.....	223
5.5.1	Logical design topology.....	224
5.5.2	Traffic behaviour- Multicasting.....	228
5.6	Quality of Service (QoS)	229
5.6.1	Bandwidth.....	230
5.6.2	Resource management- Scheduling.....	231
5.7	Summary	233
6	Conclusions and Future work	235
6.1	Introduction.....	235
6.2	Conclusions.....	235
6.2.1	How this research linked to COHERENT	236
6.2.2	Design methodologies.....	237
6.2.3	RTN/CORBA for designing real-time distributed surveillance systems	240
6.3	Future work.....	244
6.3.1	Framework for designing real-time distributed surveillance systems	244
6.3.2	Inclusion of Formal Methods to the framework	249

List of Figures

Figure 1-1. Message communication between distributed entities. Tasks a and c from node 1 communicate between them and with tasks d, e from node 2.....	5
Figure 1-2. Remote procedure mechanism. The process is illustrated from step (1) to (6).....	6
Figure 3-1. Simple communication model between two activities (producer, consumer) through an IDA component.	81
Figure 3-2. Basic protocol taxonomy refers to effects on data from read and write operations.....	83
Figure 3-3. DORIS development process from [Simpson 1994e].....	86
Figure 3-4. The Extended communications of <i>Route</i> protocols [Simpson 2003f]. ..	91
Figure 3-5. Example of the stretched form of the channel protocol [Simpson 1994e], [Simpson 2003f].	95
Figure 3-6. Distributed model of remote function call [Simpson 2003f].	95
Figure 3-7. Example of different approaches to the communication model; in OO objects communicate to objects. In RTN communication is from/to activity to/from IDA.	101
Figure 4-1. The states of a CORBA object and servant object life cycle [Henning and Vinoski 1999].....	117
Figure 4-2. Common Object Request Broker Architecture (CORBA)[Henning and Vinoski 1999].	117
Figure 4-3. The flow of requests to the server side and how POA dispatches them.	119
Figure 4-4. Logical view of ADVISOR system.	124
Figure 4-5. The logical view of the ADVISOR Prototype (tested at Barcelona). ..	125
Figure 4-6. Top level design of ADVISOR System Unit (ASU). Note that, the ADVISOR Prototype consists of one HCI that is connected to one ASU.....	127
Figure 4-7. Top level design of Image Processing Unit (IPU).	128
Figure 4-8. Top-level design Symbol Processing Unit (SPU).....	130

Figure 4-9. Links between the CORBA features illustrated in Figure 4-2 with the CORBA design of ADVISOR.133

Figure 4-10. The CORBA design of ADVISOR using DORIS graphical notation. ASU_ SUBSYSTEM (Appendix C-14, pp. 298), COM (Appendix C-13, pp. 297), CORBA_ SUBSYSTEM (Appendix C-15, pp. 299), IMPLREPOSITORYSUBS (Appendix C-22, pp. 306), CONFI_ PARAMETERS (Appendix C-21, pp. 305), CROWD_ MONITOR (Appendix C-20, pp.304).135

Figure 4-11. The configuration of COM subsystem (Appendix C-13, pp. 297). See Appendix C for more detailed view of each of the components that appear in this figure: COMSUBSYSTEM (Appendix C-19, pp. 303), MULTIDISTRIBUTION (Appendix C-17, pp. 301), DISTRIBUTIGNAL (Appendix C-18, pp.302)......137

Figure 4-12. The whole path of one of the signals coming from CORBA_ SUBSYSTEM to ASUSUBSYSTEM. This signal comes from OA_OUT inside CORBA_ SUBSYSTEM (see interface called Sync_put) to one of the objects inside the ASUSUBSYSTEM (see interface called Sync_get).139

Figure 4-13. The links between the modules used below to express the architecture design of ADVISOR Prototype using MAGDE tool. Moreover, this figure illustrates the differences between the ADVISOR system and the ADVISOR Prototype (see also Figure 4-4 and Figure 4-5).142

Figure 4-14. Most of the levels of decomposition of the ADVISOR Prototype system. The figure also illustrate the modules that represent the CORBA solution for ADVISOR Prototype. All the subsystems that are represented in this figure are described in more detail in the main text. All the subsystems can also be found, in clearer separate diagrams, in Appendix C. The bidirectional arrows illustrate the data communication flow between modules of different levels. Bear in mind, that some modules have been introduced due to tool restrictions even though they are not needed to model CORBA (e.g. COMUHCISUBSYSTEM in Appendix C-4, pp. 288).144

Figure 4-15. ADVISOR Prototype system using DORIS notation. This figure represents the first level of ADVISORSYSTEM. This ADVISORSYSTEM consists of three human interface subsystems and one central human interface subsystem.146

Figure 4-16. The second level of decomposition of the ADVISOR Prototype system. It represents the internal composition of the HCICENTRAL subsystem. HCICENTRAL deals with the control signals coming from the central control user.148

Figure 4-17. Three levels of decomposition of the ADVISOR Prototype system (following the hierarchical DORIS notation). It starts with the internal composition of LOCALDATAPROCESSING (Appendix C-3, pp. 287) subsystem, which is presented in the second level of this hierarchical structure. This is followed by the decomposition of DATAPROCESSINGNODE (Appendix C-5, pp. 289), which corresponds to the third level and it finishes with the decomposition of

LOCALASU (Appendix C-10, pp.294) and LOCALHCI- STATION (Appendix C-11, pp. 295) that are the third level of the hierarchical structure.150

Figure 4-18. The three subsystems (CORBA_SUBSYSTEM, HUMANINTERFACE and ASUSUBSYSTEM), which belong to the third level of the ADVISOR Prototype system design decomposition.152

Figure 4-19. Following the hierarchical MASCOT/DORIS notation, CORBA_SUBSYSTEM represents the fifth level of the ADVISOR Prototype system. CORBA_SUBSYSTEM subsystem illustrates the communication functionality of an ORB in CORBA technology.155

Figure 4-20. CONF_PARAMETERS (the sixth level of the ADVISOR Prototype decomposition) is introduced in the fifth level of the hierarchical structure in CORBA_SUBSYSTEM. CONF_PARAMETERS gives the configuration parameters required for other components outside the subsystem.156

Figure 4-21. HUMANINTERFACE subsystem presented in the third level of the hierarchical structure of the ADVISOR Prototype system. HUMANINTERFACE is introduced in LOCALHCISTATION subsystem. HUMANINTERFACE deals with the control signals coming from the HCICENTRAL and it also deals with local control signals coming from the user in the local HCI subsystem.157

Figure 4-22. ASUSUBSYSTEM is presented in the third level of the decomposition of ADVISOR Prototype system. ASUSUBSYSTEM is composed of image processing subsystems.159

Figure 4-23. The CROWD_MONITOR subsystem (the fourth level of ADVISOR Prototype system). It performs an image processing task detecting crowds.159

Figure 4-24. The MONITOR subsystem (the fifth level of ADVISOR Prototype system) deals with control signal coming from upper levels and petitions from lower levels.160

Figure 4-25. DEVICE (fifth level of the decomposition of the ADVISOR Prototype system) is a subsystem that deals with the control signals coming from upper levels. It also deals with the signals of a subsystem where the low-level image processing tasks are carried out.161

Figure 4-26. CORBA_SUB subsystem (fifth level of decomposition of the ADVISOR Prototype system) illustrates the communication functionality of an ORB in CORBA technology like in CORBA_SUBSYSTEM.162

Figure 4-27. The IMAGEPROCESSING (the fifth level of decomposition the ADVISOR Prototype system) subsystem, which performs the image processing tasks.163

Figure 4-28. This figure represents the ADVISOR system using RTN concepts. ADVISORSUBSYSTEM (Appendix C-33, pp. 317) represents the first level of the

design system. The system is composed of four computing nodes and one communication element.....	165
Figure 4-29. COMMUNICATION (Appendix C-35, pp. 319) composite IDA (second level of decomposition of the ADVISOR Prototype system), is shown in the first level of the hierarchical structure of ADVISOR. This communication element links the HCICENTRAL subsystem with each HCINODE.....	166
Figure 4-30. The HCICENTRAL (Appendix C-34, pp. 318) subsystem purely displays, if a user requires, data coming from any HCINODE. It also deals with control signals coming from any HCINODE.	167
Figure 4-31. The HCINODE (Appendix C-36, pp. 320) is the second level of the hierarchical structure of the ADVISOR Prototype system. It is composed of six subsystems, which communicate between them through IDAs: channels, pools and signals.	168
Figure 4-32. The CROWD_MONITOR (Appendix C-37, pp. 321) represents the third level of the hierarchical structure of the ADVISOR Prototype system. It is composed of two activities and two servers which communicate between them through a channel, a pool and a signal.....	169
Figure 4-33. Comparison at a first level of design of ADVISOR Prototype system using the two approaches.....	170
Figure 4-34. Comparison of LOCALHCISTATION subsystem designs using the two approaches.	173
Figure 4-35. Comparison of CROWD_MONITOR subsystem designed using both approaches.	175
Figure 5-1. Functional representation of the system.....	187
Figure 5-2. Local ARchive (LAR) component. Note that in Appendix D this subsystem the signal <i>Trigger</i> does not appear.	193
Figure 5-3. The design of LOCAL_DPU_info_MODULE.....	195
Figure 5-4. Decomposition of DPU_info_MODULE subsystem.....	196
Figure 5-5. Design of CCo_info_MODULE.	198
Figure 5-6.Design decomposition of CC1_info_MODULE.....	200
Figure 5-7. Composition of the subsystem called Locator_user.	201
Figure 5-8. The design of the subsystem called VISUAL.....	203
Figure 5-9. The subsystem called Visual_CC. This subsystem is the visual subsystem for any CC1 node.	205

Figure 5-10. The design of a subsystem that represents a visual subsystem for any mobile user.....	207
Figure 5-11. DPU subsystem. The sensors are attached to this subsystem through the CA subsystem.	208
Figure 5-12. The ALARM FEEDBACK CONTROL subsystem that represents a feedback part of the system.	211
Figure 5-13. The ALARMS FEEDBACK CONTROL CC1 subsystem that represents a feedback part of the system.	212
Figure 5-14. Design of the config_module used by the CCo to change configuration parameters.	214
Figure 5-15. The CC0 subsystem, which represents the first level of hierarchical control structure of the system.....	215
Figure 5-16. The CC1 subsystem, which represents the second level of the hierarchical structure of the system.	217
Figure 5-17. The CC2 subsystem design if the system was scaled one more level.	218
Figure 5-18. The changes that should be applied to CC1 (shown in red) if a CC2 subsystem is introduced in the design of the system.	219
Figure 5-19. Design represented using MADGE tools in Appendix D.....	221
Figure 5-20. Two previous candidate topologies of the system design, before the final topology.....	226
Figure 5-21. The final logical design network topology of the system.	228
Figure 6-1. Topological network view of the generic 3GSS presented in chapter 5.	247

List of Tables

Table 2-1. Summary of the technical evolution of intelligent surveillance systems (from Valera and Velastin 2005b).....	21
Table 3-1. Different software tools used in computer vision. This table is extracted from a Summer school in computer vision at Surrey University [Summer School 2004] and shows the wide range of tools currently available.....	52
Table 3-2. Summary of the aspects to compare at the conceptual model.....	96
Table 4-1. The concepts that will be compared between CORBA and MASCOT3/DORIS.....	114
Table 4-2. The following figures in their respective level are indexed, assigning a level number to each figure to assist in following the hierarchical designs. For a clearer representation of the figures see Appendix C.....	145
Table 4-3. The following figures in their respective level are indexed, assigning a number level to clarify the hierarchical designs.	164
Table 5-1. Summary of some of the RTN components that appear in the proposed design system.	188
Table 6-1. Summary of advantages and disadvantages using CORBA and RTN/DORIS.	242

1 Introduction

1.1 *Motivation*

This research project was carried out as part of an EPSRC¹-funded project referred to as Computational Heterogeneously Timed Networks (COHERENT). The aim of COHERENT was to model, design and verify embedded real-time systems on-chip systems (SoCs) with heterogeneous timing in order to improve timing and energy efficiency of systems with potential applications in control and image processing. As suggested in [COHERENT2005], the proposed hardware-oriented architecture called real-time network on a chip (RTNoC) should consist of computational units of diversity processing and response rates and communication components from (a finite set) of generic Asynchronous Communication Mechanisms (ACMs). COHERENT based the investigation on ACMs and asynchronous techniques to design and verify such systems rather than improving the performance of the computational units that constitute the system. Within that general context, the work reported here, investigated how potentially large scale distributed real-time visual surveillance systems might benefit from design and implementation techniques derived for asynchronous systems.

The technological evolution of vision surveillance systems starts with video-based surveillance systems consisting of analogue Closed Circuit TeleVision (CCTV) systems, i.e. a number of cameras connected to a smaller number of monitors through switches. The technological improvement of these systems led to the development of semi-automatic systems. These systems are able, separately for one or more cameras, to attract the attention of a human operator by detecting unusual conditions and raising an alarm. Current research is towards the design of large-scale automatic surveillance systems. The usual design challenge for these advanced vision systems is to distribute sensors over geographically wide areas. This distribution, from the computational point of view, consists of distributing the processing capacities over the computer network and the use of embedded signal processing devices.

¹ United Kingdom's Engineering and Physical Sciences Research Council

Such surveillance systems can be categorised as concurrent, distributed, embedded, real time systems. An important aspect of these systems is their inherent temporal diversity (heterogeneous timing), arising from the variety of timing requirements from different response times and processing rates of the functional elements of these systems, and from the parallelisation and distribution in the implementation architectures. Moreover, embedded, real-time systems are often naturally asynchronous. Nevertheless, currently the computer vision part of these systems is largely designed in a sequential and synchronous manner using an object-oriented approach. Furthermore, Common Object Request Broker Architecture (CORBA) technology is the mechanism commonly used to deal with the integration and distribution of different parts that constitute the system. The design of these systems in a synchronous manner and the run-time overhead, that object oriented and CORBA approaches might produce, may cause exhaustion of resources caused by unpredictable behaviour of some components of the system.

Moreover, forcing such systems to operate on a synchronous or semi synchronous manner when, as mentioned, they are often naturally asynchronous, might cause other important limitations at different levels. For example, at a network-level these limitations may reduce communications performance (i.e. bottlenecks) while at a chip level these limitations may increase manufacturing costs and reduce the effectiveness of the system in terms of speed. Currently, there are many ways to deal with these problems at both levels. For example, at a chip level, a possible hardware solution relies on distributing the computation between several processing elements (distributed System on Chip).

Apart from developing a general hardware-oriented architecture, the aim of COHERENT project was also to develop a design methodology which would enable the solution of “distributed SoC (System on Chip)” to be more robust and more widely applicable, enhancing its advantages whilst eliminating some of its limitations. This methodology was expected to incorporate asynchronism throughout a full spectrum, from fully synchronised to fully asynchronous, in processing and data communication aspects using heterogeneous timing. Therefore, the aim of this PhD, within the context of COHERENT, was to investigate and appraise the use of Modular Approach to Software Construction Operation and Test

(MASCOT), which is a design method based on the Real Time Network (RTN) principles, and also to investigate the use of ACMs provided by RTN, to the design of generic wide-area visual surveillance systems.

1.2 Context of the research

Chapter 1 and chapter 2 present the context and background of this research. This research project has been carried out inside a research group called the Digital Imaging Research Centre (DIRC), which is concerned with computer vision solutions. The leading research activity and perhaps the most challenging one in this group, is visual surveillance in widespread geographical systems. Traditionally the effort has been concentrated on specific computer vision algorithms for one video source. More recently, due to the advance in the power and sophistication of computer vision algorithms, the research activities are focusing on issues such as the tracking of people in a small network of video sources including conditions where people move outside the field of view of one camera and into the field of view of a neighbouring one. Therefore, it has become useful to consider if it is feasible to deploy these algorithms in real large systems. At this point, the lack of a simple and powerful way of designing and implementing large distributed vision systems became evident and this project, linked with the COHERENT project, sought to address that question. Thus, chapter 1 presents within the context of COHERENT, the background of three main fields: distributed systems, real-time systems and asynchronous design and communication techniques. The background of this research is continued in chapter 2 by presenting the literature review within the context of surveillance systems.

1.2.1 Distributed systems

A common distributed processing environment is constituted by several “nodes” that are interconnected forming a network and they communicate and coordinate their actions by passing messages². These “nodes” may consist of one or more processors sharing memory. The “logical node” also called subsystem, can be

² Message passing is a form of communication used in concurrent, parallel and object-oriented programming. It is also used in interprocess communication. Communication is made by the sending of messages.

defined as groups of concurrent executing tasks, which can be allocated in a same or in a different “physical node” [Gomaa 1993c]. An important design decision is to develop these subsystems in ways that minimise the number of interactions between subsystems (low coupling) and maximise the degree of interaction within the subsystem (high cohesion). If this design decision can be achieved, then an individual subsystem may be designed, coded and tested mostly in a standalone manner. Another beneficial effect of this design decision is that when an error occurs in a subsystem, the spread of damage to other subsystems may be limited. Once subsystems have been designed, the communication between parts is done by sending messages through the network, which implies that even though they should synchronise through signals to perform such communications, there is no single global notion of the correct time [Coulouris et al. 2001]. Thus, the characteristics that may define general distributed systems may be summarised as: concurrency between components that constitute the system, the lack of a global clock and some resilience to component failure. In the following subsections, approaches to distribution and integration of systems are presented.

1.2.1.1 Distributed Kernel

In distributed computing, a common assumption is that when a task sends a message to some other task it should not need to know where this task is situated, making the message communication transparent [Gomaa 1993c]. Some commercial operating systems (e.g. VAX/ELN) provided a distributed kernel, which directly supports this transparency in the message communication. If this property is not available then a Distributed Task Manager (DTM) is usually developed to provide this transparency. The DTM is a layer of software that stands above each operating system on each node. See Figure 1-1.

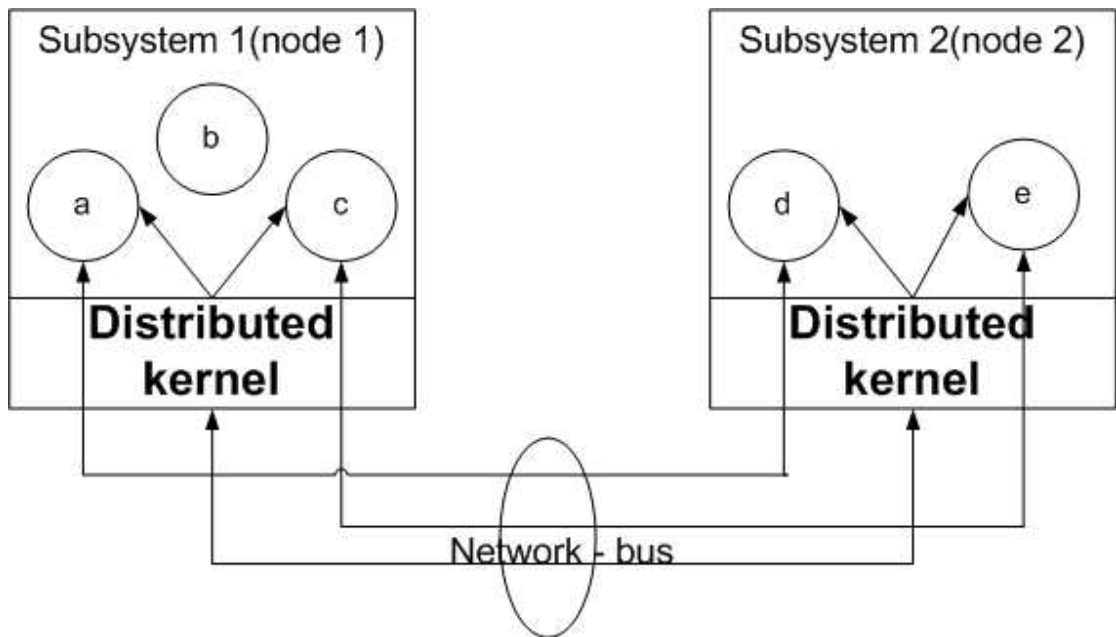


Figure 1-1. Message communication between distributed entities. Tasks a and c from node 1 communicate between them and with tasks d, e from node 2.

1.2.1.2 Message Passing Interface (MPI)

As claimed in [MPI 2003a], MPI technology tends to provide an efficient and portable standard for message passing communication programs used in distributed memory and parallel computing. It is also a specification (standard) for Message Passing Libraries³. The target platforms are systems which consist of massive parallel computing (the programmer is responsible for identifying the parallelism) such as workstation clusters or heterogeneous networks. There are currently several MPI implementations such as MPI/Pro, IBM MPI, and LAM. It is stated in [MPI 2006b] that these implementations provide different communication modes such as asynchronous communication, virtual topologies and efficient message buffer management.

1.2.1.3 Remote Procedure Call

Another technology that has been used to provide the communication in distributed systems is that of Remote Procedure Calls (RPC). This technology is based on a client-server model (local procedure call) where the client subsystem makes a request or “call” to the server subsystem and waits for the answer. In RPC the server subsystem is in a remote node hidden from the client subsystem. The procedure in the client subsystem is often called the client stub, and it handles the

³ [...] refers to a collection of routines which are embedded in application code to accomplish send, receive and other message passing operations [MPI 2003].

request with any relevant parameters, encapsulates them in a message and sends it to the server subsystem. The server procedure called server stub unpacks the message and calls the appropriate procedure to process the call. Once the request has been processed the server stub packs the results in a response message and sends them back to the client. The client stub unpacks the message and sends the results as output parameters to the clients. Thus the functions of the client and server stubs are to make the remote procedure call look like a local procedure call. See Figure 1-2.

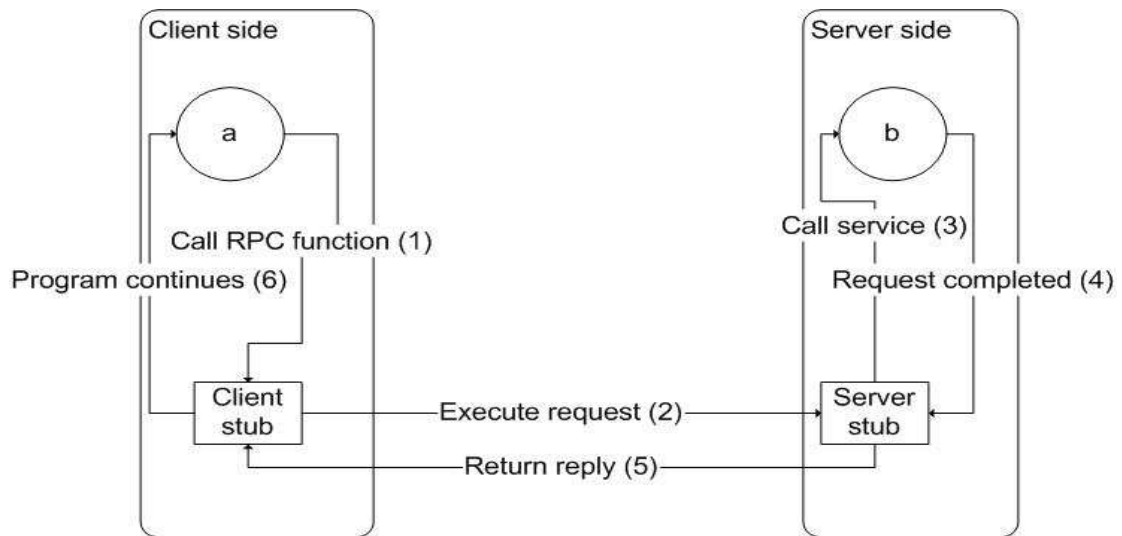


Figure 1-2. Remote procedure mechanism. The process is illustrated from step (1) to (6).

1.2.1.4 Sockets

A socket technology is an end-pair communication model between two processes across a network following a client-server communication model like that of the RPC. The client initiates the rendezvous communication by sending a connexion request to the server machine's port. If the server accepts the request, the connection creates another socket, which is bound to a new port, to connect with the client. Therefore, the initial socket remains free to listen for new connection requests from other clients. Socket technology allows creating software packages like SocketPro [Yuancai 2002] to design the communication between the client and server process to operate in a non-blocking mode, thus allowing the client and server to carry on with their own processing tasks while they are communicating.

1.2.1.5 Middleware technology

One of the recent research areas in distributed systems is the use of technologies referred to as Middleware that are applied to facilitate and manage the communication between nodes and also to allow different platforms (operating

systems) to be integrated in a distributed subsystem. Middleware is a layer of software between the network and the application, which provides services such as identification, authorization, directories and security. The philosophy of these technologies is similar to that of the distributed kernel. There are different types of middleware depending of the technology applied or the application system required [Carnegie Mellon Software Engineering Institute 2005]:

- **Object Oriented Middleware (OOM):** The most popular middleware model. It extends the object oriented paradigm to distributed systems. The applications are potentially distributed objects that interact through a transparent method similar to RPC, but with the difference that in OOM instances of objects can be returned from remote call. Examples of OOM technology include Distributed Computing Environment (DCE), Common Object Request Broker Architecture (CORBA), Microsoft's Common Object Model (COM) and Java Remote Method Invocation (RMI).
- **Message-Oriented Middleware (MOM):** Unlike RPC or OOM this middleware is based on asynchronous communications, thus the producer is not blocked waiting for the consumer to receive the message. Even though the caller and the receiver are loosely coupled, messages are addressed to their recipients and it can be disadvantageous in wide-area distributed systems for the overhead that it generates. The development of e.g. publisher-subscribers systems is a possible solution to decouple producer and consumer from the naming property. Publishers publish to the entire network and subscribers subscribe to the message.
- **Event-Based Middleware:** Refers to technology that is applied to systems that must react to events that can represent changes in the environment or process status. The request-reply paradigm that is commonly used in OOM is not suitable for this kind of system. Therefore, the communication pattern established in this middleware is based on a one-way or loosely coupled communication mechanism similar to MOM.
- **Reflexive Middleware:** Refers to technology that tries to include a "reflection" property in the middleware to achieve openness, configurability and reconfigurability. Reflection understood to be the

capacity of an entity to reason about and act upon itself, a reflective system contains a representation of its own behaviour and it is capable of change, therefore all changes made to the system's self-representation are immediately reflected [Middleware 2005].

Another current research area in distributed systems is based on the use of a called Component technology (conceptually similar to OOM). This technology considers a component entity as the fundamental building block of any application. CORBA technology and COM may also be considered Component technology. Other alternative Component technologies may be JavaBeans [JavaBeans 2006] and .NET [Microsoft .Net 2006] in a platform dependant application. JavaBeans is a component technology easy to integrate in java environments (java platform). .NET is a component-oriented development that replaces COM technology; it allows the creation of components more easily than COM..NET also allows greater interoperability than COM. Although it allows language independency it is still platform dependant (Microsoft technology).

1.2.1.6 Message communication by ports

In some distributed systems communication is based on a loosely-coupled communication pattern between source and sink by means of ports. Tasks are attached to ports, therefore the producer task does not send a message to an explicit consumer but sends the message to the output of its port and, consequently, the consumer does not need to know who the producer is. This communication model contributes to a higher degree of flexibility in the design due to the decoupling in communication between tasks, and contributes to the possibility of re-use since tasks do not need to know who and where the consumers or producers are when there are designed. Some Architectural Description Languages (ADLs) use this communication model to define their architectural designs [Medvidovic and Taylor 2000]. In successive versions of MASCOT and further extensions of RTN such as the Data Oriented Requirements Implementation Scheme (DORIS), the communication model, which will be explained in chapter 3, is based not only on ports but also on what are called windows, paths and Intercommunication Data Area (IDAs). In MASCOT there are two basic types of components: the activity component, which is concerned with information processing and the passive

component (IDA), which is concerned with information storage and transmission. Activities communicate through IDAs, which provide the necessary synchronisation, mutual exclusion and cross-stimulation facilities through appropriate access procedures.

1.2.2 Real-Time systems

In terms of computational timing, a Real Time System (RTS) not only has to produce its results but must produce the results within specified time intervals (response-time constraints) [Phillip 1996], [Naedele 2001]. “what is predictability for RTS” introduced in [Stankovic and Ramamritham 1990, pp.247], is an interesting question because the answer permits linking the predictability of RTS (in terms of timing requirements) with the underlying assumptions. The following list presents the definition of four important characteristics of any RTS. Thus, depending on these characteristics, the design of a RTS may vary significantly:

- Granularity of the deadlines: in RTS some tasks have deadlines and/or periodic timing constraints. For example, when a task is executed and the period of execution must be short, the task has a tight deadline, which means that the operating system has to react promptly. Therefore, the scheduling algorithm should be fast and simple.
- How strict are these deadlines? This can depend on the RTS and the application of it. There are some tasks that can be classified as soft real time tasks. These tasks are defined as tasks that still could be executed when the deadline is passed. Hard real time or critical tasks are the ones that should be executed before the deadline is passed otherwise they may cause major problems, e.g., in a safety critical system to miss a deadline of a critical task may provoke a loss of life.
- Size of the system and the degree of co-ordination: RTS vary considerably in size and consequently in complexity. For example, increasing the size and the degree of co-ordination between tasks may complicate the notion of predictability. Therefore, the ability to load entire systems into memory and to limit task interactions simplifies many aspects of building and analysing

RTS. However, dynamic RTS with fully resident code and highly independent tasks may not always be practical.

- **Environment:** the environment in which the RTS operates plays an important role on the design step. In small and well defined systems (e.g. a lab experiment), from the point of view of a designer, it is possible to think of these systems as deterministic even though they may not be intrinsically deterministic. For example, in hard real time or critical tasks, it is desirable to force the system to be fully deterministic, in the sense that it is imperative that the system fulfils all the timing constraints.

A common approach used to force a complex and distributed system to be deterministic is taken by imposing these systems to work in a synchronous manner. From a circuit design point of view, distributed RTSs working on synchronous modes impose the need for a common clock, which makes the practical design and implementation of these systems very difficult. Furthermore, the advance of Very Large Scale Integration (VLSI) technology, that allows the integration of large numbers of high-performance processors on one chip, makes the idea of synchronising these processors with a common clock even more difficult [COHERENT 2005]. Thus, currently there is substantial research work (including the COHERENT project) on ideas such as applying asynchronous circuits instead of synchronous circuits, also on applying different design techniques to the building of distributed RTS, and finally, applying asynchronous communication techniques such as Globally Asynchronous, Locally Synchronous (GALS) and ACMS (especially in the COHERENT project). The next sections will briefly discuss these different lines of research.

1.2.3 Asynchronous and Synchronous systems

To simplify design, most of the designs of logic circuits are based on two major assumptions: all the signals are binary and time is a discrete function. By assuming that time is a discrete function, hazards (undesired signals transitions) and feedback can be ignored [Hauck 1995]. Asynchronous circuits keep the assumption that signals are binary, but remove the assumption that time is a discrete function. This, as suggested in [Hauck 1995], may imply several possible benefits such as: no clock

skew (i.e. “the difference between arrival times of the clock signal at different parts of the circuit”) since asynchronous circuits by definition have no globally distributed clock. Exploiting asynchronous mechanisms also can lead to lower power consumption because these circuits only need to have transitions in areas involved in the current computation. Asynchronous systems (circuits) indicate that when a computation is completed rather than waiting until all possible computations have completed, as is often necessary in synchronous systems. Moreover, in many asynchronous systems as suggested in [Hauck 1995], the migration to a new technology of only the most critical parts of the system may improve the overall performance, because performance in asynchronous systems tends to depend only on the current active path rather than the longest path as it happens in synchronous systems. Furthermore, asynchronous systems can wait an arbitrarily long time for an element to complete, allowing robust mutual exclusion. The last advantage of asynchronous circuits over synchronous circuits resides on the fact that, since there is no clock to which signals must be synchronised, asynchronous circuits may handle inputs from the outside world more elegantly than synchronous circuits, because the inputs usually are by nature asynchronous [Ghosh 2001].

Nevertheless, asynchronous circuits also have some problems. Firstly, asynchronous circuits are more difficult to design in an ad-hoc fashion than synchronous circuits. In synchronous circuits, by setting the clock rate to a long enough period, all worries about hazards and dynamic states of the circuit are normally removed. Nevertheless, designers of asynchronous systems must pay a great deal of attention to the dynamic state of the circuit. Notice that as mentioned, asynchronous designs do not have assumption of taking time as a discrete function rather than a continuous function; therefore the hazards that occur between transitions have to be considered. Moreover, placement, routing, partitioning, logic synthesis and other existing CAD tools in synchronous systems have to be modified (or even are not applicable at all) for asynchronous design circuits. Furthermore, although most of the advantages of asynchronous circuits are towards higher performance, it is not clear that they are actually faster in practice [Hauck 1995].

1.2.3.1 Synchronous and asynchronous design styles

The design methodologies to produce reliable (software) systems, address the problem in three different phases [O'Donoghue and Hull 1996], [Naedele 2001]: specification or definition, design and implementation. The first step is the creation of a logical or abstract model (process of specification). Secondly we have the process of design where the implementation model for a virtual machine is developed from the abstract model. The last phase corresponds to the process of implementation where virtual machine is placed in a physical machine [Muñoz 2002]. Design methodologies commonly require the support of CASE (Computer Aided Software Engineering) tools for their effective use. Some design methodologies are discussed further in chapter 3.

At this point, a brief introduction to a formal description for designing embedded real-time systems is presented, because of the importance that this design methodology has in the research work. Modular Approach to Software Construction Operation and Test (MASCOT), as introduced earlier on, is one of the real-time software development methodologies that has been considered in this work. It incorporates design representation, a method of deriving the design, a way of constructing software consistent with the design and tools for executing the constructed software and for testing it. The MASCOT method provides a design language (textual form) and a graphical notation (MASCOT network diagram).

There are other design tools based on a given formalism (a “formal method”). Although these methodologies are not going to be discussed in further chapters, they are introduced for completeness here. These design tools, used also to design asynchronous circuits, could be grouped in three different categories based on their underlying models [Muñoz 2002]: models based on logic such as Hardware Description Languages (HDL) descriptions, models that extend process algebra (both usually expressed in textual notation) like Signal Transition Graph (STG)/ State Graph (SG) synthesis, and state machine models that are often expressed in graphical notations such as Petri Nets or Timed Transition models like the multiple-input change Asynchronous Finite State Machine (AFSM) synthesis (e.g. burst-mode).

Petri Nets [Naedele 2001], [Mustafa 2000] is a mathematical model, which is used to specify the operations to be performed in a multiprocessing or multitasking environment, in others words; it is a model suitable to express concurrency. Petri Nets can be used to model systems and to analyse timing constraints and race conditions. However, if the system is highly complex, timing can become obscured. The method of STG is an interpreted free-choice Petri Nets (PN). The main goal of STG is to have the ability of expressing concurrency, but a weak point lies on its difficulty in specifying choices. This means that, future behaviour depends on a non-deterministic choice of equally likely choices. On the other hand, a burst-mode AFSM is specified by a state diagram which consists of a finite number of states, a set of labelled arcs connecting pair states, and a start state. Each arc is labelled with a set of possible signal transitions. Each transition consists of an input and an output burst. Given a state, when all of the specified sets of input transitions occur, the order of which is arbitrary, the machine generates a set of concurrent output changes and moves to a new state. Although the input choice of the burst-mode can be more flexible than STG, and e.g., it has been useful in specifying a number of controllers such as the Small Computer System Interface (SCSI) data transfer protocol [Yun et al. 1993], its main disadvantage is that the burst-mode still does not allow input transitions to be concurrent with output transitions.

The correctness of the RTS not only depends on the logical result of computations, but also on the time at which the results are produced. Therefore, the test for correctness of such systems is usually performed by formal proof (specification) and by verification, which is the process of proving that the system fits the assumptions made. Correctness proofs (sometimes called formal verification) are associated with formal methods. There are two techniques for going through the verification: analysis and synthesis. There is a fine distinction between them and sometimes they are intermingled (sometimes synthesis implies analysis). Analysis tries to verify all the properties of the RTS (e.g. timing constraints between tasks) by inspecting each part of the system and studying it. Synthesis tries to verify the properties of the system by building the system from the specification and then, examining all constraints. Then the analysis technique examines the different parts constituting the system in order to deduce its correct operation as a logical

consequence of design decisions, while synthesis experiments with the behaviour of the system by examining if the built system accomplishes the expected results.

Another formal description technique is called LOTOS (Language Of Temporal Ordering Specification). It is an ISO (International Standardisation Organisation) standard for designing services and protocols used in the communications of open systems [Muñoz 2002], [Turner 1993]. It is generally applicable to distributed, concurrent processing systems. The behaviour of a system can be characterised by LOTOS as a sequence of events or actions that happen in an orderly way in time. These actions are stated by gates and in order to represent the temporary sequence of these gates there is a set of operators, which can be built by behaviour expressions.

1.2.3.2 GALS

Globally Asynchronous, Locally Synchronous (GALS) is an approach based on the idea of guiding the overall hardware design towards a global asynchrony, although each part, that integrates the system, works in synchronous manner. There is substantial research work in GALS techniques, and as stated in [COHERENT 2005] “is widely expected to become popular”. The next paragraph refers to an example of the GALS approach as a matter of illustration.

In [Cristian and Fetzer 1999], the authors present a formal definition of a model called ‘timed asynchronous distributed system model’ or ‘timed model’ in short. The authors believe that this model is a good descriptor for existing distributed systems built from networked workstations. The main reason for it is that the timed model allows the processes to have access to the hardware clocks in a local access. It means that all the processes that are in one workstation (which is called a node in terms of the network) are considered as local processes and they have access to the hardware clock of the machine, but they do not have access to the clocks of other nodes. Hence, there is a locally synchronous process, because local processes are synchronised with the local nodes’ clock, but globally, the system is asynchronous because there is no global clock in the network.

1.2.3.3 Asynchronous Communication Mechanisms (ACMs)

ACMs may be defined as inter-process communication devices which allow writer and reader processes that are communicating, unconstrained access to the mechanism. In this way, the communicating processes do not share a clock. ACM are essentially implemented through shared variables or registers commonly as FIFO queue model. Even though there is significant research work conducted on the verification of existent ACMs [Clark 2000], [Xia 2000], [Mustafa 2000] or on the creation of new components which follow the ideas of ACMs like in [Cristian and Fetzer 1999]. The work here is focussed on the presentation and discussion of a specific taxonomy of ACMs [Simpson 1994e] which is presented in chapter 3 and used in chapters 4 and 5. [Simpson 1990c] defined three main properties: asynchrony, data coherence and data freshness, which are important in order to define certain types of ACMs. Therefore, the taxonomy of protocols illustrated in chapter 3, depends on how these protocols deal with data asynchrony, coherence and freshness. The asynchrony property refers to the unconstrained access to the mechanism, in terms of “when” and at “what” rate the writer and the reader can access the mechanism. The data coherence property refers to the atomicity of the data inside the mechanism, i.e. when the writer accesses the data the reader cannot read the same data at the same time. The data freshness property refers to the fact that the data that the reader and writer are dealing with is always the newest one.

1.3 *Aim and original contributions*

The aim of this project, within the context of COHERENT, has been the study of the application of specific ACMs and RTN principles to the system design of surveillance (multimedia) applications. The other aim of this work has been to try to overcome a major obstacle so as to enable the field (visual surveillance) to move forward by highlighting the need of a creation of a framework for designing surveillance systems. In chapter 2 a full review of the state-of-art in visual surveillance field has been presented, which has been published in a journal and included as the Introduction Chapter of a recent book. The original contributions of this work are presented in chapters 3, 4 and 5. The contributions presented in two workshops and two conferences correspond mainly to the work presented in chapters 4 and 5. The contribution of chapter 3 is based on establishing a

comparison framework between two software techniques Object Oriented (OO) and RTN; the conclusions of chapter 3 establish the theoretical ideas that can guide the creation of the framework for designing surveillance systems. Once the theoretical ideas for the framework are established, one of chapter 4's contributions consists in applying these ideas to the design of an existing surveillance system. The other contribution of chapter 4 consists of the comparison of the architecture design of the same surveillance system using two different software approaches based on OO and RTN concepts respectively. The bases for the creation of the framework are finally established in chapter 5 which presents and discusses an original design of a generic surveillance system.

1.4 *Structure of the thesis*

This thesis is structured into six chapters. Chapter 1 discusses the background of the research within the field from the point of view of the main areas addressed by the COHERENT project. In chapter 2, a brief introduction to computer vision systems is given followed by a historical review of the evolution of visual surveillance, the general requirements for designing surveillance systems and concluding with an overview of currently popular image processing techniques and design approaches used in non-trivial surveillance systems. As mentioned, the main original contributions of the work are contained in chapters 3, chapter 4 and chapter 5. In chapter 3, an overview is given of different (software) system development methods that may be used for designing wide-area intelligent surveillance systems. We show that it is important to consider what is a major trend in current approaches, mainly the use of the object-oriented paradigm, against a methodology more firmly rooted in distributed safety-critical systems namely that of Real Time Networks (RTN). Then, we develop a framework, which is presented in chapter 4, to compare a popular object-oriented tool used to build these systems (CORBA) and those associated with the proposed method RTN method (MASCOT, DORIS through a case study. In chapter 5, a design of a new distributed surveillance system is proposed using the recommended method. Conclusions and future work are discussed in chapter 6.

2 The state of art of Intelligent Surveillance Systems

2.1 *Introduction*

In this chapter a preamble of vision systems is presented with an overview of surveillance systems mainly based on [Valera and Velastin 2005b]. The overview consists of three main parts: an historical introduction of these systems, a description of the general requirements and finally the state-of-art of the existing vision surveillance systems at the time of writing. The historical introduction looks at the wider picture of the evolution of these systems, starting from the first vision surveillance systems to the latest systems which are still a subject of current research. Then, an introduction of the general requirements in surveillance systems on different applications is presented, illustrating the essential functionality of such systems. After that, a survey of the state-of-art of different existing vision surveillance systems is presented, starting by an overview of conventional techniques used to build these systems, moving afterwards to the presentation of some examples of such systems and finishing with a discussion of some properties that we find very important to include in the analysis of these systems such as distribution and communication.

2.2 *Vision Systems*

One of the major historical society advances was industrialization and therefore the automation of certain processes. Since then, research and development has leant towards the automation of most activities in industry, reducing cost, time and use of human resources. Vision systems may increase, in some fields, the degree of automation in processes or even introduce a certain degree of automation in processes that were not automated. For instance, the application of vision system is widely used in the medical field for diagnostic purposes as in [Tierney et al. 2000], or to improve the efficiency in information cataloguing. High-speed data streams resulting from the operation of on-line instruments and imaging systems are important steps leading to a modern health care system in which the communication between centres allows one to exchange information and consequently improve the efficiency of health care.

Traditionally, surveillance systems were built for monitoring certain activities in military units such as planes and ships using sensors like radars or sonars. Recent events, including major terrorist attacks, have led to the increase in demand for security in society. This in turn has forced governments to make personal and asset security a priority in their policies. Vision systems are rapidly gaining more importance in the surveillance field, providing a form of automation within the surveillance task of the environment where it is applied. Therefore, the demand for remote monitoring for safety and security purposes has received particular attention in some areas like road traffic control and public or private installations such as car parks, airports or public transport installations such as bus or underground railway networks. To see more applications please refer to [Valera and Velastin 2005b].

2.3 Evolution of Intelligent Surveillance Systems

As mentioned in chapter 1, the historical evolution of vision systems in surveillance applications goes from what literature in the field calls the first generation vision surveillance system through to the second and then third generation surveillance systems. Table 2-1 shows a summary of the evolution of such systems. Analogue Closed Circuit TeleVision (CCTV) systems are considered as the first generation of surveillance systems. These systems consisted of groups of cameras connected directly to monitors. In subsequent developments, the cameras were connected through a switch or matrix which distributed the analogue signal to one or more monitors. Initially the systems were installed in closed spaces, although rapidly they were installed in open spaces as well. In [Nwagboso 1998] the integration of these systems to monitor transport systems is discussed. As shown in Table 2-1, currently, the majority of CCTV systems use analogue techniques for image distribution and storage, even though conventional CCTV cameras generally use a digital Charge Coupled Device (CCD) to capture images. The digital image is then converted into an analogue composite video signal, causing some picture degradation, which is then connected to the CCTV matrix, monitors and recording equipment generally via coaxial cables. The current research on these systems is based on switching the analogue CCTV systems to digital technology.

The rapid increase in the use of CCTV systems implied an expansion in size and complexity. At the same time this expansion resulted, perhaps surprisingly, in a decrease in the relative effectiveness of surveillance and of recognition of activities of interest in real-time. The substantial improvement in the techniques of digital image processing and the low cost of dedicated PCs for these image processing techniques influenced the introduction of new technologies in surveillance systems. Then, a new second generation of surveillance system arose. The introduction of such systems has provided improvements in surveillance applications by providing certain automation, for example, as is the case for motion detection methods used to detect presence and to minimise recordings of uneventful (empty) scenes or the very successful introduction of automatic plate number recognition systems e.g. for road traffic congestion/offence charging. The type of image processing techniques ranges from simple change detection or the elimination of image noise to more complicated processing tasks like recognition and tracking of objects and the interpretation of scenarios. The current research in this second generation is based on improving the efficiency and robustness of computer vision algorithms such as event detection. There is also some research on automatic learning techniques for recognising patterns of behaviours and scene variations.

The introduction of new technologies in the market such as high speed networks has led to the creation of remote control surveillance. These systems are based on the use of sensors, like cameras installed for the purpose of surveying where all the information is processed in a remote location. The third generation of surveillance systems consists of the integration of these new technologies with the processing techniques coming from previous systems. Therefore, such systems are based on the distribution and separation of the processing tasks into a low level and high level partly due to the proliferation of the devices called Digital Signal Processors (DSP), which allows building intelligent cameras or smart cameras with autonomous (local) processing capacities.

1st generation	
Techniques	Analogue CCTV systems
Advantages	<ul style="list-style-type: none"> – They give good performance in some situations. – Mature technology.
Problems	Use analogue techniques for image distribution and storage
Current Research	<ul style="list-style-type: none"> – Digital versus analogue <ul style="list-style-type: none"> • Digital video recording – CCTV video compression
2nd generation	
Techniques	Automated visual surveillance by combining computer vision technology with CCTV systems
Advantages	Increase the surveillance efficiency of CCTV systems
Problems	Robust detection and tracking algorithms required for behavioural analysis
Current Research	<ul style="list-style-type: none"> – Real-time robust computer vision algorithms. – Automatic learning of scene variability and patterns of behaviours. – Bridging the gap between the statistical analysis of a scene and producing natural language interpretations.
3rd generation	
Techniques	Automated wide-area surveillance system
Advantages	<ul style="list-style-type: none"> – More accurate information as a result of combining different kind of sensors. – Distribution
Problems	<ul style="list-style-type: none"> – Distribution of information (integration and communication) – Design methodology – Moving platforms – Multi-sensor platforms
Current Research	<ul style="list-style-type: none"> – Distributed versus centralised intelligence – Data fusion – Probabilistic reasoning framework – Multi-camera surveillance techniques

Table 2-1. Summary of the technical evolution of intelligent surveillance systems (from Valera and Velastin 2005b))

2.4 *Requirements of an Intelligent Surveillance System*

To create and develop such systems it is essential to define the requirements of the system, which match the needs of the user enabling these demands to be satisfied. The main goal that is expected of third generation vision surveillance application, based on end-user requirements, is to provide cost effective good scene understanding (and learning) aimed at attracting the attention of human operators in real-time in a widespread geographic area, using a variety of sensors and sources of contextual information necessary for decision support (such as the availability of response units in an area where a problem has been detected).

From the architectural design point of view, this requirement implies different constraints. In scene understanding, e.g., the high variability in the scene conditions and the poor structure of monitoring hint of the need to use more sophisticated image processing algorithms, pattern recognition methods and robust scene description. For example, the mounting position of the cameras in a metro-station and consequently the video-signal of the digital pictures are often not in optimal conditions. Problems may be caused by poor lighting, environments that cause reflections or by the heights and the perspective of the resulting mounted cameras (the position of the cameras is generally optimised to traffic monitoring or to give a human monitor maximum visual coverage and not necessarily to security or to machine monitoring).

Good performance processing capacities are required in multi-sensor environments, especially when there are different kinds of sensors in diverse spatial locations acquiring the same type of real-time information in a monitored area. Therefore, spatially distributed multi-sensor environments present interesting opportunities and challenges for surveillance. Recently, there has been some investigation of data fusion techniques in surveillance to cope with the sharing of information obtained from different types of sensors [Collins et al. 2000a]. The communication aspects within different parts of the system play an important role either due to the bandwidth constraints or the asymmetric nature of the communication [Regazzoni et al. 2001]. Another relevant aspect is the security in the communications between modules. For some vision surveillance systems, data needs to be sent over open

networks and the information protection leads to a critical issue for ensuring privacy and for authenticating [Barni et al. 2000] conditions of these services. The trend in the requirements of such systems also tends to include the viability of adding an automatic learning capability in these systems to improve the end-user constraints factors, by automatically developing models of scenes to be recognised as potentially dangerous events from a training set of presented examples [Thonnat and Rota 2000], [Ivanov and Bobick 2000], [Gong and Xiang 2003].

2.4.1 Surveillance system requirements for transit applications

Requirements may differ from one surveillance application to another. In surveillance systems for intelligent transport [Pellegrini and Tonani 1998], the continued increase in traffic density emphasises the need to take action on the deterioration of traffic congestion through competent traffic management, enhancing safety and security within the traffic network. Therefore, the functionality and the effectiveness of the measurement of traffic scenes by monitoring and collecting data using vision surveillance systems should substantially assist in better traffic control, incident management and traffic law enforcement.

To achieve this, the system should be an integrated system which can link into incident monitoring system, in-vehicle systems which are likely to accept information related to safety and security from the law enforcement and the existing traffic control systems. Most of the technology on the current traffic control systems in the UK is mainly CCTV linked into a control unit and generally used for passive traffic monitoring. The natural linkage that should be implemented is the control interface systems, the surveillance signal processing unit and the central processing systems, which encapsulates the database of the vehicle details in the traffic network. In extensive capability of the control and processing unit, the users needs, in terms of the organisational and personnel requirements will have to also be met. In [Pellegrini and Tonani 1998] it is considered that the response time of the whole system, including human response to an accident in a highway shall be very fast (less than 5 min) in order to avoid another possible collision and minimise the false alarm rate (ideally to zero).

The incident monitoring system can be a network of smart surveillance cameras that should automatically trigger image save routines in order to provide the footage of vehicle crashes on the computer. Five minutes of the recording prior to the incident and the incident itself are stored in a computer for a post analysis by the enforcement agencies or insurance companies [Pellegrini and Tonani 1998]. The surveillance should work by continuously monitoring accident black spots on the network by storing the video images uninterruptedly on a computer in a loop and re-recorded over the past scenes until the smart camera detects the start of vehicle collision. The localization of these cameras over the road network follows the same criteria as the CCTV cameras. Adverse weather conditions such as fog or dense rain may limit their efficacy. In closed areas like tunnels, because of geometrical constraints, fixed cameras should be used each covering no more than 300 metres of straight road in order to avoid possible occlusions. In non-straight roads, up to one camera every 100 m might be needed [Pellegrini and Tonani 1998]. The image acquisition and recording of air-pollution monitoring system should be triggered using the same techniques as that of incident monitoring system.

2.4.2 Surveillance system requirements for port applications

In another environment such as ports, security (in its boundaries and inner areas) is a growing issue as it represents the main gates for international trades around the world, where several personnel work day and night for different activities. In fact, port areas differ in the destination of use such as industrial, commercial, tourism or marinas for pleasure boats activities. All these activities are associated with different infrastructures and are carried out during different periods requiring specific personnel and equipment. Then surveillance is required to guarantee the “feeling of” security and control at gates or at public areas opening. Although usually the video-based control of cargo handling and transfer equipment exist in many port terminals, surveillance in the form of traffic control applications like truck access management and the control of their movements inside ports and terminals is also applicable in this domain, as well as the surveillance of goods and workers. In this environment, the installation of an impressive amount of cameras is required because of the usual wide extension of surveillance area, providing great

amounts of data to process and transmit. Therefore, bandwidth constraints requirements and good performance in the processing units are required. Moreover, the difficulty to survey increases depending on the kind of traffic, e.g. cargo is enclosed in containers which all look the same and furthermore, it is not possible to see the contents inside them, thus a multi-sensor environment is required. In this type of scenario, clearly an availability of 365 days per year, 24 hours a day is what is required.

2.4.3 Metro and Railway Stations surveillance system requirements

End-user requirements for Railway Station Surveillance Systems and Metro Stations are based on two principles obtained from statistical studies of real situations that occur in railway and metro stations [Ronetti and Dambra 2000]. The first principle is grounded on the need of the company; to survey the people, end-users and employees, and to survey their assets, which may be damaged as a cause of vandal behaviours or failures. The second principle is based upon the need to increase the *perception* (or feeling) of security; reducing the feeling of security tends to produce losses for the company because people choose not to travel in their networks. From these two basic end-user requirements consequent requirements can be extrapolated: to be able to detect and recognise certain events and to have a better scene understanding. A good monitored infrastructure and location is required giving a good view of all areas of the facilities. Therefore, a skilful management of all this information is required, in other words, the system should have a full-coverage, be extensible and may integrate different technologies and consequently bandwidth constraints should be taken into account and of course be usable by staff. For example, the linking of different technologies allows a wireless call from a train operator to generate its position on a display, hence it gives a better control and easier maintenance, because it is possible to know straightaway if there is any problem and sometimes to know which kind of problem it is because the operator is able to report it.

To reduce the passengers' feeling of insecurity and guarantee security, a fast response is vital in a dangerous situation. The system must produce the necessary alarms in real-time and provide the results with sufficient clarity to attract a human

operator's attention by pre-selecting only the interesting outputs, which are usually images. For example, if an emergency call is made by the public or personnel when an incident occurs in the installation, the conversation may be coupled with the cameras in order to record pictures of the callers and their conversation. Thus, the operator can be alerted of this event with enough information to handle the incident in a proper way. It is assumed that the operator is not a computer expert so the machine interface needs to be simple.

Therefore, the system requires good performance in terms of response time and low false alarm rates. From a safety point of view it is preferable to have a false alarm than a non-detected alarm. The system needs to be reliable enough to cope with long periods of loss of video inputs and failure-tolerant as a failure on a part of the system should not paralyse the entire system. In [London Underground Limited (n.d.)] reliability is defined in terms of Mean Time Between Failure (MTBF) and Mean Time To Repair (MTTR) assumed to be four hours. The MTBF has three different delimitations depending on the kind of failure: for a complete system failure the MTBF should be greater than 2.6×10^4 hours, for a single failure with more than one output it should be greater than 1.8×10^4 hours and finally for a single failure with one output the MTBF should be greater than 10^4 . The availability should be 99%, 24 hours per day, 365 days per year. The systems needs to be capable of storing all the information extracted from different sensors, especially the outputs from the cameras, with enough quality to allow them to be used in other fields like a police investigation or court of law. In countries like the UK, the surveillance videotapes may be used as evidence in court [Geradts and Bijhold 2000] or used in crime investigation by the police.

The last important requirement in these applications is that the system should interface with existing equipment without much cost for a technical adaptation. This technical compatibility concerns the type of cameras (b/w, colour, pan-tilt-zoom), the transmission types (fibre optical, coaxial, wire), switching matrix with possible special interfaces, monitor-places with associated panels and keyboards.

2.5 State-of-art in the design of visual surveillance systems

This section is divided in two main subsections: the first part (subsection 2.5.1) summarises research that addresses the main image processing tasks that were mentioned in the previous section i.e. object detection, object recognition, tracking, behaviour, activities analysis and databases. It is important to highlight that the availability of a given technique or set of techniques is necessary but not sufficient to deploy a potentially large surveillance system, which implies networks of cameras and distribution of processing capacities to deal with the signals from these cameras. Therefore, in the second part of this section what has been done to propose surveillance systems that address these requirements is reviewed. The majority of the surveillance systems reviewed in this chapter are based on transport or parking lots applications [Valera and Velastin 2005b]. The reason as explained in [Valera and Velastin 2005b], is because most reported distributed systems tend to originate from academic research which has tended to focus on these domains (e.g. by using university campuses for experimentation or the increasing research funding to investigate solutions in public transport).

2.5.1 Processing components in surveillance systems

A typical configuration of processing modules is illustrated in Figure 2- 1. These modules constitute the low-level building blocks necessary for any distributed surveillance system. Each of the following subsections outline the most popular image processing techniques used in each of these modules.

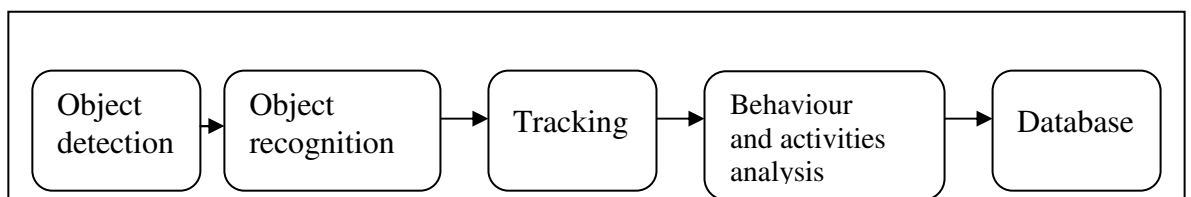


Figure 2- 1. Traditional flow of processing in visual surveillance from [Valera and Velastin 2005b].

2.5.1.1 Object detection

There are two main conventional approaches to object detection: ‘temporal difference’ and ‘background subtraction’. The first approach consists in the subtraction of two consecutive frames followed by thresholding. The second technique is based on the subtraction of a background or reference model and the

current image followed by a labelling process. After applying one of these approaches, morphological operations are typically applied to reduce the noise of the image difference. The temporal difference technique has good performance in dynamic environments because it is very adaptive, but it has a poor performance on extracting all the relevant object pixels. On the other hand, the background subtraction has a better performance on extracting object information but it is sensitive to dynamic changes in the environment (see Figure 2- 2 and Figure 2- 3).



Figure 2- 2. Example of a temporal difference technique used in motion detection (from [Valera and Velastin 2005b]).



Figure 2- 3. Example of a background subtraction technique used in motion detection. In this example a bounding box is drawn to fit the object detected (from [Valera and Velastin 2005b]).

An adaptive background subtraction technique involves creating a background model and continuously upgrading it to avoid poor detection when there are changes in the environment. There are different techniques to model the background, which are directly related to the application. For example, in indoor environments with good lighting conditions and stationary cameras, it is possible to create a simple background model by temporally smoothing the sequence of acquired images in a short time as described in [Haritaoglu et al. 2000], [Nguyen et al. 2003a], and [Jaynes 1999]

Outdoor environments usually have high variability in scene conditions, thus it is necessary to have robust adaptive background models, even though these robust models are computationally more expensive. A typical example is the use of a Gaussian Model (GM) that models the intensity of each pixel with a single Gaussian distribution [Wren et al. 1997] or with more than one Gaussian distribution Gaussian Mixture Models(GMM). In [Boult et al. 2001], due to the particular characteristics of the environment (a forest), they use a combination of two Gaussian Mixture Models to cope with a bimodal background (e.g. movement of trees in the wind). The authors in [Stauffer et al. 2000] use a mixture of Gaussians to model each pixel. The method they adopted handles slow lighting changes by slowly adapting the values of the Gaussians. A similar method is used in [Pavlidis et al. 2001]. In [Ng et al. 1999] the background model is based on estimating the noise of each pixel in a sequence of background images. From the estimated noise the pixels that represent moving regions are detected. Other techniques use groups of pixels as the basic units for tracking, and the pixels are grouped by clustering techniques combining colour information (R,G,B) and spatial dimension (x, y) to make the clustering more robust. Algorithms as such Expectation Minimisation (EM) are applied to track moving objects as clusters of pixels significantly different from the corresponding image reference, e.g. in [Bennewitz et al. 2002] the authors use EM to simultaneously cluster trajectories belonging to one motion behaviour and then to learn the characteristic motions of this behaviour.

In [Oren et al. 1997] the reported object detection technique is based on wavelet coefficients to detect frontal and rear views of pedestrians. By using a variant of Haar wavelet coefficients as a low-level process of the intensity of the images, it is possible to extract high-level information of the object (pedestrian) to detect, e.g. shape information. In a training stage, the coefficients that most accurately represent the object to be detected are selected using large training sets. Once the best coefficients have been selected, they use a Support Vector Machine (SVM) to classify the training set. During the detection stage, the selected features are extracted from the image and then the SVM is applied to verify the detection of the object. The advantage of using wavelet techniques is that of not having to rely on explicit colour information or textures. Therefore, they can be useful in applications

where there is a lack of colour information (a usual occurrence in indoor surveillance). Moreover, using wavelets implies a significant reduction of data in the learning stage. However, the authors only model the front and the rear views of pedestrian. In the case of groups of people that stop, talk or walk perpendicular to the view of the camera, the algorithm is not able to detect the people. Furthermore, an object, with similar intensity characteristics as a frontal or rear human, is likely to generate a false positive. Another line of research is based on the detection of contours of persons by using principal component analysis (PCA). Finally, as far as motion segmentation is concerned, techniques based on optic flow may be useful when a system uses moving cameras as in [Ferryman et al. 2000], although there are known problems when the image size of the objects to be tracked is small.

2.5.1.2 Object recognition, tracking and performance evaluation

Tracking techniques can be split in two main approaches: 2D models with or without explicit shape models and 3D models. For example in [Ferryman et al. 2000] the 3D geometrical model of a car, a van and a lorry is used to track vehicles in a highway. The model-based approach uses explicit *a priori* geometrical knowledge of the objects to follow, which in surveillance applications are usually people, vehicles or both. In [Zhi-Hong 2003] the author uses two 2D models to track cars: a rectangular model for a passing car that is close to the camera and a U-shape model for the rear of the car in the distance or just in front of the camera. The system consists of an image acquisition module, a lane and car detection, a process co-ordinator and a multiple car tracker. In some multi-camera systems like [Jaynes 1999], the focus is on extracting trajectories, which are used to build a geometric and probabilistic model for long-term prediction, and not the object itself. The *a priori* knowledge can be obtained by computing the object's appearance as a function of its position relative to the camera. The scene geometry is obtained in the same way. In order to build the shape models, the use of camera calibration techniques becomes important. A survey of different techniques for camera calibration can be found in [Hemayed 2003]. Once *a priori* knowledge is available, it may be utilized in a robust tracking algorithm dealing with varying conditions such as changing illumination, offering a better performance in solving (self) occlusions or (self) collisions. It is relatively simple to create constraints in the objects' appearance model by using model-based approaches; e.g. the constraint

that people appear upright and in contact with the ground is commonly used in indoor and outdoor applications.

The object recognition task then becomes the process of utilising model-based techniques in an attempt to exploit such knowledge. A number of approaches can be applied to classify the new detected objects. The integrated system presented in [Remagnino et al. 1997] and [Ferryman et al. 2000] can recognise and track vehicles using a defined 3D model of a vehicle, giving its position in the ground plane and its orientation. It can also recognise and track pedestrians using a prior 2D model silhouette shape, based on B-spline contours. A common tracking method is to use a filtering mechanism to predict each movement of the recognised object. The filter most commonly used in surveillance systems is the Kalman Filter [Remagnino et al. 1997], [Nguyen et al. 2003a]. Fitting bounding boxes or ellipses, which are commonly called 'blobs', to image regions of maximum probability performs another tracking approach based on statistical models. In [Wren et al. 1997] the author models and tracks different parts of a human body using blobs, which are described in statistical terms by a spatial and colour Gaussian distribution. In some situations of interest the assumptions made to apply linear or Gaussian filters do not hold, and then non-linear Bayesian filters, such as Extended Kalman filters (EKF) or particle filters have been proposed. Work described in [Arulampalam et al. 2002] illustrates that in highly non-linear environments particle filters give better performance than EKF. A Particle Filter (PF) is a numerical method, which weights (or 'particle') a representation of posterior probability densities by resampling a set of random samples associated with a weight and computing the estimate probabilities based on these weights. Then, the critical design decision using particle filters relies on the choice of importance (the initial weight) of the density function.

Another tracking approach consists in using connected-components [Boulton et al. 2001] to segment the changes in the scene into different objects without any prior knowledge. The approach has a good performance when the object is small, with a low-resolution approximation, and the camera placement is chosen carefully. Hidden Markov Models (HMMs) have also been used for tracking purposes as presented in [Hai Bui et al. 2001], where the authors use an extension of HMM to

predict and track objects trajectories. Although HMM filters are suitable for dynamic environments (because there is no assumption in the model or in the characterisation of the type of the noise like as required when using Kalman Filters), off-line training data are required. Recent research has been carried out on the creation of semi-automatic tools that can help create a large set of ground truth data that is necessary for evaluating the performance of the tracking algorithms [Black et al. 2003].

2.5.1.3 Behavioural analysis

The next stage of a surveillance system recognises and understands activities and behaviours of the tracked objects. This stage broadly corresponds to a classification problem of the time-varying feature data that are provided by the preceding stages. Therefore, it consists in matching a measured sequence to a pre-compiled library of labelled sequences that represent prototypical actions that need to be learnt by the system via training sequences. There are several approaches for matching time-varying data. Dynamic Time Warping (DTW) is a time-varying technique widely used in speech recognition, image pattern as in [Rath and Manmatha 2003] and recently in human movement patterns [Oates et al. 2000]. It consists of matching a test pattern with a reference pattern. Although it is a robust technique, it is now less favoured than dynamic probabilistic network models like HMM (Hidden Markov Models) and Bayesian Networks [Nguyen et al. 2003b], [Ivanov and Bobick 2000]. The last time-varying technique that is not as widespread as HMM, because it is less investigated for activity recognition, is Neural Networks (NN). In [Thonnat and Rota 2000] the recognition of behaviours and activities is done using a declarative model to represent scenarios, and a logic-based approach to recognise predefined scenario models.

2.5.1.4 Database

One of the final stages in a surveillance system is storage and retrieval. Relatively little research has been done in how to store and retrieve all the obtained surveillance information in an efficient manner, especially when it is possible to have different data formats and type of information to retrieve. In [Makris et al. 2004] the authors investigate the definition and creation of data models to support the storage of different levels of abstraction of tracking data into a surveillance database.

In [Decleir et al. 1999] the authors develop a data model and a rule-based query language for video content based indexing and retrieval. Their data model allows facts as well as objects and constraints. Retrieval is based on a rule-based query language that has declarative and operational semantics, which can be used to gather relations between information represented in the model. A video sequence is split into a set of fragments and each fragment can be analysed to extract the information (symbolic descriptions) of interest to store into the database. In [Stringa and Regazzoni 1998] retrieval is performed on the basis of object classification. A stored video sequence consists of 24 frames; the last frame is the key frame that contains the information about the whole sequence. Retrieval is performed using a feature vector where each component contains information obtained from the event detection module.

2.6 *Examples of surveillance systems*

In following sections examples of surveillance systems are presented although to read more examples refer to [Valera and Velastin 2005b] where an extend sample of examples of surveillance systems is presented. In this section a distinction between surveillance for indoor and outdoor applications is made. The reason is because of the differences in the design at the architectural and algorithmic implementation levels. The topology of indoor environments is also different from that of the outdoor environments.

2.6.1 Commercial surveillance system for outdoor applications

An example of a commercial system intended for outdoor applications, is DETER [Paulidis and Morellas 2002], [Pavlidis et al. 2001] (Detection of Events for Threat Evaluation and Recognition). The architecture of the DETER system is illustrated in Figure 2- 4. It is aimed at reporting unusual moving patterns of pedestrians and vehicles in outdoor environments such as car parks. The system consists of two parts: the computer vision module and the threat assessment or alarms management module. The computer vision part deals with the detection, recognition and tracking of objects across cameras. In order to do this, the system fuses the view of multiple cameras into one view and then performs the tracking of the objects. The threat

assessment part consists of feature assembly or high-level semantic recognition, the off-line training and the on-line threat classifier. The system has been evaluated in a real environment by end-users, and it had a good performance in object detection and recognition. However, as it is pointed out in [Pavlidis et al. 2001], DETER employs a relatively small number of cameras because it is a cost-sensitive application. It is not clear whether the system has the functionality for retrieval and even though the threat assessment has good performance, there is a lack of a feedback loop in this part that could help improve performance.

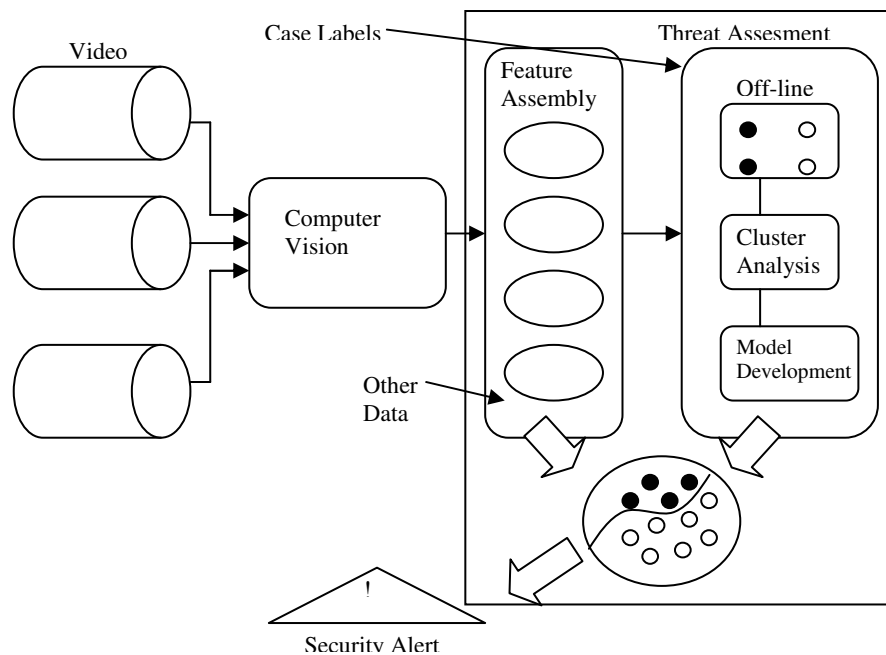


Figure 2- 4. Architecture of DETER system (from [Valera and Velastin 2005b]).

2.6.2 Surveillance systems for parking lots applications

Another integrated visual surveillance for vehicles and pedestrians in parking lots is presented in [Remagnino et al. 1997]. This system presents a novel approach to deal with interactions between objects (vehicles and pedestrians) in a hybrid tracking system. The system consists of two visual modules capable of identifying and tracking vehicles and pedestrians in a complex dynamic scene. However, this is an example of a system that considers tracking as the only surveillance task, even though the authors pointed out in [Remagnino et al. 1997] the need for a semantic interpretation of the tracking results for scene recognition. Furthermore, a “handover” tracking algorithm across cameras has not been established.

It is important to have a semantic interpretation of the behaviours of the recognised objects in order to build an automated surveillance system that is able to recognise and learn from the events and interactions that occur in a monitored environment. For example in [Ivanov et al. 1999], the authors illustrated a video-based surveillance system to monitor activities in a parking lot that performs a semantic interpretation of recognised events and interactions. The system consists of three parts: the tracker which tracks the objects and collects their movements into partial tracks; the event generator which generates discrete events from the partial tracks according to a simple environment model and finally, a parser that analyses the events according to a Stochastic Context-Free Grammar (SCFG) model which structurally describes possible activities. This system, as the one in [Remagnino et al. 1997], is aimed at proving the algorithms more than at creating a surveillance system for monitoring a wide area (the system uses a single stationary camera). Furthermore, it is not clear how the system distinguishes between cars and pedestrians because the authors do not use any shape model.

In [Jian-Guang et al. 2003] visual traffic surveillance for automatic identification and description of the behaviours of vehicles within parking lots scenes is presented. The system consists of a motion module, model visualisation and pose refinement, tracking and trajectory-based semantic interpretation of vehicle behaviour. The system uses a combination of colour cues and brightness information to construct the background model and applies connectivity information for pixel classification. Using camera calibration information they project the 3D model of a car onto the image plane and they use the 3D shape model-based method for pose evaluation. The tracking module is performed using EKF. The semantic interpretation module is realised by three steps: trajectory classification, then an on-line classification step using Bayesian classifiers and finally natural language descriptions are applied to the trajectories patterns of the cars that have been recognised. Although this system introduces a semantic interpretation for car behaviours, it is not clear how this system handles the interactions of several objects in the same scene at the time, and consequently the occlusions between objects. Another possible limitation is the lack of different models to represent different type of vehicles (c.f. [Remagnino et al. 1997] includes separate 3D models for a car, van and lorry).

2.6.3 Surveillance systems for traffic control application

The author in [Nwagboso 1998] expresses the need to integrate video-based surveillance systems with existing traffic control systems to develop the next generation of advanced traffic control and management systems. Most of the technologies in traffic control are based on CCTV technology linked to a control unit and in most cases for reactive manual traffic monitoring. However, there are an increasing number of CCTV systems using image processing techniques in urban road network and highways. Therefore, the author in [Nwagboso 1998] proposes to combine these systems with other existing surveillance traffic systems like surveillance system, which are based on networks of smart cameras. The term “smart camera” (or “intelligent camera”) is normally used to refer to a camera that has processing capabilities (either in the same casing or nearby), so that event detection and storage of event video can be done autonomously by the camera. Thus, normally, it is only necessary to communicate with a central point when significant events occur.

Usually integrated surveillance systems consist of a control unit system, which manages the outputs from the different surveillance systems, a surveillance signal processing unit and a central processing unit which encapsulates a vehicle ownership database. The suggestion in [Nwagboso 1998] of having a control unit, which is separated from the rest of the modules, is an important aspect in the design of a third generation surveillance system. However, to survey a wide-area implies geographical distribution of equipment and a hierarchical structure of the personnel who deal with security. Therefore for better scalability, usability, and robustness of the system, it is desirable to have more than one control unit. Their design is likely to follow a hierarchical structure (from low-level to high-level control) that mirrors what is done in image processing where there is a differentiation between low-level and high-level processing tasks.

Following the aim of [Beymer et al. 1997], the authors in [Heikkila and Silven 1999] develop a vision-based surveillance system to monitor traffic flow on a road, but focusing on the detection of cyclists and pedestrians. The system consists of two main distributed processing modules: the tracking module which processes in real-

time and is placed roadside on a pole and the analysis module which is performed off-line in a PC. The tracking module consists of four tasks: motion detection, filtering, feature extraction using Quasi-Topological features (QTC) and tracking using first order Kalman filters. The shape and the trajectory of the recognised objects are extracted and stored in a removable memory card, which is transferred to the PC to achieve the analysis process using Learning Vector Quantization for producing the final counting. This system has some shortcomings. The image algorithms are not robust enough (the background model is not robust enough to cope with changing conditions or shadows) and depend on the position of the camera. The second problem is that even though tracking is performed in real time, the analysis is performed off-line, therefore it is not possible to do flow statistics or monitoring in real-time.

2.6.4 Surveillance system for port applications

In [Pozzobon et al. 1998] the architecture of a system for surveillance in a maritime port is presented. The system consists of two subsystems: image acquisition and visualisation. The architecture is based on a Client/Server design. The image acquisition subsystem has video server module, which can handle four cameras at the same time. This module acquires the images from camera streams, which are compressed, and then the module broadcasts the compressed images to the network using TCP/IP and at the same time records the images on hard disks. The visualisation module is performed by client subsystems, which are based on PC boards. This module allows the selection of any camera using a pre-configured map and the configuration of the video server. Using an internet server module it is possible to display the images through internet. The system is claimed to have the capability of supporting more than 100 cameras and 100 client stations at the same time, even though the reported implementation had 24 cameras installed mainly at the gates of the port. This is an example of a simple video surveillance system (with no image interpretation), which only consists of image acquisition, distribution and display. The interesting point in this system is to see the use of a client and server architecture to deal with the distribution of the multiple digital images. Moreover, the acquisition and visualisation modules have been

encapsulated in a way such that scalability of the system can be accomplished in a straightforward way, by integrating modules into the system in a “drop” operation.

2.6.5 Surveillance systems for public transport applications

In [Ronetti and Dambra 2000] a railway station CCTV surveillance system in Italy is presented. The system has a hierarchical structure distributed between main (central) control rooms and peripheral site (station) control rooms. The tasks that are performed in the central control room are: acquisition and display of the live or recorded images. The system also allows the acquisition of images from all the station control rooms through communication links and through specific coding and decoding devices. Digital recording, storage and retrieval of the image sequences as well as the selection of specific CCTV camera and the deactivation of the alarm system are carried out in the central room. The main tasks performed in each station control room are: acquisition of the images from the local station CCTV cameras, the link with the central control room to transmit the acquired or archived images in real time and to receive configuration procedures. The station control room also handles the transmission of an image of a specific CCTV camera at higher rate under request or automatically when an alarm has been raised. The management and deactivation of local alarms is handled from the station control room. Apart from the central control room and the station control rooms, there is a crisis room for the management of railway emergencies. Although this system presents a semi-automatic, hierarchical and distributed surveillance system, the role played by human operators is still central because there is no processing (object recognition or motion estimation) to channel the attention of the monitoring personnel.

Ideally, a third generation of surveillance system for public transport applications would provide a high level of automation in the management of information as well as that of alarms and emergencies. That is the stated aim of the following two surveillance systems research projects (other projects in public transportation that are not included here can be found in [Velašin 2003]).

CROMATICA [CROMATICA 1999] (Crowd Monitoring with Telematic and Communication Assistance) was an EU-funded project whose main goal was to

improve the surveillance of passengers in public transport, enabling the use and integration of technologies like video-based detection and wireless transmission. This was followed by another EU-funded project called PRISMATICA [Ping Lai Lo et al. 2003] (Pro-active Integrated Systems for Security Management by Technological Institutional and Communication Assistance) that looked at social, ethical, organisational and technical aspects of surveillance for public transport. A main technical output was a distributed surveillance system. It is not only a wide-area video-based distributed system like ADVISOR (Annotated Digital Video for Intelligent Surveillance and Optimised Retrieval) [ADVISOR 2003], but it is also a wide-area multi-sensor distributed system, receiving inputs from CCTV, local wireless camera networks, smart cards and audio sensors. PRISMATICA then consists of a network of intelligent devices (that process sensor inputs) that send and receive messages to/from a central server module (called “MIPSA”) that co-ordinates device activity, archives/retrieves data and provides the interface with a human operator. Figure 2-5 shows the architecture of PRISMATICA, which is a modular and scalable architecture approach using standard commercial hardware. PRISMATICA employs a centralised approach.

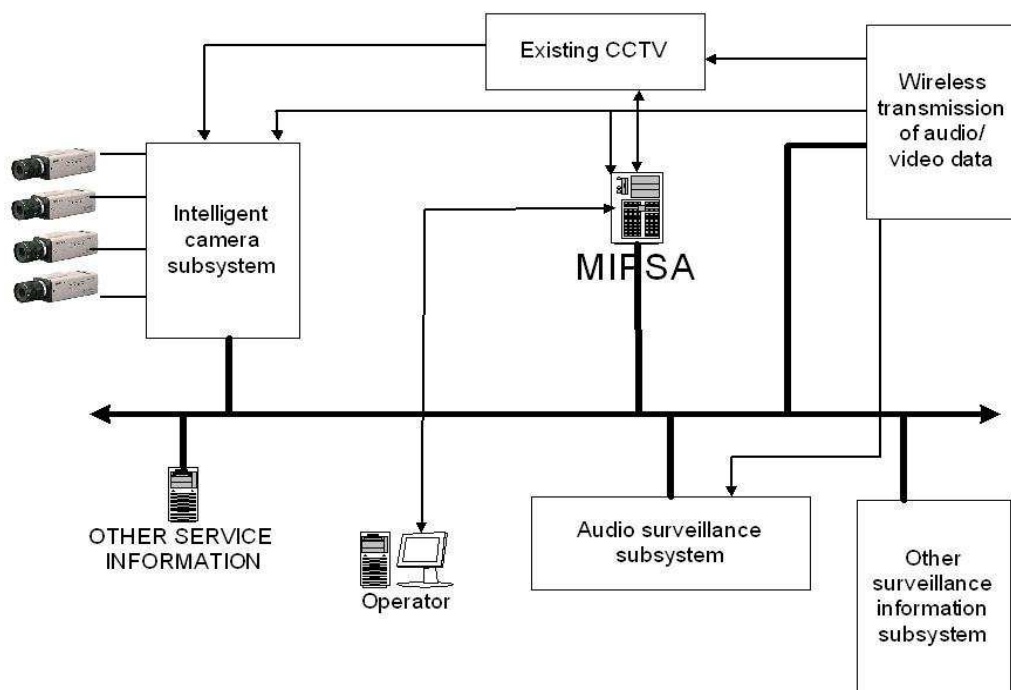


Figure 2-5. Architecture of PRISMATICA system (from [Valera and Velastin 2005b]).

PRISMATICA is built with the concept of a main or central computer which controls and supervises the whole system. This server thus becomes a critical single point of failure for the whole system.

In [Christensen and Alblas 2000] the authors report the design of a surveillance system with no server to avoid this centralisation, making all the independent subsystems completely self-contained, and then setting up all these nodes to communicate with each other without having a mutually shared communication point. This approach avoids the disadvantages of the centralised server, and moves all the processes directly to the camera making the system a group of smart cameras connected across the network. The fusion of information between “crunchers” (as they are referred to in the article) is done through a defined protocol, after the configuration of the network of smart cameras or “crunchers”. The defined protocol has been validated with a specific verification tool called spin. The format of the information to share between “crunchers” is based on a common data structure or object model with different stages depending e.g. if the object is recognised or is migrating from the field of view of one camera to another. However, the approach to distributed design is to build using specific commercial embedded hardware (called EVS units). These embedded units consist of a camera, processor, frame grabber, network adapter and database. Therefore, in cost-sensitive applications where a large number of cameras are required, this approach might be unsuitable.

2.6.6 Multi-camera surveillance system

As part of the VSAM project, [Collins et al. 2001b] presents a multi-camera surveillance system following the same idea as [Yuan et al. 2003], i.e. the creation of a network of “smart” sensors that are independent and autonomous vision modules. Nevertheless in [Collins et al. 2001b], these sensors are capable of detecting and tracking objects, classifying the moving objects into semantic categories such as “human” or “vehicle” and identifying simple human movements such as walking, while in [Yuan et al. 2003], the smart sensors are only able to detect and track moving objects. Moreover, the algorithms in [Yuan et al. 2003] are based on indoor applications. Furthermore, in [Collins et al. 2001b] the user can interact with the system. To achieve this interactivity, there are system-level

algorithms which fuse sensor data, perform the processing tasks and display the results in a comprehensible manner. The system consists of a central control unit (OCU) which receives the information from multiple independent remote processing units (SPU). The OCU interfaces with the user through a Graphical User Interface (GUI) module.

Monitoring wide areas requires the use of a significant number of cameras to cover as much area as possible and to achieve good performance in the automatic surveillance operation. Therefore, the need to co-ordinate information across cameras becomes an important issue. Current research points towards developing surveillance systems that consist of a network of cameras (monocular, stereo, static or PTZ (pan-tilt-zoom)) which perform the type of vision algorithms that we have reviewed earlier, but also using information from neighbouring cameras. The following sections highlight the main work in this field.

2.6.7 Co-operative camera systems

An example of co-operative camera system is Co-operative Camera Network (CNN) [Paulidis and Morellas 2002], which is an indoor application surveillance system consisting of a network of nodes. Each node is composed of a PTZ camera connected to a PC and a central console to be used by the human operator. The system reports the presence of a visually tagged individual inside the building by assuming that human traffic is sparse (an assumption that becomes less valid as crowd levels increase). Its purpose is to monitor potential shoplifters in department stores.

In [Michelsoni et al. 2003] a surveillance system for a parking lots application is described. The architecture of system consists of one or more Static Camera Subsystems (SCS) and one or more Active Camera Subsystems (ACS). Firstly, the target is detected and tracked by the static subsystems, once the target has been selected a PTZ, which forms the ACS, is activated to capture high resolution video of the target. The data fusion for the multi-tracker is done using the Mahalanobis distance. Kalman filters are used for tracking, as in [Xu et al. 2004].

In [Krumm et al. 2000] the authors present a multi-camera tracking system that is included in an intelligent environment system called 'EasyLiving' which aims at assisting the occupants of that environment by understanding their behaviour. The multi-camera tracking system consists of two sets of stereo cameras (each set has three small colour cameras). Each set is connected to a PC that runs the "stereo module". The two stereo modules are connected to a PC which runs the tracker module. The output of the tracker module is the localisation and identity of the people in the room. This identity does not correspond to the natural identity of the person, but to an internal temporary identity which is generated for each person using a colour histogram that is provided by the stereo module each time. The authors use the depth and the colour information provided from the cameras to apply background subtraction and to allocate 3D blobs, which are merged into person shapes by clustering regions. Each stereo module reports the 2D ground plane locations of its person blobs to the tracking module. Then, the tracker module uses knowledge of the relative locations of the cameras, field of view, and heuristics of the movement of people to produce the locations and identities of the people in the room. The performance of the tracking system is good when there are fewer than three people in the room and when the people wear different colour outfits, otherwise, due to the poor clustering results, performance is reduced drastically.

In [Marchesotti et al. 2003] a multi camera surveillance system for face detection is illustrated. The system consists of two cameras (one of the cameras is a CCD pan-tilt and the other one is a remote control camera). The system architecture is based on three main modules using a client/server approach as a solution for the distribution. The three modules are: sensor control, data fusion and image processing. The sensor control module is a dedicated unit to control directly the two cameras and the information that flows between them. The data fusion module controls the position of the remote control camera depending on the inputs received from the image processing and sensor control module. It is interesting to see how the authors use the information obtained from the static camera (the position of the recognised object) to feed the other camera. Therefore, the remote control camera can zoom to the recognised human to detect the face.

An interesting example of a multi tracking camera surveillance system for indoor environments is presented in [Nguyen et al. 2003a]. The system is a network of camera processing modules, each of which consists of a camera connected to a computer, and a control module, which is a PC that maintains the database of the current objects in the scene. Each camera processing module realises the tracking process using Kalman filters. The authors develop an algorithm which divides the tracking task between the cameras by assigning the tracking to the camera which has better visibility of the object, taking into account the occlusions. This algorithm is implemented in the control module. In this way, unnecessary processing is reduced. Also, it makes it possible to solve some occlusion problems in the tracker by switching from one camera to another camera when the object is not visible enough. The idea is interesting because it shows a technique that exploits distributed processing to improve detection performance. Nevertheless, the way that the algorithm decides which camera is more appropriate is performed using a “quality service of tracking” function. This function is defined based on the sizes of the objects in the image, estimated from the Kalman filter, and the object occlusion status. Consequently, in order to calculate the size of the object with respect to the camera, all the cameras have to try to track the object. Moreover, the system has been built with the constraint that all the cameras have overlapping views (if there were topographic knowledge of the cameras the calculation of this function could be applied only to the cameras which have overlapping views). Furthermore, in zones where there is a gap between views, the quality service of tracking function would drop to zero, and if the object reappears it would be tracked as a new object.

As it has been illustrated, in a distributed multi-camera surveillance system it is important to know the topology of the links between the cameras that make up the system in order to recognise, understand and follow an event that may be captured on one camera and to follow it in other cameras. Most of the multi-camera systems that have been discussed in this review use a calibration method to compute the network camera topology. Moreover, most of these systems try to combine the tracks of the same target that are simultaneously visible in different camera views.

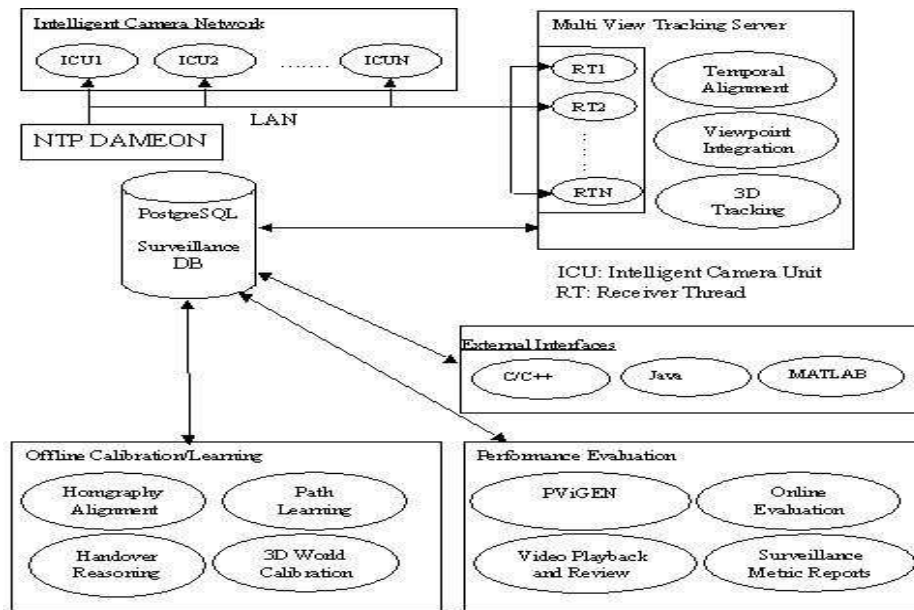


Figure 2- 6. The architecture of a multi-camera surveillance system (from [Valera and Velastin 2005b]).

In [Makris et al. 2004] the authors present a distributed multi camera tracking surveillance system for outdoor environments (its architecture can be seen in Figure 2- 6). An approach is presented which is based on learning a probabilistic model of an activity in order to establish links between camera views in a correspondence-free manner. The approach can be used to calibrate the network of cameras and does not require correspondence information. The method correlates the number of incoming and outgoing targets for each camera view, through detected entry and exit points. The entry and exit zones are modelled by a GMM and initially these zones are learnt automatically from a database using an EM algorithm. This approach provides two main advantages: no previous calibration method is required and the system allows tracking of targets across the “blind” regions between camera views. The first advantage is particularly useful because of the otherwise resource-consuming process of camera calibration for wide-area distributed multi camera surveillance systems with a large number of cameras [ADVISOR 2003], [Ronetti and Dambra 2000], [Pellegrini and Tonani 1998], [Velastin 2003].

2.7 *Distribution and communication*

In section 2.5.1 different techniques that have been applied to develop more robust and adaptive algorithms have been exemplified. In section 2.5.2 a review of different architectures of distributed surveillance systems has been presented.

Although the design of some of these systems can look impressive, there are some aspects where it will be advantageous to dedicate more attention for the development of distributed surveillance systems for the next years. These include the distribution of processing tasks, the use of new technologies as well as the creation of metadata standards or new protocols to cope with current limitations in bandwidth capacities. In [Berris et al. 2003] the authors propose the use of MPEG-7 as the standard format data for surveillance systems.

Other aspects that should be taken into consideration for the next generation of surveillance system are the design of scheduling control and more robust and adaptive algorithms. A field that needs further research is that of alarm management, which is an important part of an automatic surveillance system e.g. when different priorities and goals need to be considered. For example in [Garcia et al. 2000] the authors describe work carried out in a robotics field, where the robot is able to focus attention in a certain region of interest, extract its features and recognise objects in the region. The control part of the system allows the robot to refocus its attention in a different region of interest, and skip a region of interest that already has been analysed. Another example can be found in [ADVISOR 2003] where in the specification of the system, requirements of the system like “to dial an emergency number automatically if a specific alarm has been detected” are included. To be able to carry out these kinds of actions command and control systems must be included as an integral part of a surveillance system.

Other work worth mentioning in the context of large distributed systems has considered extracting information from compressed video [Norhashimah et al. 2003], dedicated protocols for distributed architectures [Ye et al. 2001], [Wu et al. 2001], [Almeida et al. 2002], and a real-time communications [Conti et al. 2002]. Work has also been conducted to build an embedded autonomous unit as part of a distributed architecture [Brodsky et al. 2001], [Saad and Smith 2003], [Christensen and Alblas 2000]. Several researchers are dealing with PTZ [Ng et al. 1999], [Marchesotti et al. 2003] because this kind of camera can survey wider areas and can interact in more efficient ways with the end-user who can zoom when necessary. It is also important to incorporate scheduling policies to control resource allocation as illustrated in [Jackson and Rouskas 2002]. Work in multiple robot

systems [Rybski et al. 2002] illustrates how limited communications bandwidth affects robot performance and how this performance is linked to the number of robots that share the bandwidth. A similar idea is presented in [Marcenaro et al. 2001] and [Valera and Velastin 2003a] for surveillance systems while in [Wu et al. 2001], an overview of the state-of-the-art of multimedia communication technologies and a standard is presented.

2.8 Summary

The growing demand for safety and security has led to more research in building more efficient and intelligent automated surveillance systems. This chapter has presented the state of development of intelligent distributed surveillance systems, including a review of current image processing techniques that are used in different modules that constitute part of surveillance systems and a short historical summary of surveillance systems. The main future challenge is to develop a wide-area distributed multi-sensor surveillance system which has robust, real-time computer algorithms able to perform with minimal manual reconfiguration on variable applications. Such systems should be adaptable enough to adjust automatically and cope with changes in the environment like lighting, scene geometry or scene activity. The system should be extensible enough, be based on standard hardware and exploit plug-and-play technology.

Such systems should be built through a combination of different disciplines being clearly needed such as computer vision, telecommunications and system engineering. Moreover, much could be borrowed from other fields such as autonomous robotic systems on the use of multi-agents, where non-centralised collections of relatively autonomous entities interact with each other in a dynamic environment. In a surveillance system, one of the principal costs is the sensor suite and payload. A distributed multi-agent approach may offer several advantages. First, intelligent co-operation between agents may allow the use of less expensive sensors and therefore a larger number of sensors may be deployed over a greater area. Secondly, robustness is increased, since even if some agents fail, others remain to perform the mission. Thirdly, performance is more flexible, there is a distribution of tasks at various locations between groups of agents. For example, the

likelihood of correctly classifying an object or target increases if multiple sensors are focused on it from different locations.

On the whole, the work on intelligent distributed surveillance systems has been led by computer vision laboratories perhaps at the expense of system engineering issues. It may be essential in the coming future for the development of distributed surveillance systems, to have available a well-defined framework to design distributed architectures firmly rooted on systems engineering best practice, as used routinely in other disciplines such as aerospace control systems. This is where we have concentrated on the work reported here. Therefore, the next chapter introduces the field of system engineering by focusing on design methodologies. Chapter 3, after presenting a brief historical review of different design methodologies used mainly to design real time systems, emphasises the discussion of design methodologies through the comparison between two design methodologies: Object Oriented methodologies and RTN.

3 Design methodologies for real-time distributed intelligent surveillance systems

3.1 *Introduction*

This chapter considers the methods that may be used for designing wide-area distributed intelligent surveillance systems. The design of these systems presents significant challenges, as they can be categorised as having distributed, concurrent, real-time and embedded properties. It is desirable, and indeed necessary, to apply sound systems engineering principles and practices in their specification, design and realisation in order to ensure that these complex systems operate as required (functionally and temporally). This chapter will outline the importance of design methods in the development of these systems. We start by summarising the conventional concepts required in software engineering to create the architecture for a system, followed by an overview of applicable software design methods. In this work, system architecture is defined as the underlying structure of a system (an abstract representation), i.e. the constituent components and the relationship between these various components.

Chapter 2 contained examples of real-time distributed intelligent surveillance systems that have been realised without applying any specific methodical design approach. This is only feasible when the systems are relatively small and simple. More recently, Object Oriented (OO) design approaches are starting to be widely used to design these systems. In this work, an alternative design approach called Real Time Networks (RTN) is considered for the design of such systems. The use of distributed object oriented technologies has led to the development of environments such as CORBA to deal with the design and integration of distributed systems. ADVISOR and PRISMATICA are some of the latest surveillance systems at the time of writing that use CORBA, as has been mentioned in chapter 2. However, our survey of methods indicates that OO/CORBA may not be suitable for this kind of system and that the use of Real Time Networks (particularly MASCOT 3/DORIS) may offer significant advantages, so the investigation of methods continues with a comparison of the conceptual models of OO/CORBA and RTN/DORIS.

To provide a historical context, the survey of design methods starts with a brief introduction to several well-known, but not strictly either OO or RTN, real-time design methods such as HOOD, Yourdon and Constantine, Jackson System Development (JSD), NRL and DARTS. Even though these design methods do not appear to have been reported as being used in the design of surveillance systems, they are presented here because they highlight some important characteristics of real time systems, which are also significant for the third generation of surveillance systems. This is followed by an introduction to OO design methods like ROOM, BOOCH, OMT and UML by giving a summary of the important features of each method. Subsequent sections will then introduce MASCOT 3 and DORIS in greater depth.

This chapter finishes with a comparison between the essential concepts of Real-Time Networks (as embedded in MASCOT 2-3/DORIS), and OO by discussing the abstract models, the communication models, functional aspects and performance aspects of both approaches. Chapter 4, through a case study, compares the structural design (architectural) level as well as run-time aspects of the two technologies: DORIS and CORBA.

3.2 *Design methods*

System or design system engineering may be defined as an interdisciplinary approach to build large and/or highly complex systems. This approach emerged around the World War II period especially in military systems. As stated in [Wikipedia 2001] “[...]While hardware engineering typically deals with just hardware and software engineering deals typically with the software, the systems engineer is responsible for seeing that the software properly operates on the hardware, and that the system composed of the two entities is capable of properly interacting with its external environment, especially the user, while performing its intended function...[]”. System engineering will be necessary in the deployment of future generations of surveillance systems, which will be larger and more complex than those being researched at present. This work focuses upon the software design phase in the development cycle of a system, defining and characterising such

systems and subsystems and the interactions among them. Therefore, to successfully build a third generation surveillance system requires traceable design methods capable of encapsulating the different levels of abstraction that need to be handled (from a global view of the system down to the detailed implementation aspects).

3.2.1 Design methods in surveillance systems

Real-time systems such as surveillance systems must respond to external events within required time limits, therefore timing requirements are very important in the design of these systems. To meet critical response time requirements, the system is often composed of a hierarchy of concurrent processes that communicate and co-operate to perform the overall function of the system. Other attributes that are important for designing real-time systems are performance, reliability, traceability and dependability⁴ constraints.

There are four important objectives that design methods for distributed real-time systems (e.g. surveillance systems) should accomplish. First, these methods need to be able to deal with concurrent operations in the system. The second objective is the capability of developing reusable software through modularisation and information hiding. The third objective is to be able to define the behavioural aspects of the system in terms of timing constraints and functional aspects. The last objective is the analysis of the operation of the design by determining its performance and the fulfilment of requirements.

OO has become popular in computer vision. Particularly, OO libraries, packages and programming languages like C++ and Java, and recently design notations like UML [Summer School 2004] have been common software tools to use to develop video-surveillance systems, see Table 3-1. The OO approach is used in this field because of the advantages claimed for OO in giving a modular approach to analysis and design. Another reason for using OO is because experts in video-based surveillance are mainly familiar with OO technology. Moreover, the design developers, who use object oriented techniques argue that OO techniques reduce

⁴ In computer science dependability is defined as: "[...] the trustworthiness of a computing system which allows reliance to be justifiably placed on the service it delivers [...]".

complexity within design and are suitable for prototyping. They also argue that OO implementations are flexible and easily accommodate changes so reducing maintenance costs, and that OO techniques can provide other important benefits like extensibility and reusability; design and analysis results can be stored in libraries that can be used again at a later time. Many design patterns exist that can be used to find already proven structures reducing development time.

Likewise, they believe that OO techniques improve stability when the requirements change, they have good support for reliability and safety concerns and also that, OO techniques have an inherent support for concurrency. In later sections, a discussion of all these properties, through a comparison with the RTN design method proposed in this work, is presented.

YEAR	LANGUAGES	PACKAGES
2002	C,C++,Java, Lisp, Matlab, Python, Tcl	Maple, Matlab, MathCAD, Mathematica, OpenGL, Statistica, VTR, WiT
2003	C, C++, Java, Lisp, Matlab, Python, Verilog, Mathematica, Maple	Matlab, OpenGL, VXL
2004	C,C++, Java, Matlab, Mathematica, Maple, Python, Perl,	Matlab, OpenGL, VTR

Table 3-1. Different software tools used in computer vision. This table is extracted from a Summer school in computer vision at Surrey University [Summer School 2004] and shows the wide range of tools currently available.

3.2.2 Classification by structural principles

There are several design methods for real-time systems such as structured design methods (SD), Jackson System Design (JSD), MASCOT 2-3, DORIS or Object Oriented Design (ODD). Each of these design methods emphasises a particular set of criteria to characterise the components of the system [Peters and Pedrycz 2000] e.g. procedural modules in structured design, concurrent tasks in MASCOT, or

objects in OOD. Each item in the following list emphasises a particular structural principle supporting a particular method, even though each structured principle is not necessarily exclusive to any one method.

- **Structured design:** is based on applying algorithm decomposition to break a large problem down into smaller steps. The design method realises a top-down decomposition of a system into modules.
- **Parnas Information Hiding:** is based on the decomposition of the software system into modules, where each module should hide a design decision that could change [Parnas 1972].
- **Jackson design:** is based on the idea that data structure is the key component of the software design. Hence, program structure, which reflects the problem structure, is best obtained from the consideration of the data structure [Jackson 1983].
- **Data-driven design:** consists of deriving the structure of a software system by mapping system inputs to outputs. This method has been applied to build a number of complex systems like management systems where there exists a direct relation between the inputs and the outputs of the system, but in which there is a little concern for time-critical events.
- **Event-driven design:** generates the architecture of a system by mapping an event to a response of external stimulus depending not only on the stimuli itself but on what happened previously on the system. Such systems are called reactive systems and usually are state dependent.
- **Object-Oriented design:** models the software system as a collection of cooperating objects. It can be considered as a bottom-up design approach [Graham 1994].

3.2.3 Current research in design methods

A formal software development method, which addresses the problem of producing embedded, real-time, distributed, dependable systems, is normally made up of three important phases: specifications, architectural modelling and implementation. The different activities which are involved in these phases are: requirements capture, architectural description for large/small-scale design, coding, testing, validation and

verification. Some significant ideas used in the current research into the development of such methods, and tools that help to create good software for large complex, systems are the following [Carnegie Mellon Software Engineering Institute 2005]:

- There is a wish to create methods that allow all stages of development to be carried out in a semi or fully automatic way (from the specification and requirements capture of the system to the implementation and building). One example is MDA (Model Driven Architecture) technology which is created by the OMG (Object Management Group). It puts forward the value of separating application logic from platform technology (i.e. CORBA, J2EE, .NET or Web Services). By having fully specified platform-independent models, it helps to insulate applications from technology evolution, supporting interoperability between platforms and applications and promoting the creation of a methodology to migrate from platform to platform.
- Another theme in current research, which was discussed in one of the invited talks in [ICSE2004], is centred on AOSD, which stands for Aspect Object Software Development. AOSD is a new set of software development techniques that supports the modularisation of system properties or ‘crosscutting concerns’ and their subsequent composition with other parts of software system. Typical crosscutting concerns are error handling, performance optimisation and design patterns.
- Following the importance in creating methods in order to develop good software, there is a need for computer assistance to help the software development processes: software tools, software development environments, and Computer Aided Software Engineering (CASE). Without such tools, the methods become too laborious for use on large complex systems.
- Emphasis on finding out exactly what the users of the system really want and need (requirements engineering) and validation and verification of the design using formal methods.

- Formal specification of the requirements of a system. There is an emphasis as well on trying to ensure in an automatic way that the software is free of errors (verification).

3.2.4 General criteria for comparing design methods

There are many different software methods at present. Each of them has its own advantages or disadvantages. It is not possible to identify a collection of tools and methods that are ideal in all circumstances; some development methods are not applicable to particular domains and can therefore be disregarded. Therefore, a relevant step is to establish and decide the most important criteria against which software methods could be compared. It is difficult to find a universal set of factors that allows comparison between any numbers of methods. However, comparison criteria applicable to the design methods relevant to this project and presented in sections 3.3 and 3.4 can be identified as:

- Underlying concepts resulting from the application of a given set of structural principles.
- Clarity of the proposed designs
- Integration of the different development phases
- How appropriate/inappropriate a method is for a given application area and scalability (good for small/large-scale design?)
- Run-time support
- Extent of tool support
- Formalisation of the designs

3.3 *Traditional design methods for real-time systems*

As has been mentioned before, during the last forty years, increasing emphasis has been placed on formalising the process of specification, design and implementation resulting in the development of several methods. In the past twenty years several software design methods have been developed. Up to the 1980s most design methods followed the structured design approach. The following sections highlight some significant design methods that have been used to design industrial real time

concurrent systems mainly based on control application. These include the Yourdon Systems Method, Jackson, NRL and DARTS.

3.3.1 The Yourdon Structure Method (YSM)

The Yourdon Structure Method (YSM) had its origins in structure methods back in the late 70s. YSM is the evolution of different Yourdon methods and it may be considered one of the most recent and extensive structure analysis design method for real-time systems [Yourdon and Constantine 1979]. The evolution of YSM starts with a first generation of Yourdon methods, which included structure analysis (contribution from DeMarco) where the system is modelled as a network of processes transforming data, and structure design (described by Constantine), which is based on a modular hierarchical system design approach. The main modelling components introduced were:

- Data Flow Diagram (DFD): represents the communication between processes or components. There are five different kinds of DFD components: data stores, instantaneous/continuous control processes and instantaneous/ continuous data processes.
- Context diagram: represents external communications, where one node represents the system and the rest of the nodes represent other systems in its environment.

Later on, system dynamics (using State Transition Diagrams), semantic information modelling and subject domain orientation were added to the second generation of Yourdon methods:

- Entity Relationship Diagram (ERD): uses Entity Relationship Attribute ideas and shows the conceptual structure of the data processes and each entity type (object type) and its relationship. It also shows the contents of data stores and dataflow.
- State Transition Diagram (STD): along with state transition table, it specifies the transitions that occur in the state machine of each continuous control process. Therefore, STD along with “minispecs”, which represent

the behaviour of a continuing data process, is used to represent the behaviour of the component of the system.

Major advances were achieved in 1984 by McMenamin and Palmer who introduced event partitioning. By using event partitioning, it is possible to define the functionality of the system in terms of event-response pairs, where each defined event has a response that the system has to complete. Thus, it is possible to draw a DFD fragment for each event-response pair. Finally, other advances were achieved by Ward and Mellor (1985) who introduced real-time specifications. YSM is configured by the structure analysis which is defined by ERD, context diagrams, DFD and STD and by the structure design, which is defined by DFD and STD.

3.3.2 Jackson System Development (JSD)

The Jackson System Development (JSD) [Jackson 1983a] [Jackson 1994b] is a design modelling (analysis) approach [Cameron 1986]. This method is based on the Jackson Structured Programming (JSP) approach, that is a program design method and that assumes that the specification of a program is defined by its data inputs and its data outputs. JSD designs model the system as a combination of functions, data and events. The first step in this method is to consider that the design models the real world and that the system is just a simple simulation of the real world. Each real world entity is mapped to a concurrent task, functions (executable operations) are added to this simulation of the real world to produce the system outputs. The basic components and the notation of this method are the following (the graphical notation can be found in Appendix A):

- Action Structure Diagram: this presents the modelled real world entity structure in the form of a time ordered sequence of the events received by it. The order of this sequence of events is described by three basic concepts: sequence, iteration and selection. The graphical notation is similar to Structure Diagrams in JSP diagrams.
- System Specification Diagram: this shows the structure of the different tasks that constitute the designed system plus the interfaces between them. These interfaces are represented by two different diagrams: data stream and state

vector notation. Data stream shows the message communication between tasks. State vector shows the internal data accessed by a task itself or accessed by other tasks. Only the task that maintains the data can write on it, the rest of the tasks can only read it.

In summary, the JSD method constitutes a complete development process for designing and building a system. This development process is defined in three phases: the modelling, the network and the implementation phase. In the modelling phase real world entities are identified and hence the entity structure diagram is configured. The attributes of each event received by an entity and the attributes of the entity itself are also defined. Therefore, for each entity a software model task is defined. In the network phase the message communication between the identified tasks is defined and also the internal data of these tasks. Thus, in the network phase, an initial network diagram is derived. Finally, at the implementation phase two issues are discussed: firstly, how these tasks are to be mapped onto a directly executable implementation version and scheduled, and secondly, how to organise and manage the data (the process state vectors).

The JSD development process imposes a clear and systematic mapping of the designed tasks to their implementation through the integration of these three phases. Concurrency is the main design concept in JSD, making it an appropriate method for designing concurrent systems. On the other hand, the concept of partitioning the system into subsystems or modules is not sufficiently developed, making it difficult to have a clear picture of the whole designed system. Besides, it may be quite arduous sometimes to represent complex timing behaviour of some entities in the system with the sequence events model in any of the diagrams.

3.3.3 NRL

The Naval Research Laboratory created the NRL [Gomaa 1993c] method to fulfil the perceived gap between software engineering from academia and software practising engineering coming from industry. In NRL “the system is viewed as a finite state machine whose outputs define the system outputs as functions of the state of the system’s environment” [Gomaa 1993c]. The main concept in this

method is to apply information hiding and modularisation to the design of systems. Therefore, modules that represent different parts of the system are designed following the information hiding concept [Parnas 1972]. Thus, design decisions of the module that are expected to be changed are hidden from the rest of the modules. Therefore, if a change, e.g. of an attribute in the module, is made, the rest of the modules are unaffected by this change. The visible parts of the module are defined by the abstract interface specification. Furthermore, the modules are designed and implemented to be stored in libraries and therefore to be reused. NRL organises the information hiding modules (IHM) in a tree-hierarchical structure to overcome the complexity of the design of large-scale systems where there is a substantial number of modules, making it easier to trace the modules through subsequent development phases of the project. The software in NRL is structured in three main orthogonal structures:

- **Module Structure:** this is achieved by information hiding. There are three different categories of IHM: Hardware and Behaviour hiding modules and Software decision modules. Two different modules define the hardware hiding module: extended computer modules and device interface modules.
- **Uses Structure:** shows the executables subsets of the software.
- **Tasks Structure:** determines the number of activities required at run-time, contributing to more flexible scheduling. The tasks are determined by analysing the operations provided by the module.

The NRL method has no graphical notation and the designs are expressed through tables which are used to summarise the design decision and the IHMs. The states of the system also are represented through what are called state transition tables. The main steps in the development process associated to this method are:

- **The establishment of the requirements of the system:** in this phase the method considers the specifications of the system as a white box.
- **Module structuring phase:** in this phase the module structure is identified from the specifications i.e. the hardware hiding module, the behaviour hiding module and the software decision module. Once the module structure

is identified the abstract interface and the operations required by this module are defined.

- Module internal structures: in this phase further decompositions of the module may be done and the structure task is established and therefore the implementation of the design.

In conclusion, NRL emphasises the information hiding concept in the design and the creation of modules that are relatively modifiable and reusable. Although the complexity of designing large scale of systems is reduced by allowing hierarchy in the design of these modules, it is difficult to have a full picture of the whole system. Besides, in NRL there is not a clear definition of the stages of the development process, even though there exists a clear differentiation between requirements and design. Moreover, it is not straightforward to go from the specification document to task structure.

3.3.4 ADARTS

Ada based Design Approach for Real Time Systems (ADARTS) is a refinement of a previous method called Design Approach for Real Time Systems (DARTS) to support Ada based design [Gomaa 1984a], [Gomaa 1989b]. The DARTS and ADARTS methods originated to tackle a common problem in industrial real-time systems development, namely that the majority of design methods do not take into account the important characteristic that real-time systems usually consist of a group of concurrent tasks. ADARTS combines NRL and Object Oriented Design (OOD) to design a system by applying the module criteria from NRL to identify IHM structures. Moreover, ADARTS then uses the object structuring criteria from OOD to identify concurrent tasks and defining their interfaces, which then define the communication and also the synchronisation interfaces between tasks.

Therefore, the basic components in ADARTS are the IHMs, which are defined through information hiding module structure criteria, and the concurrent tasks which are defined through the task structure criteria. Both criteria are applied to functions (transformations) on the data/control flow diagrams. Each transformation is perceived as a dynamic structure if the function is executed under the control of a

task. The transformation is perceived as a static structure if the function is related to operations into a module based on hiding structure criteria. The steps that define the development process in ADARTS are the following:

- Firstly, the behaviour of the system being designed is described through what is referred as structure analysis specification for real time systems (RTSA). The RTSA notation (please refer to Appendix A) is used for modelling the problem domain. At the end of this step hierarchical control/data flow decomposition is performed using state transitions, data flow and system context diagrams.
- The second step consists in identifying concurrent tasks by applying task structure criteria. Inter-task communications and synchronisation interfaces between tasks are also identified. Tasks interfaces represent loosely/tightly coupled message communication, events synchronisation or IHMs.
- Once the concurrent tasks are defined the next step is to identify the modules by applying the hiding module structure criteria.
- At this stage the Ada support tasks and the Ada task interfaces are added.
- The last step consists in defining the interfaces specifications for the tasks and the modules. These specifications are the external view of the task or the module which they represent.

3.4 Survey of some important OO design methods

From the beginning of the nineties, most of the new design methods have followed an object oriented approach. Since 1988, more than twenty different object oriented methods have been developed [Graham 1994]. OO approaches can be differentiated between those which are based on Analysis (OOA) and those which are based on Design (OOD). This work focuses on the underlying structural principles (i.e. conceptual structure) of the model. The conceptual structure of OO models is usually based on the concept of “class” (abstract type) and the use of component called “objects” which are instances of class types. In addition to the concepts of class and object, there is a property called inheritance, universally important in OO methods, which may be applied to classes or objects. This will be discussed in later

sections. The application domain plays an important role in the decomposition principles expressed in OO methods (e.g. “use case” diagrams).

The majority of OO approaches tend to have been published in conferences or workshops papers, and relatively few are comprehensively documented in books or manuals. The following subsections present some of these design methods like ROOM, BOOCH, HOOD, OMT and finally UML, which although is not a design method *per se*, is likely to become a standard modelling language based on an OO approach (for further details of these OO methods, please refer to the References and to Appendix A). The survey here presents BOOCH and OMT because they are considered to be the main methods from which UML has originated. The HOOD and ROOM methods are summarised because they are examples of OO based methods used specifically to create real-time, distributed systems. Moreover, the ROOM design method is presented to show an example of an OO method which has the semantics to express and execute its models through a virtual machine.

3.4.1 ROOM

Real-time Object-Oriented Modelling (ROOM) is a modelling language which was developed to design real-time systems. Abstraction is the driving mechanism in ROOM, which is handled by recursion or functional decomposition, incremental modelling and reuse. ROOM notation is consistent through the three main phases of the design process: specification, design modelling and implementation. The ROOM method supports hierarchical structure modelling. The object paradigm in ROOM is based on defining the object as an encapsulated entity that communicates to other objects through its interface. Thus, ROOM represents models by classes which are incarnated by objects. Even though ROOM supports inheritance relationships between classes, it does not support multiple inheritance, using aggregation instead, which is a “part-of” relationship. Object communication is based on a message passing port-to-port connection. Its metamodel consists of six basic elements (to see the graphical notation, please refer to Appendix A):

- Actor: it represents an active object which may have its own thread of execution (similar to an agent object) and is used to define high-level structures of the system. Actors are created by actor class definitions.
- Passive Data Object: it is an element complementary to an actor as its functionality has to be activated through actors. For this reason, it is defined as passive. Passive Data Objects are also defined as data objects and represent an abstract data type. They are created from a data class definition. They are not part of the high-level structure of a system and only exist in the context of execution: only the active object (actor) that encapsulates this data can access it.
- Message Object: it is used by Actors to send information via a communication service. A message is a data type containing a signal and a priority attribute and an optional message data object.
- Protocol: messages are grouped and structured by a protocol class definition. A protocol element is defined by a signal that identifies each message in addition to possible Passive Data Objects that are sent or received by the message. It is also defined by characterising a given direction for each message sent (in or out). There are also two optional specifications: the validity and the quality of service of the message. The Protocol is defined by one of the Actors in the communication. In the ROOM model the concept of data sharing between concurrent threads (Actors) does not exist. The Passive Data Objects are copied and sent by messages through the interface of an Actor.
- Interface element: it allows the communication between Actors. There are three types of interface elements: Port, Service Access Point (SAP) and Service Provision Point (SPP). The interface of an Actor is defined by ports which are used for communication. Ports define the set of messages, which are part of a protocol and are constituent of the Actor's interface.
- Behaviour element: this defines the behaviour of an Actor, which is part of its specification, through a ROOMChart. A ROOMChart is basically a finite state machine and a variant of an extended state machine. It supports nested hierarchical states.

The structure concept described in the metamodel of ROOM is simple and easy to understand. It consists of active software entities called Actors and passive software entities called Passive Data Objects. Actors communicate using port-to-port connection. These entities are modelling in a hierarchical manner, thus some elements are grouped in one layer and communicate with other higher or lower level layers. Because ROOM is a design method based on an object paradigm approach, the software entities are represented as objects. It is claimed in [Erik Wyke 2000] that in ROOM there is consistency in the development process. At the design phase, the objects are encapsulated in a shell that defines an interface to communicate with the rest of the elements and at the operational phase all the elements are treated as executable parts of the whole model.

Therefore, objects are considered as programs written in a high-level language that can be executed. The run-time support is provided by a ROOM virtual machine that provides the services that the system needs and where it is possible to execute the design model. The operating system kernel does not necessarily support thread synchronisation. ROOM is a non-shared memory model, only the object that encapsulates the data can access it. Because data is not shared, only copies of this data are shared. The scheduling policy is based on assigning priorities to events rather than tasks using a pre-emptive approach. Therefore, ROOM is appropriate for modelling and implementing real-time event-driven systems (in these systems priorities are assigned to events instead of tasks). However, many of the scheduling policies used in real time systems are based on task (threads) rather than event-priorities. One of the drawbacks as highlighted in [Erik Wyke 2000] is that implementation decisions of structural parts have to be taken before the design starts. Moreover, even though some behaviours are well captured through ROOMCharts, how to express explicit timing requirements is not straightforward.

3.4.2 BOOCH

The Booch design method [Booch1991], [Booch1994] represents a software system by means of class diagrams and the behaviour between objects by state transition diagrams. The metamodel is based on four static or structural diagrams (class diagram, object diagram, module diagram and process diagram) and two additional

diagrams: state transition and sequence diagrams, which represent the dynamics or behaviour of the designed system. These diagrams may represent parts of the architecture design of the system and furthermore, the architecture design of the whole system. Each of these diagrams has a template component associated with it, where the important aspects of the components in the diagram are captured. For example, the class diagram has an associated class template. The graphical notation of all these diagrams is presented in Appendix A. The following list summarises the purpose of each diagram:

- Class diagram: it represents the classes and their relationships. Classes can be parameterised and there is a distinction between class, superclass and metaclass which is expressed in the class template.
- Object diagram: it represents the objects that compose the system and their relationships. A single object diagram is like a snapshot in time of a transitory event because objects can be created and discarded at run-time.
- Module diagram: this is used to illustrate the number of classes and objects in the module. There is a special module diagram called subsystem which is almost like a class diagram (please refer to Appendix A). The difference lies in a conceptual distinction: a class diagram is classified as a (“kind of”) hierarchy; all of the classes which appear in the same class diagram have an inheritance (“type of”) relationship with one another. By contrast, a subsystem is categorised as (“part of a”) hierarchy, because the classes appearing in the diagram have an aggregation (“part of”) relationship with one another.
- Process diagram: it shows how the processes (not objects) are going to be mapped to processors. This diagram represents a part of or the whole of the physical architecture of the system.
- State transition diagram: this uses a state chart-like notation and represents the events that cause a transition of a state and the actions that result from that state change.
- Sequence diagram: this illustrates the interactions between objects occurring at run-time, e.g. if one object asks for data from another object in order to continue working, then, this cooperation is illustrated by the sequence diagram.

As discussed, in the Booch method the software entities are conceptually decomposed into objects represented by class diagrams. The behaviour of these entities is shown in object diagrams and the communication is represented by sequence diagrams. Even though the Booch method is expressed using six separate diagrams, the full notation includes a large number of icons and symbols which may produce an unclear graphical design.

A systematic way to go from the specification to the design and finally to the implementation and construction of the system does not seem to exist yet. A class diagram may be seen as a representation of the abstract entities that will compose the system, which have been captured from the specification. At the same time the class diagram is seen as a design part of the system. The process diagram may give an approximate idea of the physical architecture of the system but in an informal way. Furthermore, run-time facilities are not carefully explored although [Booch1991] mentions that the scheduling policies can be designed using the timing diagrams.

3.4.3 Rumbaugh (OMT)

This subsection summarises a design method which was quite popular some years ago, created by Loomis in 1987, and popularised by James Rumbaugh in 1991 [Rumbaugh et al. 1991]: the Object Modelling Technique (OMT) (please refer to the References and to Appendix A for more information). OMT has some influence from structure methods and has a detailed notation. OMT basically consists of three phases:

- Analysis: produces three models: the object, the dynamic and the functional model. These models progress from the initial requirements specifications. The object model is similar to an UML object diagram, as described in the following subsection, and is a diagram illustrating the relationships among objects and classes constituting the system: please refer to Appendix A to see the OMT object diagram and its notation. The dynamic model presents the state transition diagram for each object. Subclasses inherit the state

transition diagram of their superclasses adding states and transitions. The functional model is a dataflow diagram used at a high abstraction level with passive objects as data stores.

- System design: subsystems, tasks and processes composing the system are described and the concurrency between each other is identified using the dynamic model.
- Object design: the object model is completed using the information extracted in the functional and in the dynamic model. Some implementation aspects such as the design of the algorithms, packaging and documentation are included in this phase.

3.4.4 HOOD

The Hierarchical Object Oriented Design (HOOD) method was developed in 1987 by CISI, CRI A/S and Matra Marconi Space under a European Space Agency project to build large real time systems [HOOD 1986a]. HOOD was developed as an integral part of a full development process for a software system. Therefore, it aimed to support all of development from requirements analysis through to integration. It claims to bring the possibility of parallel development of different parts as well as automated code generation and testing, and also a post-partitioning support. It is stated in [HOOD 2004b] that HOOD is a design method which helps the designer to partition the software into different modules with well defined interfaces by decomposing the software units hierarchically. These units are based upon identification of objects, classes and operations. HOOD [HOOD 1989a] integrates principles from other approaches such as Abstract Machines [Diehl et al. 2000], and also assimilates some OOD concepts from the Booch method and some hierarchy principles found in General Object Oriented Design (GOOD) [Seidewitz and Stark 1986] to enforce its hierarchical structure design such as the use of senior hierarchy concepts, which deal with the organisation of several objects into “layers” which define (each of them) a virtual machine. It incorporates functional approaches by supporting modular programming, object based approaches by supporting encapsulation and object identity, and object oriented approaches by supporting inheritance properties. The HOOD method comprises textual and

graphical notation and its main concepts can be summarised as follows (for the graphical notation of the components please refer to Appendix A):

- Object: this component is equivalent to a module of software. The formalism of its graphical and textual notation is supported by what is called Object Description Skeleton. The static HOOD Object properties are the object interface, which is the visible part of the object, and the internals of the object, which are the hidden part of the object. The interface defines the operations provided by the object with the associated parameters and resources. In addition, it provides types and the operations required from other objects. The object internals define the implementation of the provided interface.

The communication between objects is made through service requests, by executing operations (similar to procedure calls). A client object requests an execution of an operation and the object that performs the execution is called a server object. Threads, which have a control flow, activate operations on objects. There may be several threads executing simultaneously in the same object. One key feature in HOOD is that these threads of execution can be specified one by one using the concept of constrained operation, which can be defined through state constraints, concurrency constraints, protocol constraints or time constraints. Moreover, HOOD defines the dynamic properties of an object by describing the effects on a client object:

- Sequential execution: control flow, which is executed within the internals defined in OPERATION Control Structure (OPCS), is transferred from the client to the server directly. Once the server is finished control is transferred back to the client.
- Concurrent execution: control flow is not transferred directly to the server. On the server side, the OBJECT Control Structure (OBCS) protocol deals with the incoming requests from the clients and the execution of these requests depends on its internal state as well as on the control protocol.
- Class: is defined as an abstract data type of an object. The difference in HOOD between type and a class lies in the fact that a class may inherit other

properties and operations from other classes, whereas type cannot inherit properties or operation from other types. Classes are object oriented elements that define the shared-code for all the instances of the class.

HOOD defines the concept of a virtual node to deal with distributed systems. The virtual node symbol represents a cluster of HOOD objects for a given HOOD design tree which can be allocated into software blocks and executed in a given physical local or remote memory. HOOD defines a symbol to represent the use relationship which indicates required services from one object to the other, thus the use relationship defines client-server associations. To represent the top-down design that is defined in HOOD as the decomposition of the parent object into child objects, providing the same functionality to its child objects, HOOD defines an “include” relationship symbol which represents these hierarchies broken down into relationships between objects.

In summary, the concept of structure in HOOD approach is basically guided by a separation of concerns. Each software partition entity that is represented by an object has well defined interfaces and data modelling and its own description of functional and behavioural aspects, promoting the reuse of software modules and the support to repartition the software. To express these concepts HOOD has a set of formal textual and graphical notations. One of the differences between HOOD and other OO design approaches resides on the reduction of number of symbols used to express the design, making HOOD designs clearer at first sight and easier to use from the designer point of view. The approach of hierarchical decomposition of the modules, where high level structures are refined into more detail by other structures, makes this a feasible approach to deal with the complexity that a large real-time system has by nature, because the designer does not have to deal with all the details of the system at once.

A number of rules can be applied to HOOD design which can be reviewed by automated tools to check consistency and completeness. These rules can be categorised into “definition” rules, “methodological” rules, “usage” rules and “code generator” rules. For example, the “definition” rules (i.e. include relationship rules, use relationship rules, break-down rules, operation rules or consistency rules) are

statements to check the basic definitions and properties of the elements of the method. The methodical rules check for the completeness in the design phase (i.e., to check and ensure consistency between representations). Therefore, HOOD supports a development process that encompasses the different design phases and helps ensuring consistency and traceability in the design.

Run-time support in HOOD is approached by applying the constrained operation concept which, as it has been outlined before, allows the independent specification of the functional and temporal behaviour of each thread of execution allowing therefore the possibility of scheduling analysis. However, in [Burns and Wellings 1994], the authors state that HOOD has a lack of explicit support for common hard time abstractions.

There are several tools available from different vendors for designing systems using HOOD approach. As explained in [HOOD 1986a] “HOOD was designed right from the start with consideration for tools support”, therefore the notation and rules have been designed for being produced and reviewed by tools. As discussed, HOOD is a design method derived from industrial experience and it has been considered for serious real-time software development even though it applies the concept of inheritance to design, which will be discussed later on, and this may contribute to difficulties in the traceability, performance and testing of the designs.

3.4.5 UML (Unified Modelling Language)

UML is a modelling language rather than a method, which is the OMG’s most-used specification for modelling application structures, architectures, data structures and business processes. Even though UML is a modelling language and not a design method it tends to be considered as the successor of OO design and analysis and it can be seen as being the result of the combination of the Booch, OMT and Jacobson OOSE (Object Oriented Software Engineering) approaches. It is expected to be the standard modelling language in the future and also provides the key foundation for OMG’s Model-Driven Architecture, a technology which has been introduced in section 3.2.3. At the time of writing, its current developed version was UML 2.0. For a more detailed description of this modelling language please refer to [Booch et

al. 2000]. UML defines a notation and a metamodel. The metamodel is one of the layers of a four-layer metamodel architecture which are the following [UML 2 Metamodel 2005]:

- Meta-metamodel
- Metamodel
- Model
- User objects

The UML metamodel is defined as an instance of a meta-metamodel which defines the language for specifying a model (i.e. class, attributes, operations and components). The complexity of the UML meta-model is managed by organising it into logical packages.

The notation is the syntax of the modelling language and is represented by the graphical components in the models (to see the graphical notation for these components please refer to Appendix A). The principal elements that compose the modelling language can be classified into two groups based on whether they model the structure or the behaviour of a system. They are introduced in the following list:

Structural modelling

- ***Actor***: it represents a set of roles that a user plays with respect to the system. In the metamodel, an actor is a subclass of Classifier; it has a name and may communicate with a set of Use Cases.
- ***Class or static structure diagram***: presents the UML metamodel and illustrates the static structure of the model such as classes and types and their internal structure and also their relationships, but not the temporal information.
- ***Object diagram***: is a schema of specifications of instances, including objects and data values. It presents an instance of a class diagram at a point in time.
- ***A state chart diagram***: shows the behaviour of an interaction or instances such as an object. It illustrates possible sequences of states and actions from which these instances react to discrete events such as signals. In other words, it represents a state machine diagram.

- **Component diagram:** illustrates the dependencies among software components and also describes the classifiers that specify them and the artifacts that implement them, e.g. implementation classes, source code files, executable files or scripts.
- **Package diagram:** illustrates the packages classes and the dependencies among them.
- **Composite structure diagram:** shows the composition of different elements, which are going to be run-time instances that are interconnected through communication links.
- **Deployment diagram:** a graph of nodes connected by communication associations. It describes the configuration of run-time processing elements and the software components, processes and objects that execute on them. Components that do not exist at run-time do not appear on this diagram.

Behavioural modelling

- **Use Case diagram:** a schema of actors, a set of use cases, perhaps some interfaces, and the representation of relationships between these elements i.e. associations and generalisations between actors and the use cases, and generalisations, includes, aggregations and extensions (variations of the main success scenario) between the use cases. In essence, a use case may be considered a technique for capturing the functional requirements of a system.
- **Activity diagram:** is a special case of a state chart diagram where states represent the performance of actions or “subactivities” and the transitions are triggered by the completion of the actions.
- **State Machine package:** may be used to model discrete behaviour through a finite state transition system. There are two types of state machine packages: the behavioural state machine and the protocol state machine.
- **Timing diagrams:** represents changes of the states or other conditions of a structural element over time.
- **Interaction diagrams** (sequence or collaboration diagrams): describe how groups of objects collaborate and interact. There are two kinds of interaction diagrams: the sequence diagram and the collaboration diagram. The sequence diagram presents the explicit sequence of communication between

objects making it more suitable for the design of real-time systems and complex scenarios [UML 2 Metamodel 2005]. The collaboration diagram represents an interaction established around the roles and also represents their relationship.

It is important emphasise that for the design of Real-Time systems, UML provides other components that are not mentioned on the previous list. These components are called Ports (capsule) and Connectors and are used in composite structure diagrams. Ports components are linked by Connectors and isolate the component which makes it independent of its environment. The Connectors are not necessarily a bundle of software on its own but are protocols to which Ports conform. In [UML 2 Metamodel 2005] it is pointed out that a good design decision is to design these ports as objects on their own to guarantee the decoupling between Ports and Connectors.

3.4.5.1 Discussion

With regards to the structure concept and the UML designs formalism, the four-layer architecture which defines UML language architecture is claimed in [UML 2 Metamodel 2005] to be an architecture for defining the semantics required by complex models. The UML graphical notation shows a wide range of different definition components and relationships, giving freedom for modelling a system. On the other hand, even though the graphical notation is presented in a clear way, to have such a wide choice of diagrams can make the design decisions difficult. There is no mapping of different diagrams in a model which induce inconsistency in the modelling design. At the same time, some of the semantics are hard to understand at first sight and can lead to different interpretations of the same model.

Hierarchy in the composition of models is not fully explored; therefore it is difficult to represent the design of a large system. In [Christensen and Alblas 2000] they used UML to represent the design of small distributed surveillance system (with three cameras). Even though the internal structure of different system components is well-defined through UML diagrams, there are no diagrams that illustrate the integration of these components to build the whole system. Therefore, it is not possible to see the design of the whole system at once. In [Hull et al. 2004] even

though the authors presented a mapping in UML of the internal structure of different components of Real Time Networks, presented in next section, they do not illustrate any mapping in UML of a whole Real Time Network system design. Therefore, there is no diagram that presents the whole system at once.

At the time of writing, there is a large amount of research conducted to polish and extend UML because it is supposed to be the standard modelling language of OMG. Furthermore, it is also considered one of the kernels of OMG's Model Driven Architecture. Thus, there is a large range of tools to support the use of UML to design although there is not an official list, e.g. ARTiSAN's Real-time studio or Metamatrix Metabase Modeler (for more information please refer to [UML 2006]). There are also some UML tools like Rational Rose that systematically creates UML models from OO languages like C++ and UML CASE tools like Visio Studio or Artiso Visual Case.

A compromise exists between clarity of the models and formal specification languages. The current UML description is not a completely formal specification language but it is easier to understand. The UML specification is based on a combination of languages: a subset of UML components, and OCL and natural language to describe the abstract syntax and semantics of the full UML. The syntax is described in UML in *Abstract Syntax* [UML 2 Metamodel 2005]. The static semantics of a language are defined in *Well-Formedness Rules* [UML 2 Metamodel 2005] and the dynamic semantics in the *Semantics* section of UML specification [UML 2006]. The static semantics are defined as a set of invariants of an instance of the metaclass, and each of them is defined by an OCL expression with an informal explanation of the expression.

UML has been considered to be an appropriate modelling technique for business applications. It is claimed in [UML 2 Metamodel 2005] that this technique is suitable for real-time applications because behaviours of different components can be defined through state, activity, collaboration and sequence diagrams and further real-time extensions like defining queuing orders or priority mechanisms based on different approaches like ARTiSAN's or through the definition of Ports and Connectors. However, these behaviours and actions represent behaviours between

objects or activities and not between different parts, components or modules that compose the system. Thus, there is not a clear overall picture of the behaviour of the system. Even though there is a useful diagram called the deployment diagram, that shows all the components that will exist at run-time, there is a lack of support for scheduling techniques.

The deployment diagrams just show “what” and “who” will exist at run-time but they do not show “how” these components will interact at run-time, a property that is important in a real-time system. Furthermore, as it is discussed also in [Erik Wyke 2000], even though UML is a good technique to represent the structure of the data of the system, it has a lack of representation of the flow or the quantity of data information, which are directly linked to the performance of the system. Besides, even though UML tries to represent time using different type of diagrams e.g. timing diagrams or sequence diagrams, these representations are based on expressing the internal time of the element on its own or time as a sequence interaction between elements, but there is no explicit time interaction between the element and its environment.

3.5 Real Time Networks (RTN)

RTN is strongly based on a shared data model. This approach consists of conceiving a system as a network of active⁵ (internally sequential) processing components (called activities) interconnected through dedicated passive elements (called IDAs). Activities cannot distinguish one network context from another. Therefore, this provides reusable software components and allows the inclusion of these activities, without change, in special test systems for prototyping or integration testing proposes, if necessary in execution environments, which differ from the final target configuration. The network is *per se* a spatial form of representation (i.e. the activities may be mapped in several processors) and so it may be suitable for use in a wide range of distributed application areas [Simpson 1992d]. However, it is possibly best suitable for real-time embedded systems where the software has a degree of complexity and is highly interactive.

⁵ Each active process has its own thread of control

3.5.1 MASCOT- RTN principles

MASCOT (Modular Approach to Software Construction Operation and Test) is an extension of Real Time Networks [Phillips 1967]. MASCOT is a software design method for the design and implementation of large real-time concurrent systems. Ken Jackson and Hugo Simpson originated its essential concepts at the UK Royal Signals and Radar Establishment (RSRE) during the period 1971-5 [Bate 1986]. The first technical notes (MASCOT 1) were published in 1978 [IECCA and MUF 1978a]. The first official handbook was issued in 1983 (MASCOT-2) [IECCA and MUF 1983b]. Further refinements and extensions were continued until the official standard for MASCOT-3 [IECCA and MUF 1987c] was published in 1987. The RTN approach in MASCOT/ DORIS differs from most other design methods, because design solutions are expressed in terms of a set of concurrent components, which work independently and interact through explicitly identified data areas. This structures the logical design and provides an early natural temporal partitioning of the different components that compose the whole system. It provides the means for temporal and physical (spatial) decoupling, aiming at maintaining at the same time predictable temporal properties. The proponents of the MASCOT approach for software development believe that it [IECCA and MUF 1983b]:

- Defines a formal method (i.e. every step in the process to obtain the final software structure is clearly defined) of expressing the software structure.
- Imposes a disciplined approach to design by ensuring a close correspondence between functional elements in design and constructional elements for system integration.
- Provides a highly modular structure supporting a program acceptance strategy based on the test and verification of single modules and group closely related modules.
- Provides for a small easily implemented executive for execution of the program at run time.
- Provides for a straightforward and flexible method for system building.
- Can be applied through all stages of the software life of the project.

The dominant RTN principle embodied in MASCOT is that the flow of data through the system is controlled solely by a set of concurrent software process

[IECCA and MUF 1978a], which means that MASCOT uses the concept of data flow network between concurrent processes that constitute the network, as the means for expressing software structure. It emphasises the importance of structure, data, communication and the production of systems as a real-time networks, as opposed to programs. Moreover, it is also stated by its proponents that MASCOT addresses the important issues of dependability, flexibility (by allowing the designer to easily accommodate the continuous changes that may occur during the design process, into the system design) and re-use (by using template facilities which are discussed in other sections here). The MASCOT method provides a design language (textual form) and a graphical notation (MASCOT network diagram), which are two complementary forms for representing the network architecture and controlling an evolving design structure. The method provides the visibility necessary to support management and control of the design during development and subsequent maintenance. This visibility can be achieved by the use of CASE tools to process the design, supported by a database to hold the design details providing the status progression feature of MASCOT.

MASCOT aims to support strong design features; safety critical functions are protected from interference and corruption by enforcing strong partitioning of the overall design task. MASCOT allows distribution of system functionality to be represented, by explicitly providing small independent units of execution which can be identified early on (through the need to define activities). These units are suitable for distribution in a multiprocessor environment, which can be analysed for their temporal properties in terms of information propagation.

3.5.2 The MASCOT network design

The Real-Time Network approach of MASCOT applies concurrency as a direct solution of the problem. The main distinguishing feature, based on the RTN principles, is the explicit recognition of Intercommunication Data Area (IDA) components located between concurrent processes, which are known as ACTIVITY components. An activity is an active element, which is the fundamental processing element in a MASCOT network. It is a single sequential processing thread that can be scheduled independently so that, conceptually, all activities may be executed in

parallel and concurrently [Simpson 1986a]. An IDA is a passive element, which is used either for independent information storage or for information transmission. An IDA is effectively an encapsulated data type whose detailed physical representation is hidden from its users. An IDA component allows several independently scheduled single sequential program threads (activities) to be simultaneously active or temporarily suspended. The IDA safeguards the integrity of data by using the minimum amount of mutual exclusion needed to avoid critical data clashes. The IDA maintains the propagation of data in the network by providing cross-stimulation between activities.

MASCOT takes requirement specifications obtained by other means as its starting point [Simpson 1986a]. MASCOT data flow networks should be static. Activities should not be created dynamically and the system network should remain invariant at run time to avoid hazards in terms of unconstrained resource demands and non-deterministic timing. However, it seems that special measures can be used for those applications that cannot be implemented without dynamic network creation [Mustafa 2000]. Even though these measures are not published, one of them (taken from a private conversation [H. Simpson2005]) may consist in creating and building a new component, and then it is inserted into the network. Once it is established, the old component is removed from the network. Another measure may be the use of a protocol that is discussed in section 3.6.4.1, called Remote Thread Invocation (RTI), which activates a thread at run-time. MASCOT assumes that the software system is being designed for a particular virtual machine called the MASCOT kernel, and the implementation of this virtual machine on a specific computer or a set of computers (depending on the configuration) is a separate problem. The kernel is a set of procedures, constants and data-types, which provides the run-time executive level facilities for purposes such as process scheduling, synchronisation, interrupt handling, execution control and monitoring. These facilities are defined in a context interface specification.

During development, the structure of the application software evolves as a set of interconnected but independent components that make no direct reference to each other. The components in MASCOT such as IDAs or activities are defined as templates during the design process. The idea of a template idea in RTN could be

associated to the idea of a class in OO. A component is an instantiation of a template. Specifications define an interface (a type of connection) so that connections between components are established from the corresponding interface specification. Components contain definitions which describe a set of data-types and named constants. MASCOT templates are reusable so they can enable the creation of multiple components derived from the same template or the creation of the same component in different execution environments (different system designs). The definitions of the textual forms of the design structure (modules) are inserted into a MASCOT database, which supports the development process. The textual forms of modules (template) are subdivided into three parts:

- *Name*: defines its class (note that it does not have the same meaning as “class” in OO) and it gives the template a unique identification.
- *Specification*: consists of the information required for components of that type to be included in inter-module dependencies.
- *Implementation*: defines the internal details of the template. For simple active templates (such as those for activities) this defines the executable program, and for IDAs it defines the data attributes and access mechanisms.

Designs in MASCOT are expressed in a hierarchical manner rather than in terms of a flat network. At the lowest level, MASCOT entities are software objects capable of either performing data processing functions (activities or active entities) or data communication functions (IDAs or passive entities). A system defines a self-contained set of interconnected components. Some of these components can be grouped together to form a composite form of a processing function and are thus known as subsystem. Other components can be grouped together to form a composite form of a communication function and are thus called composite IDA. A system differs from a subsystem only in having, by definition, no external dependencies other than those, which may be satisfied during system building [IECCA and MUF 1987c]. The system is the outermost level of the network design, which encompasses the whole of the application. Explicitly or implicitly, it constitutes a complete description of the software.

In MASCOT (as indeed in any branch of engineering or the management of complex systems), it is good practice to apply the principle of “containment of complexity” during the elaboration of the design structure. The final hierarchical design structure should contain the minimum number of levels consistent with the ability to easily see how each component, at any level, plays its role to satisfy the requirements generated by the next level up. The final hierarchical design structure should be composed as a network of subsystems that communicate through different forms of IDAs: channels, pools or generalised IDAs. However, what is executed is a flat network of primitive elements (activities).

3.5.3 MASCOT communication model

As has been mentioned in the previous subsection, a MASCOT network design is represented as a set of concurrent operations such as subsystems, activities, IDAs or servers, which are the components that allow all the interactions of the system with the environment, e.g. the action of capturing images coming from the cameras in any surveillance system is performed by server components in this work. Notice that the server component in MASCOT differs significantly from a server object in OO, because in OO, the server object is commonly an active element that processes requests coming from clients. Therefore, it does not necessarily interact with the environment but with the elements that constitute the system. All these operations are interconnected to form a data flow network. The combination of different individual operations produces the overall system processing function and the data flow between these operations through the network takes place in accordance with the MASCOT communication model [Mustafa 2000], which is discussed in the following paragraphs.

The communication (between activities through IDAs as adjacent activities never occur), takes place along the paths of a MASCOT network. A path or connector between a pair of entities such as activities is a specification (Access-Interface) defining a set of operations (mainly reading and writing operations) implemented by the IDA. Every path in a MASCOT network is connected at one end to a port (*provided* action) of a component. Ports are belong to active entities such as activities and are represented by a solid circle. At the other end of each path a

window (*required* action) is connected, which is represented by a solid rectangle (see Figure 3-1). Windows exist only within passive entities such as communication components (IDAs). However, sometimes it is necessary for data to be passed directly from one IDA (passive entity) to another. Therefore, an RTN extension such as DORIS, allows IDAs to possess ports as well as windows. Ports and windows refer to the access interfaces that are connected, to obtain a full characterisation of how the components are connected to one another in a unique manner, which is decided by the designer.

Figure 3-1 shows a basic MASCOT communication model which is a representation of a network design of two activities connected by a “channel” IDA (different types of IDAs are discussed later). The graphical representation of an activity is a rounded shape whereas that of the IDA is a rectangular shape (see Figure 3-1). The names inside these shapes are template names and the ones outside are component names. As mentioned, a component is an instantiation of a template. In the example, activity *prod* has a port *P1* that is connected by the path *Put* to a window *W1* in the IDA *idacom*. In path *Put*, data flows from port *P1* (data source) to window *W1* (data sink). However, a port can act as a sink and a window can act as a source as shown in the path *Get*. This figure tries to illustrate that there is not dependency between the processing functions (defined in the activities which can be seen as a thread of executions) and data flow execution (realised in the IDA component).

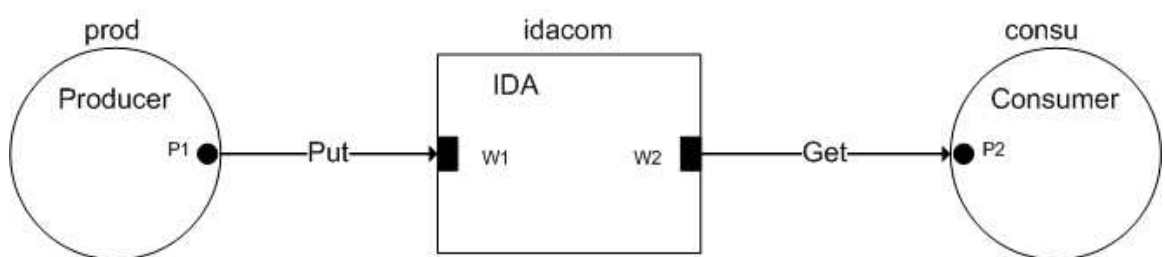


Figure 3-1. Simple communication model between two activities (producer, consumer) through an IDA component.

The interaction and communication within systems modelled on RTN principles are achieved by the reading and writing operations that are applied to the data in IDAs. When an active partner (such as activity) invokes these operations, data is

transmitted in one or both directions. The path or connector is not a resource that is dynamically created or allocated. If it were, then this would mean that not enough attention to the required nature of the communications would be given at design time and therefore full characterisation of the system architecture cannot be achieved. MASCOT defines a basic classification form of IDAs or interaction protocol based on how these operations act upon the shared data: destructive writing or reading operation and non-destructive writing or reading operation. Notice that the term protocol used here “is not a type of colloquy defining a message dialogue between two processes” [Simpson 2003f, pp.158] (protocols, in RTN, have no relation to the concept of a layered protocol hierarchy as in Open Systems Interconnection). Another important remark is that in RTN, the functional and design models emphasise the difference between protocol and connector. A protocol is a set of rules whereas a connector implements and enforces the rules. Consequently, operations of opening/closing access or connectors are not included in the protocol [Simpson 2003f]. This implies an easy association of the two models without compromising the distinction between functionality and design. Furthermore, a connector is at the same level as the entities (such as activities or IDAs) that use it (the connector is not seen as a property or “method” in OO argot inside the entity) and the components that use it remain attached to the connector following the construction of the RTN network at build time.

3.5.3.1 Communication mechanisms

In RTN the protocols define the dynamic effects arising from the interaction operations themselves. Destructive writing means that new data can freely overwrite existing data by destroying it (writer cannot be held up), while non-destructive writing means that new data can only be placed in a vacant space (writer can be held up). Whereas destructive reading will destroy current data and hence it will make a vacant space and non-destructive reading leaves the current data in place. These operations reflect four basic forms of interaction between communicating processes that are expressed in three basic types of protocols: Pool, Signal, Channel and in one special form of interaction called Constant. These protocols provide a sufficient set of characteristics for implementing a range of applications, by identifying all possible dynamic interactions between a reader and a writer. The basic parts of the protocol taxonomy are shown in Figure 3-2.

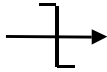
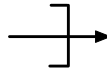
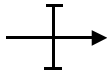

<u>Interaction of basic operations</u>	<u>Destructive reading</u>	<u>Non-destructive reading</u>
Temporal interaction effects	Reader can be held up	Reader cannot be held up
<u>Destructive writing</u> Writer cannot be held up	 Signal (Event data)	 Pool (Reference data)
<u>Non-destructive writing</u> Writer can be held up	 Channel (Message data)	 Constant (Configuration data)

Figure 3-2. Basic protocol taxonomy refers to effects on data from read and write operations.

Earlier versions of MASCOT supported just two different forms of IDA: pool and channel. Later versions of MASCOT incorporate another form of IDA called a signal. Figure 3-2 summarises the temporal interaction and which can be described as follows:

- **Pool** (non-destructive read and destructive write): It allows reference data (a single coherent record) to be consulted at any time by the reader or updated at any time by the writer like a table or a dictionary [Simpson 1990c]. Neither the reader nor writer process can be held up; therefore it is possible to lose the oldest data if the writer process is faster than the reader process. This protocol provides the opportunity to implement an explicit fully asynchronous communication between the entities that communicate with one another. Although this asynchronous communication is necessary in the real world, it has not been fully explored in the design methods discussed earlier. However, some of these methods claim to present mechanisms where the communication between elements is asynchronous, although this communication mechanism is not explicit in the form of a defined protocol

as the “pool” in RTN. Moreover, as discussed, this protocol may be used in a range of applications, e.g. in surveillance systems this protocol can be used to send background images data to the rest of activities that need this kind of data or raw images from sensor devices.

- ***Signal*** (destructive read and destructive write): It allows event or control data (a single coherent record) to be overwritten at any time by the writer, but only consulted once by the reader. Some data may not be consulted at all if the reader is too slow and the writer overwrites the event data before it has been read. The writer cannot be held up, but the reader can be held up. In surveillance systems this protocol can be used to send control data like variable operational parameters (i.e. thresholds, ROI or lighting changes) or event data like changing orientation of the cameras, where it is important for the reader process to act upon the last available (most current) data and not waste time with what has become obsolete (perhaps due to the slowness of the reader which could otherwise bring the system to a halt by delaying the producer of such data).
- ***Channel*** (destructive read and non-destructive write): It allows one message data item to be passed between producers and consumers. The read operation is destructive, since it removes an item from the channel in a FIFO manner. Either the reader or writer processes can be held up corresponding to a case where reader and writer need to synchronise on the presence of data. This protocol may be used in surveillance systems to send data that cannot be lost like the resulting tracking positions of the objects on the scene that are being followed.
- ***Constant*** (non-destructive read and non-destructive write): It is regarded as configuration data. The value of a constant is established at build time and may not be re-written. In surveillance systems there are some initial parameters like camera calibration parameters, thresholds for background detection or motion capture that need to be set up at the configuration step.

3.6 DORIS- further extension of RTN principles

Further successful developments of MASCOT continued until DORIS, which is the acronym of Data Oriented Requirements Implementation Scheme. DORIS is an

integrated set of methods and associated tools for development of hard real-time, embedded multiprocessor systems. It consists in a design notation based on Real Time Networks and various implementation techniques for system construction, implementation and analysis [Simpson 1992d]. Features of the DORIS design notation, as in MASCOT, include [Simpson 1990b]: support for a wide range of synchronous and asynchronous communication protocols (which are appropriate for both shared-memory and distributed implementations), features that support the partitioning of the design amongst large design teams, and support for the re-mapping of a design to the hardware platform, as it evolves during the life of the project. DORIS extends MASCOT-3 with an augmented set of pre-defined protocols, which are aimed for distributed real-time systems.

Figure 3-3 shows the DORIS coverage for the three stages of the development life cycle. They are **Definition**, **Design**, and **Implementation** (in software and hardware) [Simpson 1992d]. Different methods are used for each of these stages and each can be used in isolation, but the strength in DORIS comes from their integration. For the definition and design stages, DORIS uses extended versions of two existing well-established methods based on the concept of data flow. Controlled Requirements Expression (CORE) [Mullery 1979] is used for the capture, analysis and specification requirements and MASCOT is used for designing. For the implementation stage, DORIS it uses a new architectural approach known as DIA (Data Interaction Architecture) [Simpson 1990b]. DORIS data flow networks should be static but flexible. The network should be static because, as in MASCOT, dynamic creations should not be allowed, but flexible to allow for many changes in a design, which occur during development and subsequent maintenance. Dynamic process creation should not be allowed so as to guarantee the performance of the system, due to its consequent hazards in terms of unconstrained resource demands and non-deterministic timing.

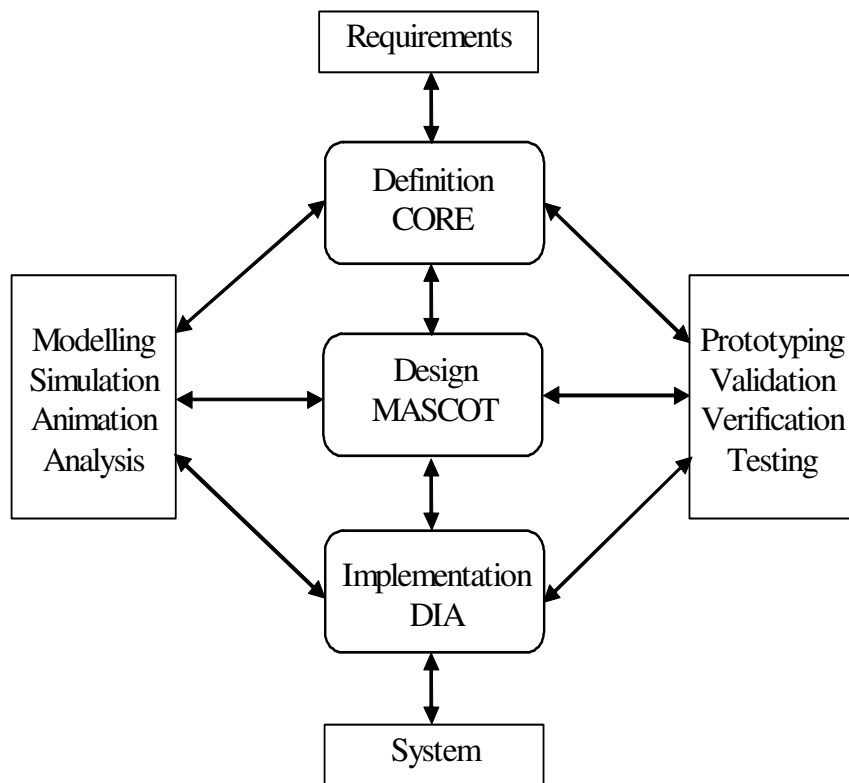


Figure 3-3. DORIS development process from [Simpson 1994e].

As discussed, Real-time Networks are characterised by the explicit recognition of shared data areas for communication and interaction between processes. DORIS uses the concept of Shared Data Area to provide a unifying theme through the three stages of **Definition**, **Design**, and **Implementation**. This ensures traceability during the development process, where the general concept of shared data area describes “shared information” at the Functional level, “shared data” at the Design level and “shared memory” at the Implementation level.

3.6.1 Definition: COntrolled Requirements Expression (CORE)

The definition stage of design is requirement analysis, where the examination of requirements produces a top-level system description. This provides the basis for formal or informal reasoning about the behaviour of the system and it is called the Functional Definition of the system. The Functional Definition is a representation of the system expressed in terms of component functions (transformers of information) and their interactions (information transfers), such as would be

generated by the application of CORE. The level of definitions should be sufficient to analyse the behaviour and performance of the system to the extent necessary to ensure that the given requirements can be met.

The aim of CORE is to establish a full understanding of the problem and associated requirements for a system solution, reducing ambiguities and inconsistencies. It consists of a set of defined steps helping to ensure a correct transition from the problem through to a system design definition. The Functional Definition in DORIS draws on the CORE method, but the CORE notation is not used. Instead design notational forms similar to MASCOT-3 are used, consisting of round corner shapes used for active processing functions and rectangular shapes used for information storage functions.

The definition phase begins with gathering information and analysing it, which leads to the definition of the problem domain and to the identification of a set of *viewpoints* from which the operating environment of the system is characterised. These *viewpoints* form the definition elements of the system design. The next stage is defining the function of each viewpoint and the information passed between viewpoints (data flow based analysis). Analysis of viewpoints is the basis for the formation of a Functional Definition of the system. The Functional Definition is a graphical description of a system design as a network of functional elements (viewpoints) linked by the information flows, and it is annotated with any identified route protocols. The role of the Functional Definition Diagram (FDD) is to act as a bridge that helps achieve a smooth flow from the definition phase into the design phase in DORIS. Its network format is consistent with the notation used for the DORIS design phase, and hence encourages propagation of requirements and ensures traceability of requirements and design information between these two phases.

3.6.2 Design: MASCOT

During Functional Definition, a network of interacting component functions is identified in detail for two reasons [Simpson 1992d]. First, the aim is to ensure that such a network of functions will indeed meet the system requirements. Second, to

allow some of the functions to be partitioned according to the most appropriate implementation technology, which are among others the Software System, the Processing Infrastructure, and the Hardware Instances. The design of the Software System is expressed in terms of the MASCOT Network application as a hierarchical set of MASCOT-3 components. The notations and conventions of MASCOT-3 are restricted and expanded. The design follows the same “principle of the containment” applied in MASCOT, with continuing emphasis on parallelism, communication and hierarchical breakdown.

The Functional Definition of the DORIS definition phase becomes the top-level system of the MASCOT design and its elements become MASCOT subsystems. Each successive level of decomposition is an implementation to some degree of the functionality and communication defined in the level above. Consistency and traceability are ensured by the continued satisfaction of Access Interfaces defined at higher levels of the design structure. Design visibility is enhanced by a graphical representation and the ability to display multiple levels of the design hierarchy on a single diagram. The MASCOT textual representation is the formal description of the system. Graphical and textual forms are equivalent and may be derived from each other. For the design phase of DORIS, the following three languages have been provided to aid the user in the design procedure of a system:

- ***DORIS Design Language*** (DDL): a subset of MASCOT-3 with additional syntax to allow the parameterisation of subsystems, activities, IDAs and access interfaces, and the definition of route IDAs (see section 3.6.4).
- ***Hardware Description Language*** (HDL): used to represent the hardware components and the interconnection between these components, which make up the hardware system. Typical components for the HDL are processors, private memory and shared memory in Asynchronous Dual Port Memory (ADPM).
- ***Mapping Description Language*** (MDL): used to map the abstract software design (software components) onto the system hardware. The activity instances and private IDA instances are mapped onto processor private memory and the shared-IDA instances are mapped onto ADPM. MDL contains also all priority rules and information for activities.

3.6.3 Implementation: DIA

RTN can be set in an operating environment which offers supporting services to concurrent processing, such as interrupts, pre-emptive scheduling, co-operative scheduling and multiple processors [IECCA and MUF 1987c]. The principles of RTN of shared memory, shared data, and shared information provide essential visibility of independent threads of execution, whose interaction between them is decoupled. In addition, the RTN approach encourages additional design partitioning by expressing a design solution in terms of a set of concurrent asynchronous processing threads that are suitable for flexible distribution in a multiprocessor hardware configurations. This addresses concurrency and parallelism at the highest and earliest level of definition and design, and it regards concurrency and parallelism as part of the solution rather than as part of the problem. Therefore, appropriate execution environments for supporting real-time networks based designs are those that reflect the network principles of independent processors communicating through shared memory. The Data Interaction Architecture (DIA) [Simpson 1990b] is based on the explicit recognition of shared memory as a means of communication between concurrent processes, thus its implementation form gives direct support for network design concepts.

3.6.4 Communication mechanisms

The concept of Route has been conceived to express communication designs and its symbols provide notational conventions to express basic and extended communication protocol designs. A route can be mapped into the hardware in a variety of forms to meet the communication requirement regardless of the relative location of the activities connected by the route. The dynamics of the route can be preserved over any degree of distribution, regardless of the communication medium (private memory, shared memory, serial link, multiplexed bus, etc.). Therefore, route interconnections between application functions can be established once the location of each end of the route is fixed and the dynamic properties of the route remain unchanged. Based on the DIA implementation of shared memory between adjacent processors, DORIS provides three forms of route distributions as follows:

- ***Private distribution***, when the two activities that use the route are both in the same processor. See Figure 3-5.
- ***Shared distribution***, when the two activities that use the route are in different processors, connected by shared memory.
- ***Remote distribution***, when the two activities that use the route are in different processors, not directly connected by shared memory.

Therefore, the explicit definition of route protocols gives the following two crucial advantages:

- It provides a complete set of communication protocols, which describes a variety of dynamic interactions between writer and reader.
- The ability to “stretch” the route over any distance in a distributed execution environment to allow communication to take place wherever the processes connected by the route may be located, including a Private distribution where connection is made between processes located in the same processor configuration. See Figure 3-5.

3.6.4.1 Route extensions

The extension of the first protocol classification [Simpson 2003f] is based on two main concepts: the number of intermediate items and void data. The former relaxes the capacity constraint established in the basic protocol where the intermediate item is limited to one. Then, it is possible in these extended protocols to vary the amount of intermediate data buffering, including no buffering at all, but “whatever the degree of buffering, items are always read in the order that they are written” [Simpson 2003f, pp. 161]. When there is no buffering, the interactions are interlocked with a mandatory overlap of read and write operations if the data is to be transferred. The void data concept allows the item to carry no information, so that the protocol describes a pure stimulus function (note the terminology borrowed from engineering applications). This provided new notations that are added to a route symbol to indicate special meaning. A small hollow circle at the centre of a route symbol indicates that no data is transferred. An integer next to a route symbol indicates the amount of buffering within the route. The absence of an integer implicitly means unity and a zero means no buffering. The principle of concurrency

is applied to route access operations [Simpson 2003f]. Such operations can be as concurrent as possible preserving always the capacity constraint of the protocol (please refer to Figure 3-2 to see these constraints).

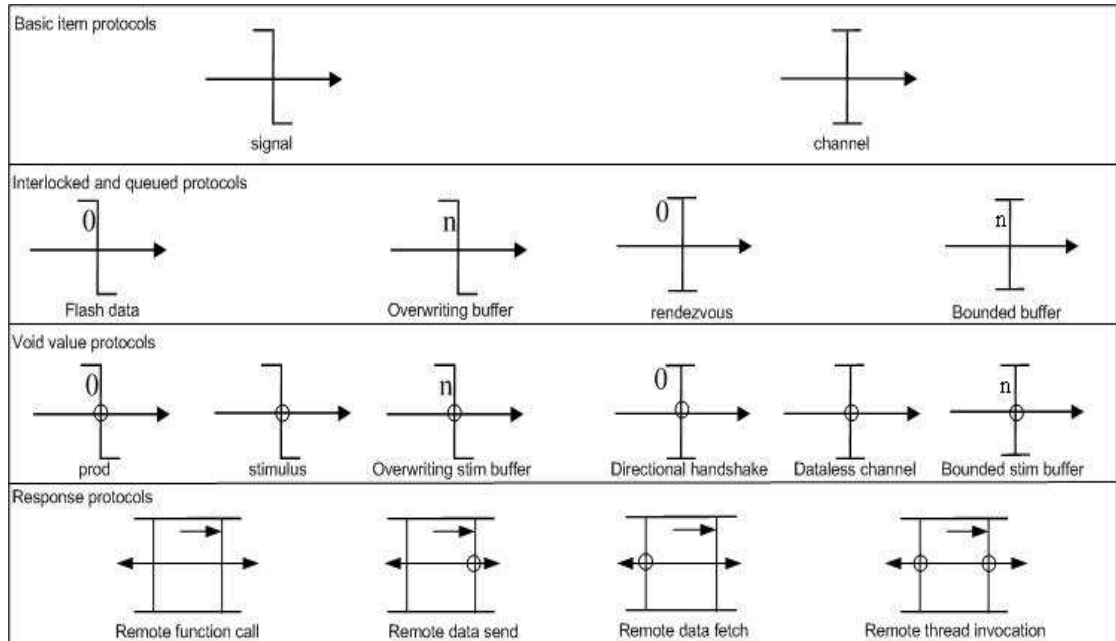


Figure 3-4. The Extended communications of *Route* protocols [Simpson 2003f].

Figure 3-4 summarises the extension communication protocols of the two basic protocols mentioned in section 3.5.3 and illustrated in Figure 3-2: the signal and channel protocols. The extension introduced additional routes, which allowed the explicit representation of these commonly occurring communication protocols and which are explained below. Note that in the previous basic protocols list (i.e. channel, signal, pool and constant protocols) and in the following list, RTN presents an extensive range of explicit communication mechanisms, defining all the possible interactions between communication elements. None of the reviewed methods present such a wide range of communication mechanisms.

- **Flash data:** Flash data is a signal route with zero buffer capacity. It is used to denote that the item will be passed only if the reader is waiting for it while the writer is inserting it, otherwise the item will be lost because there is no place to retain it.
- **Overwriting buffer:** An overwriting buffer is a signal route with a buffer capacity of more than zero. When the writer attempts to insert data in an

already full buffer, instead of waiting for a vacant space to be released by the reader, the oldest data is overwritten and the writer is able to continue. The integer “n” indicates the size of the buffer. The purpose of the overwriting buffer is to smooth the flow of message data at a variable rate without the (non-deterministic) hazard of a possible hold up to the writer.

- **Rendezvous:** A rendezvous is a bounded buffer with a capacity of zero. It uses the channel notation with an added integer “0”. The rendezvous is used to denote the meeting of two processes for the sole purpose of communicating information. Its dynamics of destructive reading and non-destructive writing are simultaneous. The temporal implication is that both processes must request to communicate before data can be transferred. This is the implicit communication mechanism that is commonly supported by most of the design methods.
- **Bounded buffer:** A bounded buffer is a channel route with a buffer capacity of more than zero. It uses the notation of the channel with an added integer “n”. The integer “n” is more than zero and indicates the size of the buffer. Data is not lost in a bounded buffer. The writing process is held up when the bounded buffer is full, and the reading process is held up when the bounded buffer is empty. The bounded buffer provides a smooth flow of message data when it is generated or processed at variable rates.
- **Prod:** Prod is equivalent to the flash data route, but with no data. Therefore is an event with no data. By using prod the reader is held up until the writer finishes “writing” the next void data. The prod uses the notation of a signal with a small hollow circle at the centre to indicate the absence of data.
- **Stimulus/Interrupt:** A stimulus or interrupt is equivalent to the signal route, but with no data. Like Prod, it is an event without data. The process raising a stimulus, or interrupt can never be held up. The stimulus/interrupt uses the notation of a signal with a small hollow circle at the centre to indicate the absence of data.
- **Overwriting stim buffer:** An overwriting stim buffer is equivalent to the overwriting buffer, but with no data. It uses the overwriting buffer notation with a small filled circle at the centre to indicate the absence of data. This route has the effect of storing remaining stimulus to a maximum “n”, and overwriting thenceforth.

- ***Directional handshake:*** A directional handshake is equivalent to the rendezvous, but with no data. It uses the rendezvous notation with a small hollow circle at the centre to indicate the absence of data. The handshake is used to denote a synchronisation point between two processes. No data is passed, but neither process can proceed until both have arrived.
- ***Dataless channel:*** A dataless channel is equivalent to channel route but with no data. It uses the channel annotation with small hollow circle at the centre to indicate the absence data. This protocol gives the effect equivalent to the raising of a single request or response with no value.
- ***Bounded stim buffer:*** A bounded stim buffer is equivalent to bounded buffer but with no data. It uses the bounded buffer notation with a small hollow circle at the centre to indicate the absence of data. This protocol has the effect of storing remaining stimulus up to an “n”, and thenceforward not allowing further insertion.

The following list presents four “response” protocols which are illustrated in the last row of Figure 3-4. These protocols are called “response” because they are modelled as a result of different pair-combination of data and dataless channels, and correspond to closed bidirectional protocols which model the client-server relationship. The bi-directional nature of these protocols, represents an interaction, where each process writes on the protocol symbol nearest to it and reads on the protocol symbol furthest away. The small arrow indicates the direction from client to server. These are asymmetric and the client uses a single operation to send and receive the results while the server uses two different operations to receive the parameters and to send the results [Simpson 2003f].

- ***Remote function call:*** The dynamics effects of a remote function call are achieved by a bi-directional channel through which parameters are passed in one direction with the results being returned in the other direction, allowing them to pass two different types of message data (parameters and results). The effects of this protocol are equivalent to a client process transmitting parameters through one of the channels and waiting for the server to take the parameters to carry out an action and return the result through the other channel. Data cannot be lost in this form of protocol [Simpson 2003f]. The

notational symbol of the remote function call contains a data flow line with an arrowhead at each end.

- ***Remote data send:*** This protocol is the combination of a channel and a dataless channel. No results are expected and the effects that this protocol raises are equivalent to an explicit acknowledgement that the data item has arrived. The client is held until the acknowledgement is received. The notational symbol of the remote data send contains a data flow line with an arrowhead at each end, together with small filled circle at one end to indicate the absence of data.
- ***Remote data fetch:*** This protocol is the combination of a dataless channel and a channel. The effects that the protocol raises are equivalent to request data from another process. In this protocol the client is held until the data is received. The notational symbol of this protocol contains a data flow line with an arrowhead at each end, together with small filled circle at one end to indicate the absence of data.
- ***Remote thread invocation:*** This protocol is the combination of two dataless channel. No parameters, no results are expected. The effects that the protocol raises are equivalent to invoking a thread of execution. The notational symbol of this protocol contains a data flow line with an arrowhead at each end, together with small filled circle at one end to indicate the absence of data on both sides.

3.6.4.2 Communication model

The protocols presented in Figure 3-2 and in Figure 3-4 represent the different types of communications between two processes. These figures illustrate that each element of the protocol taxonomy presents a different temporal interaction within the communication of the active parts (processes), giving coverage of the dynamic constraints (i.e. destructive and non-destructive data capacity) that may occur between the processes in the network. These protocols are used in a distributed environment where processes are allocated in the same physical node sharing memory. These protocols have a stretched form (routes) to allow communication between nodes that are physically allocated in different places and therefore do not have a common visibility of shared memory. These remote routes project the data shared from one place to the other place by introducing an active element (activity)

between them. Figure 3-5 shows how the data that is placed in a visible shared memory (b) is projected by the active element (a) which invokes the same operation applied to (b). In DORIS the concept of a link element is introduced in order to stretch the “response” protocols illustrated in Figure 3-2 and Figure 3-4. The link element basically projects the response protocol used to the other side, i.e. the interface becomes a remote one. The symbol annotation of the link element is presented in Figure 3-6.

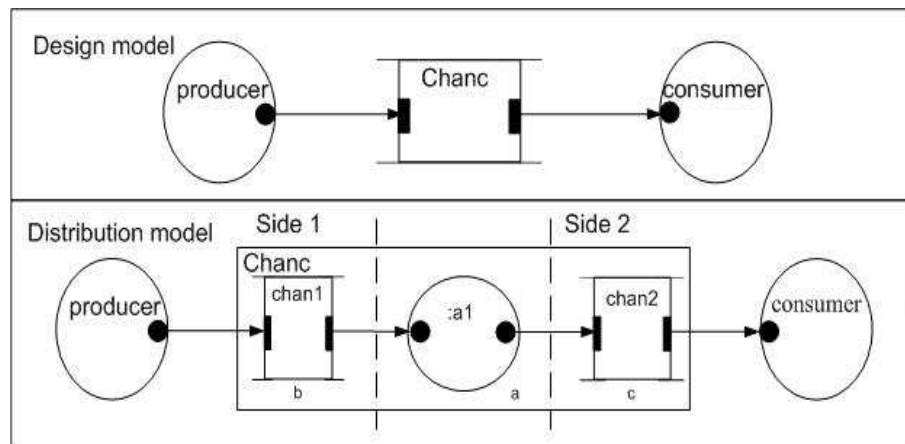


Figure 3-5. Example of the stretched form of the channel protocol [Simpson 1994e], [Simpson 2003f].

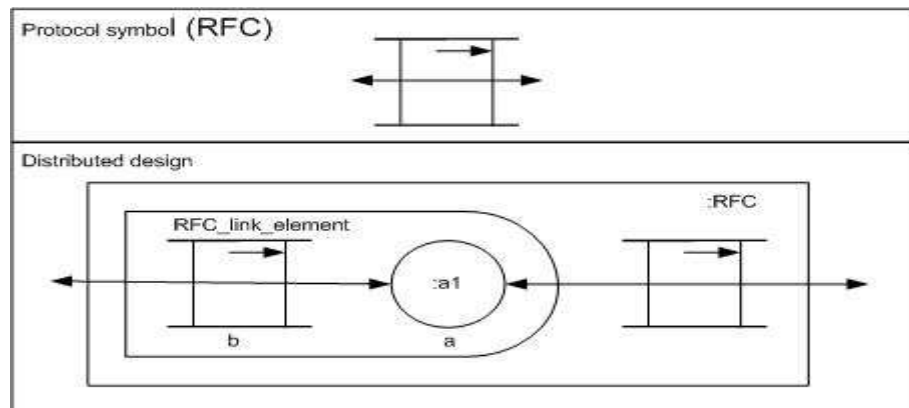


Figure 3-6. Distributed model of remote function call [Simpson 2003f].

3.7 Comparison between the OO and the MASCOT/DORIS approaches

OO and MASCOT/DORIS approaches can be suitable for a range of distributed applications, and they can provide a degree of reusability and extensibility because of the modularity of their designs. Both approaches have different key aspects, which are described in the subsequent subsections, see Table 3-2. These differences

are based on the essential concepts and the structural design model, and consequently, based on how each approach develops its concepts in order to design distributed concurrent real-time systems.

		OO	Real Time Networks
Essential concepts	Abstract model	Classes/Objects	Templates/components
	Communication model	message passing	Shared data (protocols taxonomy)
	Concurrency	Not inherent in the design.	Inherent in the design
	Information Hiding	Encapsulation	Access procedures, access mechanisms
	Modularisation	Objects, classes	Activities, IDAs, subsystems, servers
	Inheritance	Yes	No
	Dynamism	Through late bindings, inheritance and polymorphism.	MASCOT 2-Yes (although it is not advisable) MASCOT 3/DORIS- No
	Timing behaviour	Non-explicit	Partially explicit through the temporal interaction effects on the operations (reading and writing)

Table 3-2. Summary of the aspects to compare at the conceptual model.

3.7.1 The difference of abstract model between the two approaches

Conceptually one of the main differences between OO and RTN is how both approaches tackle the problem of modelling real-world entities. The OO approach abstracts the problem by modelling the real-world entities as objects. An OO design tries to reincarnate objects from the problem domain into the computer models, giving the objects in an OO program (OOP) equivalent characteristics and capabilities as the real-world entities that they are modelling. These objects are commonly grouped to simplify design and reduce code, by defining a relationship between these objects such as inheritance. For example a car object, a lorry object and a motorcycle object can be inherited from an abstract class called vehicle, because these three objects have some common attributes that can be grouped in an

abstract class called vehicle. Therefore, by creating this inheritance relationship between these objects dependency behaviour between these objects is created, while real-world entities may not express this dependency relation between them. Besides in [Boasson 2002] it is pointed out that the real-world entities usually have their own autonomous behaviour.

Moreover, OO methodologies focus on system components rather than the actions that the system has to perform. Therefore, OO designers make decisions on subdividing rather early, whilst RTN designers focus on the tasks that the system should perform and the interaction between these tasks. On the other hand, the shared data model in RTN represents the concept of the entities in the real world as independent active activities which communicate through independent passive components. These activities, that have their own autonomous behaviour, can be grouped forming a subsystem which along with other subsystem constitutes the system. This is a hierarchical design which describes decomposition within functional components rather than a hierarchical relationship between components as it is described in OO design. Moreover, the boundaries of the system designed are exposed more explicitly in RTN than in OO designs.

Modularity is reached in each approach in different ways. In OO, modularity is achieved through the concept of object, which encapsulates certain attributes and operations or methods in an entity (called class in OO nomenclature). In RTN, the system is partitioned into smaller independently operating subsystems, which only interact through explicitly defined intercommunication areas.

3.7.2 Communications

The term encapsulation, used to describe information hiding in an object, plays an important role in the OO communication pattern. Encapsulation can be considered as the process of hiding all the details of an object that need not to be visible to the other objects. In OO, an object is characterised by a condensed list of abstract attributes and a list of encapsulated procedures, which are defined as methods, operations and services. Data from an object, in OO systems, is obtained by sending a message to the object. A message consists of the address (reference) of the object

which it will send to and an instruction which consists of a method name and the required parameters.

By using information hiding, the implementation detail of a method is private to the object and hidden from the rest of the objects, only its behaviour is visible to other objects. Thus, objects have internal state but it is not directly accessible. Consequently, clients of the object are not exposed to danger when its implementation is changed as long as the interface is not also changed. Note that in OO the hiding information is necessary only if the designer wants to incorporate it in the design. It is therefore possible for the designer to make everything visible, despite this being accepted bad practice. Methods are defined as a procedure or function that alters the state of an object or causes the object to send a message, i.e. return values. Moreover, the syntax of methods defines which messages an object is able to process successfully. A message, which is often implemented as a function call, may be interpreted in different ways by different receivers which decide what will happen. The set of messages that the object can respond to is sometimes called its protocol. For each message there is an operation. The name of the message is the name of the operation and the parameters on the message are the parameters of the operation.

The communication pattern in OO is based on a client-server model where in some Object-Oriented Programming (OOP) such as C++ or Java, the client and server objects communicate by message passing. In a well designed OO, its items should be strongly coupled. The object as a whole should possess high cohesion or high modularity. It is stated by the OO community, that a message passing communication model creates weak coupling between objects and uses information hiding to ensure the access validity (interfaces) to data structures that are encapsulated in an object. Nevertheless, if there is still coupling between parts that communicate using message passing, then synchronism between communicating parts (objects) is required and to decouple the objects that communicate with one another, external mechanisms (or services) such as “time out” need to be added to the communication model. Moreover, the interaction effects between communicating objects through the use of the message passing model are not explicitly captured like it is done in the RTN taxonomy protocols. In other words,

the important nature of the required communications between different parts of the system is difficult to capture with OO approaches (where this aspect tends to be implicit rather than explicit) whereas it lies at the core of the RTN approach.

3.7.2.1 Differences between MASCOT3/DORIS and OO communication model

The communication process in both approaches promotes modularity and re-usable software by promoting weak coupling between modules or software entities as mentioned before. Nevertheless, it is addressed in a different manner. In MASCOT3/DORIS the communication model is based on shared data between active processing data entity (activities) through the passive communication entity (IDAs). Therefore, the basic functionality of IDAs is to allow communication, then in RTN the communication tends to place emphasis on a visible shared component between entities, while in OO approaches, it may be said that the communication tend to look "inwards" (the "state" of an object). As illustrated in Figure 3-2 and Figure 3-4, MASCOT/DORIS provides a rich set of explicit data communications primitives (protocols taxonomy) that really reflects what one is likely to encounter in real-time systems typified by distributed visual surveillance systems, e.g. depending on the data type used in the system or on the dynamic interactions required between the processes to communicate, a different functional behaviour is needed, which is possible to obtain from this set of communications primitives. The temporal behaviour inherent in the protocols is regarding the effects arising from resource scheduling as an implementation concern [Simpson 2003f].

Therefore, the taxonomy protocols presented in MASCOT3/DORIS reflects the functional behaviour and temporal properties of the communication between the processes in the system design. For example, in Figure 3-1 the communication between two activities (producer and consumer) is a rendezvous communication (rendezvous protocol, see Figure 3-4). The producer can only send the data if the consumer is waiting for it. Instead of using a rendezvous protocol to connect producer and consumer it could be designed to use a signal protocol (see Figure 3-2). Therefore, the producer can send the data whenever is ready without waiting for the consumer to read it. It is possible to see that the essential interaction between the producer and consumer activities has changed since the communication protocol

has changed but without changing the implementation of the access interface and activities themselves. Moreover, in RTN the access interfaces are completely independent from their attached components. In RTN two components (e.g. two activities) can communicate with each other without opening/closing access, with complete independence of the “method” itself. Moreover, the type of interactions e.g. synchronism or asynchronism is determined by the communication protocol used.

RTN shifts the emphasis from the "state" view (internal and publicly available) as it is done usually in OO approaches to the “communications” view. Thus, in RTN the internal may be considered a simple sequential activity or data processing function (the smaller the better, generally speaking to "contain complexity", that does not require all the sophistication and complication of OO). Therefore, the behaviour of a system depends on the temporal performance of these processes (data processing functions) plus the communications between processes.

In OO it is possible to represent a communication entity by creating an object with a given set of methods allowing the communication between two objects to be separated but without being able to represent explicitly the temporal properties of this communication. Therefore, it is possible to speculate that in OO the communication scheme core is not capable of reproducing the temporal essence of the real world requirements. On the other hand, if the implementation of this communication object is changed as far as the interfaces (methods) are not changed, it is not necessary to change the implementation of the two objects which are communicating through this object. This is achieved by encapsulating and hiding the methods which are part of the implementation information of the communication object. Nevertheless, in OO access interfaces are separated but not independent because they lie in the object that implements them, see Figure 3-7.

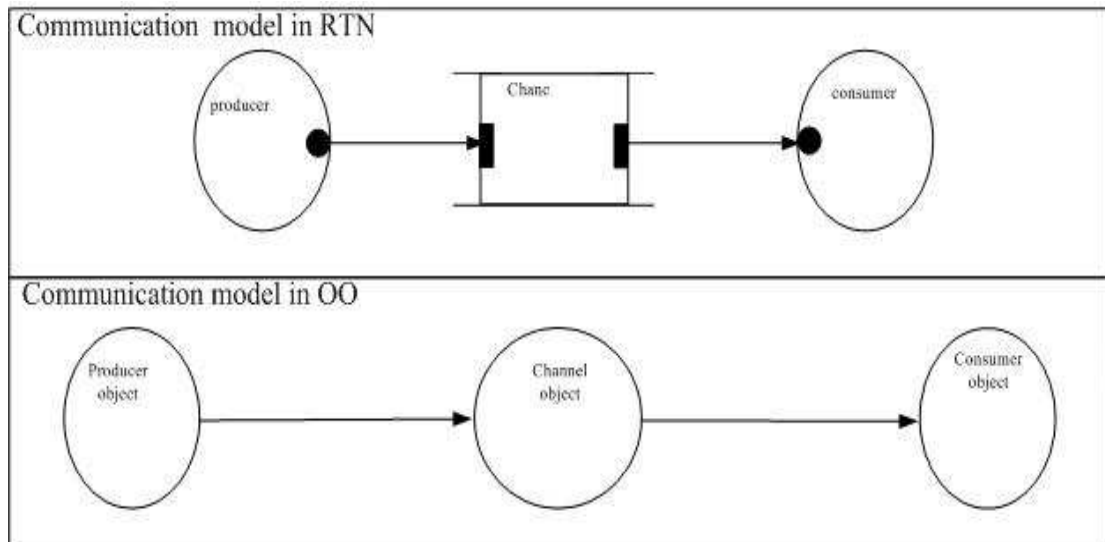


Figure 3-7. Example of different approaches to the communication model; in OO objects communicate to objects. In RTN communication is from/to activity to/from IDA.

The example shown in Figure 3-7 illustrates another difference between the two communication models. In RTN the communication components (e.g. channel) and the two processing activities are thread independent, while in the OO communication model the active objects and the communication object are not by nature thread independent, i.e. in order to design these objects to be thread independent each of the objects must be specifically designed as separate threads in a multithread programming environment. Moreover, even though in OO it is possible to represent a communication entity by creating an object having the functionality to communicate between two objects, there is no established distinct communication component, as is the case in MASCOT3/DORIS with IDAs, let alone a defined taxonomy of protocols. As discussed, the taxonomy of protocols in MASCOT3/DORIS defines a different functional behaviour in the communication depending on the type of interaction required. The design decision of the type of interaction should be determined by the subsystem in which is embedded. For example, the pool protocol defined in MASCOT3/DORIS allows a completely asynchronous communication between two active components, which means that the writer and reader work concurrently and they are never held up. The data to transfer in this protocol is a reference data type (“dictionary” data). This data may be lost because the writer can be faster than the reader. Therefore, this protocol can be used as an explicit design decision that it is better to lose data than degrading (possibly in a non-deterministic manner) the performance of the system.

These design discussions on the type of interactions between active processes that are done in MASCOT3/DORIS are not possible in OO. Furthermore, in MASCOT3/DORIS by separating the processing components from the communication components it is possible to describe in a natural way and explicitly the dynamic interactions the processes should have to communicate and also it allows to explicitly identify the data processing function primitives (processes), which is important from a system functional view point. This information, which is extracted at the design level, is important at the implementation stage but also has a major significance at run-time when scheduling policies have to be applied in order to build the real-time system.

In MASCOT3/DORIS the activities as active processes are assigned with different priorities and the communication IDAs (the protocols) usually are mapped as a shared resource between the active processes if they are allocated at the same physical node. In other words, RTN provides a strong form of design partitioning which gives a sound basis for working allocation and allows good visibility of progress during development. Activities, subsystems, IDAs can be embodied in special test systems for prototyping or integration testing purposes, if necessary in execution environments which differ from the final target configuration. Further development aspects of MASCOT3/DORIS are discussed in the next chapter. On the other hand, in OO the visibility of the communication between objects components and the explicit definition of their dynamics interactions and the data exchanged is not regarded as a crucial design decision and it is usually left to an implementation stage. Therefore, specific temporal properties and also functional behaviours, which real-world requirements have, are not reflected in the design. From a systems engineering view point, it is important to force the specification of these properties and the reflectivity of these behaviours at the design stage for example to avoid un-deterministic hazards.

3.7.3 Concurrency and Information hiding

Defining the temporal and functional behaviour of the different parts that form the whole system, is essential in order to design and build the desired system. Although most of the temporal and functional properties are defined at the stage of

specification of the system like requirements and constraints, to infer and verify these properties at the design stage is not a trivial task. In MASCOT, functional properties are made clearly visible in the design process due to the close correspondence between the functional specification of the components and the designed components at the design stage. The temporal behaviour of the activities in RTNs is mostly established by their own processing time and the nature of the interactions between them through the intercommunication data area (IDA). So, as it is explained in [Simpson 2003f, pp.157], “the overall timing properties of a system are therefore determined by a complex combination of the timing of processing operations within individual processes, taken together with the timing effects of process interaction”.

In the OO approach, although the functional behaviour can be illustrated at an initial stage of design by means of classes and objects and their relations between other classes or objects, derived designs can include more objects or relations whose functionality is not shown or is not clear. The temporal properties in the OO approach, as is mentioned in the previous subsection, are generally quite difficult to define (in fact, they are effectively ignored because time properties cannot be fully determined until the temporal interactions nature between processes is defined). Consequently OO usually does not support the verification of timing deadlines effectively, which is an important requirement in real-time systems.

The approach to concurrency, which in real time systems and especially in real time embedded systems is usually an essential property, differs in the two methods. In the OO approach concurrency is applied by implementing objects using for example. multi-threaded OO programming (effectively using a class library that includes thread classes). RTN assumes that the concurrency comes from the problem, from the solution in hardware (i.e. multi-processing) and from the design approach. The network of activities communicating through defined communication components IDAs are presupposed to be independent and concurrent. In MASCOT/DORIS it is assumed that if concurrency can be exploited, then concurrency is part of your design.

Information hiding was introduced by Parnas [Parnas 1972] as a procedure for decomposing a system into modules. According to Parnas, information hiding is a design decision, which consists of the following idea “any design decision that is susceptible to change should be hidden”. Therefore, each key design decision should be known to only one module and then information shared between modules is kept to minimum [Gomaa 1984a]. OO and RTN both hide information although they use different approaches. In OO the object can be considered as an information hiding module if the designer does not decide to make the module visible, which is considered bad programming practice. Therefore, attributes, operations and methods, (especially the attributes) are often designed in a way that only the object which defines them, can access them. In MASCOT/DORIS information hiding is provided by means of “access procedures”. Therefore, the details of the data structure and the synchronisation of the access to this data are hidden from the active processes.

3.7.4 Inheritance

As mentioned, OO has an essential component called an object which is defined by its state, identity and behaviour. The state of an object comprises all the static properties of the object and the current dynamic values of each of these properties. The identity is the property of an object which distinguishes it from all other objects. The behaviour is how an object acts and reacts, in terms of its state changes and message passing, i.e. the operations that its clients may perform upon it, also the operations that it may perform upon other objects. The relationship between objects can determine which operations can be performed and what behaviour results from the relation. There are two main kinds of relationships between objects:

- ***Using relationships*** the object involved in this type of relationship may only operate upon other objects, it may only operate by other objects or it may operate by and upon other objects.
- ***Containing relationships (inheritance)*** the object has a “is-a” relationship with other objects. This containing relationship can be called inheritance.

Another main relationship in OO languages is called instantiation, which defines the relation e.g. between objects and their classes. As discussed, there are also other relationships among classes based on a “kind of” or “part of” class relationship, i.e. respectively generalisation and aggregation. Lastly, association is the relationship which denotes some semantic connection among unrelated classes. Currently, several approaches have evolved in programming languages to express these kinds of relationships among classes.

Inheritance is a relationship that may affect the determinism and performance of the system. As mentioned in previous section 3.7.1, inheritance at the implementation level, is a mechanism for sharing and reusing code between classes. The notion of an inheritance or classification hierarchy is that it deals with the structural and semantic relationships between objects and between classes (i.e. subclass inherits from one or more super-classes) and eliminates the redundancy of storing the same data or procedure more often than necessary. A subclass typically redefines the existing structure and behaviour of its super-classes. Normally, in OOP, objects inherit methods and attributes from their superior classes, but they do not inherit the values of attributes, merely the ability to have a certain type of value. Therefore, at the design level, inheritance allows concise definitions of subclasses which are described only in terms of how they are different from their super-classes.

It is possible to have classes with a single or multiple inheritance relationships. The difficulty with multiple inheritance is that sometimes the properties inherited from two (or more) parents may be directly or partially contradictory, which may create conflicts. Therefore, there exists a compromise between complexity and reusability; the more complex a system is, the more difficult it is to maintain, and the more semantically rich it is, the more specific and therefore less reusable its components will be.

Inheritance is supported in OO systems but it is not supported in RTN approach. The problem that arises is that the increase of coupling between modules due to inheritance creates an additional type of coupling between a class and its super-class. It is also necessary to be extremely cautious about this reusability; by exposing implementation details to an object’s clients it may be difficult to reuse

the code after applying some changes. Moreover, the hierarchy itself may be exposed, so changes cannot be safely made and it is not possible to guarantee that the interface of an object has not been changed.

However, at this point it is possible to ask if RTN does not include inheritance, how does RTN provide reusability and extensibility, features that OO systems claim to have through inheritance? MASCOT uses the concept of templates and instances. For example, in Figure 3-1 “producer” is the template of the activity on the left of the Figure 3-1 and “prod” is the name of the instance of the template called producer. In RTN, instances are executed at run-time. It is possible in RTN to have more than one instance of the same template in the same design, providing then, the reusability of modules. However, in RTN, the relationship between templates and instances is unique. If the template is changed the instance is also changed.

In the OO approach, extensibility is not only an extension of the system by upgrading with new modules or new instances but refers as well to the property of extending a module or creating a different module from a primitive one, using the inheritance property. In RTN extensibility it is seen only as an upgrade or extension of the network design by adding new modules.

3.7.5 Polymorphism and dynamic binding

Another mechanism in the OO approach, that is linked to the concept of inheritance and is used to share and reuse code, is called polymorphism. Polymorphism, (literally “having many forms”), means the ability of a variable or method to have different behaviours at run-time, or more specifically the ability to refer to instances of various classes. For example, the same named method can behave differently depending on the parameters that it receives or can behave in the same way even though it has received different type of parameters. This form is normally called “overloading”. A form of polymorphism also may be used when the features of inheritance and dynamic binding interact. Dynamic or late binding means that the types of all variables and expressions are not known until run-time. In this case, polymorphic methods can be thought of as late-bound procedure calls, where the actual method or procedure to be invoked is not determined until the method is

actually applied to a specific object. As pointed out in [Graham 1994] polymorphism considerably enhances the information hiding feature of OOP. So the linking to the method can only be done at very last moment. Further, it promotes encapsulation by allowing general-purpose classes to be written that will work successfully with different types of objects.

In RTN approach, one of the forms of polymorphism mentioned can be found when some predefined access interfaces (e.g. put or get) are used in different subsystems, and therefore these access interfaces deal with different type of data but behave in the same way: put write the data no matter which type it is. The other form of polymorphism related to inheritance and dynamic binding is not explored in RTN, because it increases the non-determinism of the behaviour of the components. The behaviour of the method is not known till the last moment at run-time, because the class of the object being operated upon may not be known until run-time. Therefore, it is not possible to predict its behaviour in some critical situations and the scheduling policies are not easy to apply due to this uncertainty. In safety-critical applications the ability to predict system behaviour (or at least bound it) is obviously crucial.

3.7.6 Performance

The advantages most often put forward in favour of the OO systems are the inherent reusability of the objects and the extensibility of OO systems. It is asserted that [Graham 1994], [Booch 1991] the features of inheritance, polymorphism and dynamic binding can contribute to simplify and to reduce development time and the size of the resulting source code, which are important features in real-time embedded systems. Nevertheless, other features in OO like dynamic linking and garbage collection imply extra run-time support, introducing run-time performance overhead on the speed of OO programs. Garbage collection is a mechanism that allows the freeing of heap space for dynamically-created objects that are no longer needed so that the space in the heap is made available for subsequent new objects. The garbage collector somehow determines which objects are not referenced by the program anymore and releases the heap from such objects. Moreover, the design of a system using these mechanisms implies difficulty in testing due to the lack of

determinism for example in the schedule predictability or determinism in the behaviours of the components of the system. Moreover, when there is more than one thread of control it is more difficult to control predicted behaviours and sometimes unpredicted behaviours are arisen which may trigger deadlock situations.

Dynamic invocations in OO may imply time performance cost on the communication between objects. For example, an implementation of an invocation method, that cannot be resolved statically, must do a dynamic lookup in order to find the method, which has been defined, for the class of the receiving object. [Booch 1991] indicates that dynamic invocations clearly take much more time than simple subprogram calls that are made when the invocation of a method is done statically. Moreover, the OO approach allows the design of system components using more than one layer of abstraction. Hence, invoking a method at the high-level of abstraction may result in a cascade of invoking methods (high-level methods usually invoke lower-level methods, and so on), reducing the overall system performance. Therefore, for applications in which time is a limited resource such as in real-time systems, so many invocations may be unacceptable.

Another performance risk in OO is derived from deep class hierarchies. Many inheritance relationships provoke many super-classes, whose code must be included when they are linked into the most specific class. Thus, an excessive amount of object code is produced. The last remaining performance risk with OO systems comes from the dynamic allocation and deallocation of objects. Allocating an object on a heap is a dynamic action as opposed to statically allocating an object either globally or on a stack frame and heap allocation usually costs more computing resources. Again, for time-critical applications, the cycles needed to complete a heap allocation are not affordable.

As mentioned in previous sections, the features of inheritance, polymorphism and dynamic linking do not exist in MASCOT-3/DORIS and for reasons of good engineering practice. The main reason for this is because in RTN all the components, which constitute the network system, must be defined before starting up the system avoiding dynamic creations (as mentioned, in RTN the design network should be static) . In other words, at the design level the network is

completely described and defined. Then, at the physical mapping stage the network is created by physically allocating the activities and routes. Therefore, at run-time the network is as fully determined as possible, not allowing the possibility of late-dynamic creation of activities or routes. However, as mentioned in section 3.5.2, for a system with “imperative” dynamic requirements RTN proponents provide solutions to establish this dynamism in the RTN network.

Defining and describing, at the design level, all the communications and relationships between all the components which constitute the whole system, can be seen as a time consuming task. However, this is consistent with good engineering practice and it provides understanding of the whole system, giving the designer control over the system, as well as making the system more deterministic and predictable. Deterministic in terms of knowing before run-time how many components will exist, avoiding the possibility of resource exhaustion, because the resources are fixed before the start up process of the system. Predictability, in terms of scheduling the activities e.g. by knowing how many processes will exist it is possible to predict processing times of tasks and apply one of the known tactics for scheduling these tasks. Predictability, in terms of deadlines of the tasks, can be partially determined by the use of MASCOT/DORIS protocols. These protocols imply different temporal interaction effects on the activities that communicate, and therefore it is possible to predict e.g. when the activity will start its own process, by examining the type of protocol used.

3.8 Summary

One of the conclusions in chapter 2 was that to design a large distributed real-time surveillance system, it is necessary to establish a framework from a point of view of solid system engineering principles, which allows the creation of a system, instead of building such system as integration of different algorithms placed in different computers. Therefore, in this chapter an introduction to software design methods to create such systems has been presented. Furthermore, a comparison between the OO approach and the proposed approach RTN in this work has been presented. Even though OO design methods are wide-spread and are the technology commonly used to design systems, in this work we present RTN as a design approach for

surveillance systems because it is a mature technology inspired by hard engineering applications and that partitions the software giving complete visibility of the different components that constitute the system. Besides the similarities and differences between OO and RTN approaches, which have been highlighted in section 3.7, there are two main differences that we present in this work, RTN as a design approach for surveillance systems. The first difference lies in the fact that the OO's philosophy is to consider software as multiple-purpose flexible artefacts. However, RTN's philosophy is to consider software as an engineering fit-for purpose product (an engineering system does what it is supposed to do and nothing more). A clear example that illustrates this statement is taken from a private conversation [S. A. Velastin 2006] “[...]...A civil engineer would not construct a bridge thinking that eventually it could also be used as a ferry! At the same time, a user of the bridge crosses it with confidence that it has been built using solid engineering principles and that it is not a multiple-purpose appliance...[.]”.

The second main difference, which has been stressed through this chapter, lies in the communication model of OO and RTN. RTN explicitly expresses, through a rich set of protocols (taxonomy), the functional behaviour and the timing properties of the communication between elements in the system, because the communication is considered a crucial part in the specification of the system. In OO the communication between elements is considered an additional part of what it is important (the elements that communicate with one to another). Therefore, in OO, there does not exist any taxonomy or explicitly characterisation of the functional behaviour and the timing properties of the communication between elements in the system.

Moreover, RTN imposes a disciplined approach to design, which yields a highly modular structure, ensuring close correspondence between functional elements in design and constructional elements for system integration. DORIS also allows different interactions between the components through its protocols extensions, providing the possibility of creating an asynchronous communication between different processes. The following chapter will now present a comparison between two different specific design solutions using CORBA (OO) and DORIS (RTN) for an existing real time surveillance system.

4 Case study: ADVISOR

4.1 *Introduction*

This chapter describes a comparative study between two key technologies: CORBA and DORIS. Each of these two technologies embodies the principal concepts of OO and RTN, respectively. CORBA and DORIS technologies are used for designing and implementing solutions mainly in distributed environments. Moreover, CORBA, at the time of writing, has been presented in OMG (Object Management Group) documents as the multi-platform and multi-language solution for distribution and system integration. The OMG has also stated that CORBA will continue to expand as the particular platform for real-time, embedded, large, mission-critical enterprise computing systems using OO technology. Therefore, this chapter is centred on a comparison of the distributional properties and the architectural design that the third generation of surveillance systems require. This comparative study is done through a case-study of an existing research solution for a real-time distributed surveillance system called Annotated Digital Video for Intelligent Surveillance and Optimised Retrieval (ADVISOR). This choice has been made because a prototype of this system (called in this chapter ADVISOR Prototype) used CORBA as a system integration and distribution solution and it represented a major effort in investigating distributed surveillance systems.

Therefore, this chapter firstly presents a brief introduction to CORBA in section 4.2 (note that this section does not intend to give a comprehensive description of CORBA, but just to give a brief introduction to it by highlighting some of the features and components that make up CORBA and that are used in the ADVISOR Prototype). Section 4.3 presents a generic solution of a distributed surveillance described in terms of its aims, requirements and specifications, which is called ADVISOR system. Section 4.3 also highlights the differences between the ADVISOR system and a particular implementation of ADVISOR system that is called ADVISOR Prototype. This Prototype is the specific system used in this case-study.

Following a brief introduction to CORBA and the description of the ADVISOR system and the prototype, section 4.4 discusses and illustrates graphically the CORBA architecture design used in the ADVISOR Prototype. It is important to mention that in this chapter, in order to illustrate the different designed architectures, the DORIS graphical notation is used. The reason for this is the need to illustrate, in a graphical manner, the CORBA and DORIS solutions for subsequent comparisons, but CORBA does not have any specific graphical notation. Although it may be argued that it is possible to specify a CORBA solution using UML notation, it is easier to depict the differences in both technologies using the same graphical tool (i.e. DORIS graphical notation). The next section 4.5 presents the architectural design of the ADVISOR Prototype using CORBA. Therefore, it is important to stress that the fact that the DORIS graphical notation has been used to represent a CORBA architecture solution does not necessarily mean that such a solution uses underlying RTN concepts.

Section 4.6 presents a new architecture design solution of ADVISOR using RTN concepts (i.e. from the same requirements presented in section 4.3, a new solution is presented using only the fundamental RTN concepts). Having then presented and discussed the two architecture designs that use CORBA and DORIS approaches, section 4.7 compares the two approaches by first highlighting their differences and then by focusing on three aspects: communication, distribution and development process.

For example, we highlight that the main difference in communication between distributed processes is that while CORBA is based on a client-server relationship, MASCOT/DORIS uses a passive element. We then show that this and other differences in communication have a direct and significant effect on the architecture designs. Distribution in CORBA is based on the design and posterior distribution of CORBA objects over a distributed processing environment. These objects represent the servers that, through static/dynamic invocations, handle the requests from the client. By contrast, in MASCOT/DORIS the distribution is centred on a template substitution, which allows distribution of the elements that constitute the application network while maintaining the defined communication protocols. To end this comparison, there is a discussion of integration policy and development aspects,

such as how CORBA manages all the interactions between objects created statically or dynamically at run-time. A brief summary of these differences is depicted in Table 4-1.

This case study then leads to, as presented in Chapter 5, a general proposal for a large scale real-time distributed intelligent surveillance system architecture, using DORIS as the chosen design method. This proposed design addresses some aspects of this complex domain, highlighting one of the aims of this work: to demonstrate the importance of creating a framework to design these complex systems. A global picture of a general distributed surveillance system is given. The system is constituted by a diversity of components whose integration requires a complex analysis of the different requirements and functionalities.

Structural design model		CORBA	MASCOT3/DORIS
	Communication techniques	– Client/Server	– Paths and IDAs; Protocol taxonomy
	Distribution of components	– Static/dynamic invocations – CORBA objects	– Template substitution – Distributed protocol taxonomy – Subsystem/Activities Partitioning
	Run-time and scheduling policies	Yes (a variant of CORBA called TAO ORB Core uses pre-emptive strategy with priority based-connection)	Yes, the choice of scheduling algorithm is left to the designer
	Development aspects	No	Status progression and system building including mapping to distributed hardware.

Table 4-1. The concepts that will be compared between CORBA and MASCOT3/DORIS.

4.2 ***CORBA (Common Object Request Broker Architecture)***

As mentioned in the Introduction, this chapter is concerned with a comparative study between CORBA and DORIS. The latter has being extensively introduced in chapter 3, therefore in this chapter only CORBA will be introduced.

The Object Management Group is an international organisation founded in 1989, whose purpose is to define a set of interfaces for interoperable software. The OMG promotes the theory and practice of OO technology in software development [CORBA 2005]. The aims of OMG are the reusability, portability, and interoperability of object-based software in distributed, heterogeneous environments.

The first specification produced by OMG was CORBA [Henning and Vinoski 1999], which is an industry consensus standard and it can be considered as a possible solution for interoperability between applications. CORBA, at the time of writing, represents the next generation of client-server relationship that provides highly distributed systems and applications. CORBA assists in the creation of software architectures, but it does not design the software architecture itself [Mowbray and Zahavi 1995]. The distribution solution is defined using the OO paradigm, hiding different implementation languages, operating systems differences and object locations.

4.2.1 CORBA components

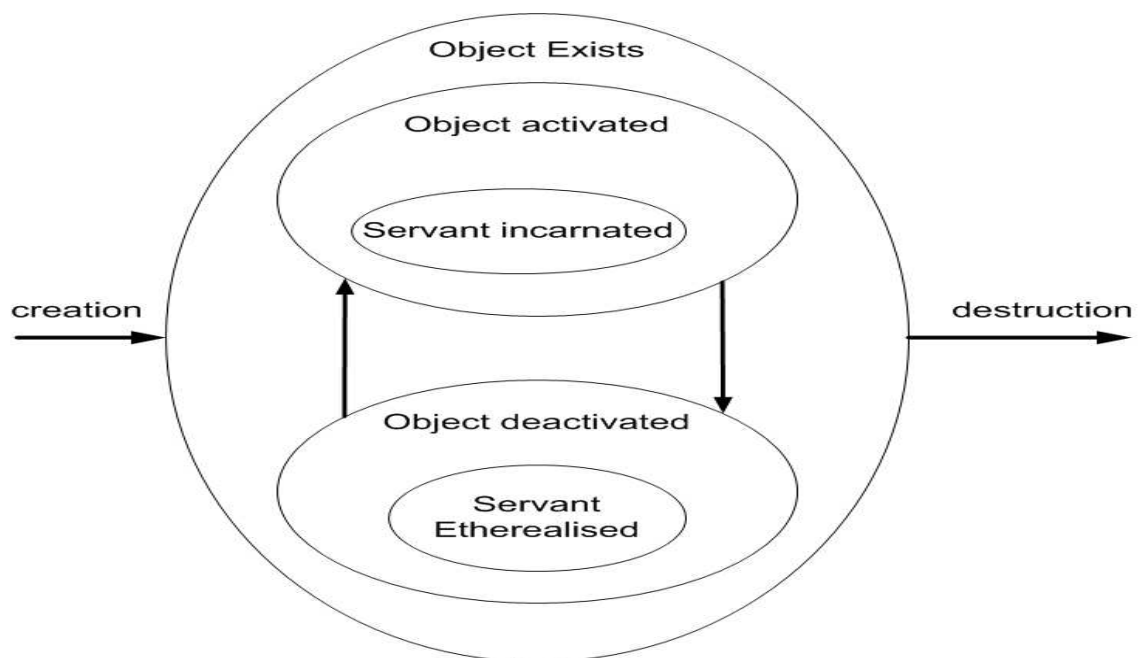
The main components that participate in the communication mechanism in CORBA, are listed next:

- CORBA object: is a “virtual” entity, which is located by the ORB, and it is able to deal with the requests coming from the client.
- Target object: is a CORBA object, which represents the object that has to deal with requests coming from the client side. This object exists in the context of CORBA invocations.
- Client object: represents the object that calls the CORBA object. There is a spatial decoupling between the client object and the CORBA object.
- Server object: is an application where one or more CORBA objects exist (see Figure 4-1). Like the target object, the server object only exists on the context of CORBA invocations.

- Object Reference: is a handle used to identify a CORBA object. For the client the object reference is an opaque entity (i.e. “black box” entity).
- Servant object: is a programming language entity (i.e. an instance) that incarnates a CORBA object. See Figure 4-1.

Figure 4-1 presents the different states that a CORBA object, in a server application, goes through to establish the communication between the server object and the client object. When the CORBA object is created an object reference is also created. Once the object is created, its state may alternate between an active status or a deactivated status, as is shown in Figure 4-1. While the object is in its active state and the servant is incarnated, it is able to receive and process requests coming from the client object. A CORBA object is incarnated only by a single servant at any point in time, although several instances of a servant can be created to represent the same CORBA object.

Note that the life cycle of the CORBA object and the servant are different; the CORBA object only exists in the context of creation and destruction whereas the servant object only exists when it is incarnated and it is destroyed when it is *etherealised*⁶.



⁶ Terminology used by [Henning and Vinoski 1999] to describe the servant state when is destroyed.

Figure 4-1.The states of a CORBA object and servant object life cycle [Henning and Vinoski 1999].

4.2.2 CORBA features

CORBA includes features aimed at accomplishing reusability, portability and interoperability between distributed integrated applications. The main features can be summarised in the following list and Figure 4-2 illustrates the relation between these features:

- OMG IDL (Interface Definition Language)
- Language mapping facilities and Application Program Interface (API)
- Static and dynamic method invocation
- Object Adapters
- Inter-ORB protocol

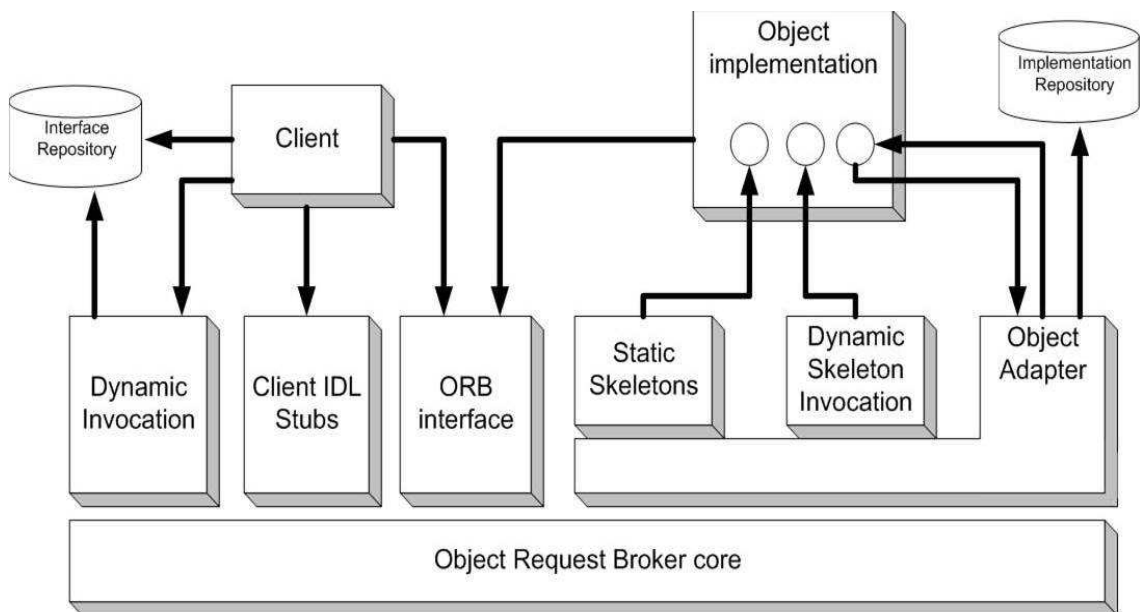


Figure 4-2.Common Object Request Broker Architecture (CORBA)[Henning and Vinoski 1999].

The OMG IDL is a strongly typed declarative language and an important notational tool for the software architecture in CORBA [Mowbray and Zahavi 1995]. OMG IDL specifies a coherent definition of interfaces. IDL provides a separation between

design and implementation because it has no implementation information, providing encapsulation of the different components and isolation between subsystems. Then, the question where to place OMG IDL interfaces becomes a design decision. The OMG IDL can be layered on top of any communication layer making then the application software independent from these underlying layers. The language mappings or bindings specify how IDL is translated into different programming languages, by defining which facilities of the programming language are used [Mowbray and Zahavi 1995].

The static and dynamic invocation facilities in CORBA allow the creation of method invocations at compile time or at run-time. In both types of invocation the client needs the object reference (e.g. ID object) of the remote object (server object) to create a request and to call the method that performs the service. The mechanisms to discover remote objects in CORBA can be achieved in three different ways (at the time of writing). First, it is possible to give the client directly a string with the CORBA object reference. The second way is done by obtaining the reference from the name of the object (through an intermediary provided by CORBA called a Naming Service). The third way is done by obtaining the reference from the type of service that the server provides (through an intermediary also provided by CORBA called the Trader Service). Object services like the Naming Services are a collection of system-level service interfaces that are included into the functionality of the ORB; these services are used to create a component, to name it, and introduce it to the system. CORBA provides run-time metadata for describing the server interfaces known by the system, which the client uses to invoke services at run-time. The IDL pre-compilers create this metadata automatically. The static method invocation can be defined like a conventional RPC but with polymorphism and inheritance properties included; e.g. the same method invocation can have different results depending on which server object deals with the call. The static interface in the client side is directly created through the client stubs by the IDL pre-compiler. Equally, at the server side, the static interface is created through the skeletons.

An Object Adapter is an object that allows the client to invoke requests on an object whose interface is unknown to the client (CORBA provides in its latest

specification [CORBA 2005] the Portable Object Adapter (POA)), see Figure 4-3. In a server application, the Object Adapter creates object references and ensures that each target object is incarnated by a servant object (see Figure 4-1).

Finally, an Object Adapter captures dispatched requests from the server side and redirects them to the corresponding servant, which incarnates the target object. The POA is illustrated in Figure 4-3, in which another component appears, apart from the ORB, called the POA manager that controls the requests that are sent to the POA.

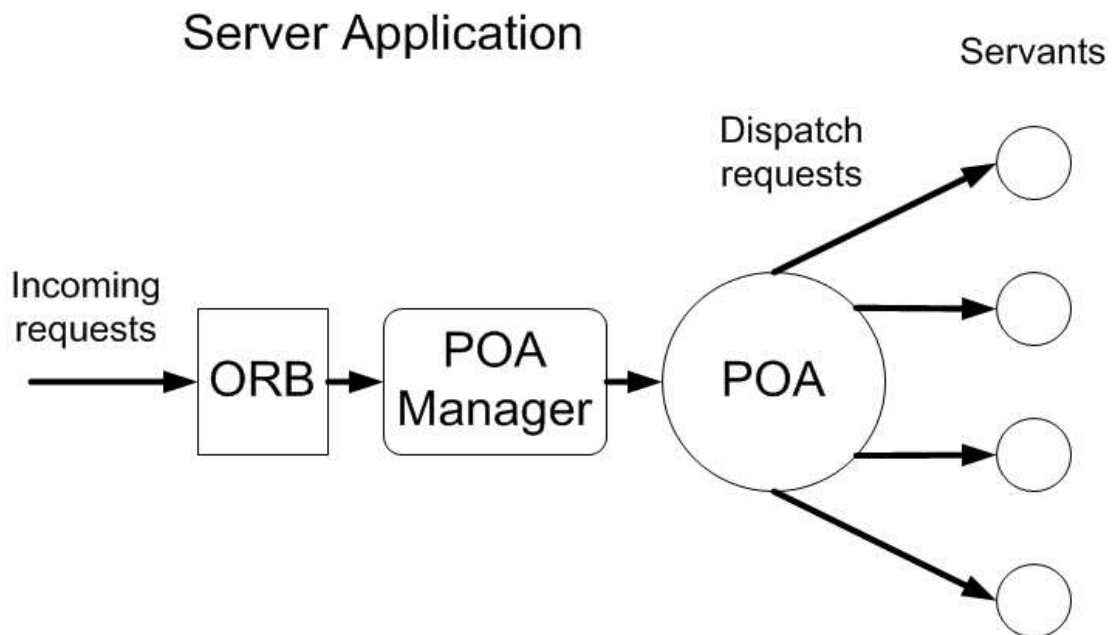


Figure 4-3. The flow of requests to the server side and how POA dispatches them.

The ORB can be defined as the object bus of CORBA. It lets the objects invoke and receive requests transparently; the client is not aware of the mechanisms used to communicate with the server objects. When a client invokes an operation, the ORB locates the target object, activates the server application and a servant if they are not activated. Furthermore, it transmits the arguments of the requests, waits for the results and returns the values of the call to the client, raising an exception when appropriate. Moreover, the ORB provides a variety of distributed middleware services as presented in the previous section and in [CORBA 2005]. The ORB allows objects to discover each other at run-time and to invoke services. Furthermore, each ORB must support an Interface Repository (see Figure 4-2),

which is a run-time repository of interfaces specifications of all the objects that the ORB⁷ recognises. On the other hand, the Implementation Interface (Figure 4-2) is a run-time database, which contains the actual implementation of the objects.

4.3 ADVISOR

ADVISOR represents, at the time of writing, one of the most advanced examples of a large distributed real-time surveillance system using OO technology. This section presents a description, i.e. requirements and system architecture, of this existing distributed real-time system. The first two sections explain the main features of the system as well as the overall goals of the system and present the requirements of the system based on the official specifications [ADVISOR 2003]. The following sections present the overall design of the system modules within ADVISOR. Finally, the last two sections explain the communication between the different modules that constitute ADVISOR, as well as explaining the type of data that the system has to deal with and the communication structure between the modules.

ADVISOR was developed as part of an EU-funded project on innovative architectures for public transport systems, focused mainly in metro stations. ADVISOR was created to provide assistance to the operators by increasing their efficiency to survey with many cameras available at the same time, but with a limited number of monitors (though the ADVISOR Prototype only worked with four cameras simultaneously). Therefore, ADVISOR was created to generate better use of transport infrastructure by improving safety and security environment e.g. in metro stations.

4.3.1 Specifications of the ADVISOR system

The ADVISOR system is intended to fulfil a set of requirements. The following list represents some of the initial requirements:

⁷ To create interoperability between CORBA software architectures, the CORBA specification includes Inter-ORB protocols like General Inter-ORB Protocol (GIOP) or Internet Inter-ORB Protocol (IIOP), which specify a set of message formats and common data representations for communications between ORBs [Henning and Vinoski 1999].

- ADVISOR is a machine vision system, capable of monitoring all CCTV cameras in an installation. The computer vision techniques operate on compressed digital video inputs.
- The goal of ADVISOR consists in assisting human operators by automatic selection, recording and annotation of “interesting” images as far as “abnormal” crowd and individual behaviours are concerned. The system is intended to enhance the effectiveness of the surveillance operation of any installation.
- The ADVISOR system uses an open and scalable architecture approach so that it is possible to develop algorithms, which can be “plugged into” the system, taking appropriate inputs and generating appropriate outputs.
- ADVISOR runs on standard commercial hardware with an interface to a wide bandwidth video distribution network. The software is implemented on a local network of processors communicating via an open software standard for distributed processing. An Object Request Broker (ORB) for the software environment is used to develop a scalable system suitable for installation in a wide range of locations using a distributed computing environment.
- ADVISOR interprets shapes and movements in scenes being viewed by the CCTV in order to build up a picture of the behaviour of people in the scene.
- That means the system is capable of interpreting the behaviour and deciding whether such behaviour represents a significant event.
- ADVISOR detects the anomalous events with high probability with low false alarm rate. The system alerts operators in real time.
- ADVISOR stores all video output from cameras. Storage capability allows continuous recording. In parallel with recording multiple video inputs, the archive function stores commentary of associated sequences (known as annotations). Therefore, the archive can search for video sequences, which match keywords in the notation data or according to specific times. Retrieval of video sequences takes place alongside continuous recording.

4.3.2 Specifications that ADVISOR Prototype did not accomplish

As mentioned, ADVISOR was developed as a part of an EU-funded project, therefore at the end of the project; a prototype had to be built. Some of the initial requirements of ADVISOR system that are not accomplished by the prototype, are presented in the following list:

- ADVISOR system is able to improve the performance in detection and recognition of anomalous events by learning via operator's feedback. This requirement is not carried out by the ADVISOR prototype; because in the prototype, the user (i.e. operator) is not able to change any parameter of the system.
- The requirement states that ADVISOR is immune to long periods of loss of video inputs. It is also immune from step changes in scene so that, a number of short video sequences can be assembled and replayed into the algorithms to demonstrate certain kind of behaviours.

Therefore, ADVISOR was the paper design in the EU-funded project of the same name, while the ADVISOR Prototype was its practical realisation that involved some limitations to the original paper design. Sections 4.3.3 and 4.3.4 present the design of ADVISOR specifications, because the ADVISOR Prototype does not accomplish all mentioned specifications, any modification to the final ADVISOR Prototype design is reported in the presented ADVISOR design sections. Note that in sections 4.5 and 4.6, the final designs correspond to the design of ADVISOR Prototype rather than the ADVISOR system, which are slightly different.

4.3.3 ADVISOR system architecture design

ADVISOR is a semi-automatic surveillance system that can be made up of one or more Human Computer Interfaces (HCI) and one or more Advisor System Units (ASU) as illustrated in Figure 4-4. Each HCI can be connected with up to four ASUs and each ASU can be connected up to two HCIs. The ADVISOR prototype (a demonstrator tested at Barcelona's Sagrada Familia metro station) consists of two HCIs and one ASU unit. One HCI is installed at a remote control centre and the other HCI, which is used mainly for debugging purposes, is installed at the same

place of the ASU module, see Figure 4-5. The hardware platform of the ADVISOR prototype consists of six PCs. In the prototype, the HCIs were designed to run on a PC as a standalone process. The ASU software module consists of two software modules: the Image Processing Unit (IPU) and the Symbol Processing Unit (SPU), see Figure 4-6. The IPU consists of four software modules and the SPU consists of two software modules, as shown in Figure 4-7 and Figure 4-8. Each of these software modules inside the IPU and SPU were originally designed to run on a standalone PC. However, the final hardware mapping in the prototype, consisted in having three software modules in one PC and each of the three remaining software modules on a separate PC. All the controllers in the ASU module (the ASU controller, the SPU controller and the IPU controller) resided together in an ASU control process in one of the PCs. The system has a “hub” topology whereby an ASU does not have links with other ASUs, thus there is no communication between them. In the same way, HCIs do not communicate between them either (see Figure 4-4). An ASU has both a maximum processing capability and storage capacity, therefore there is an upper limit to the number of cameras that one ASU can handle (in [ADVISOR 2003] is stated that the limit is around 10). The ADVISOR prototype system is capable of operating with up to four camera inputs simultaneously at 5 frames per second.

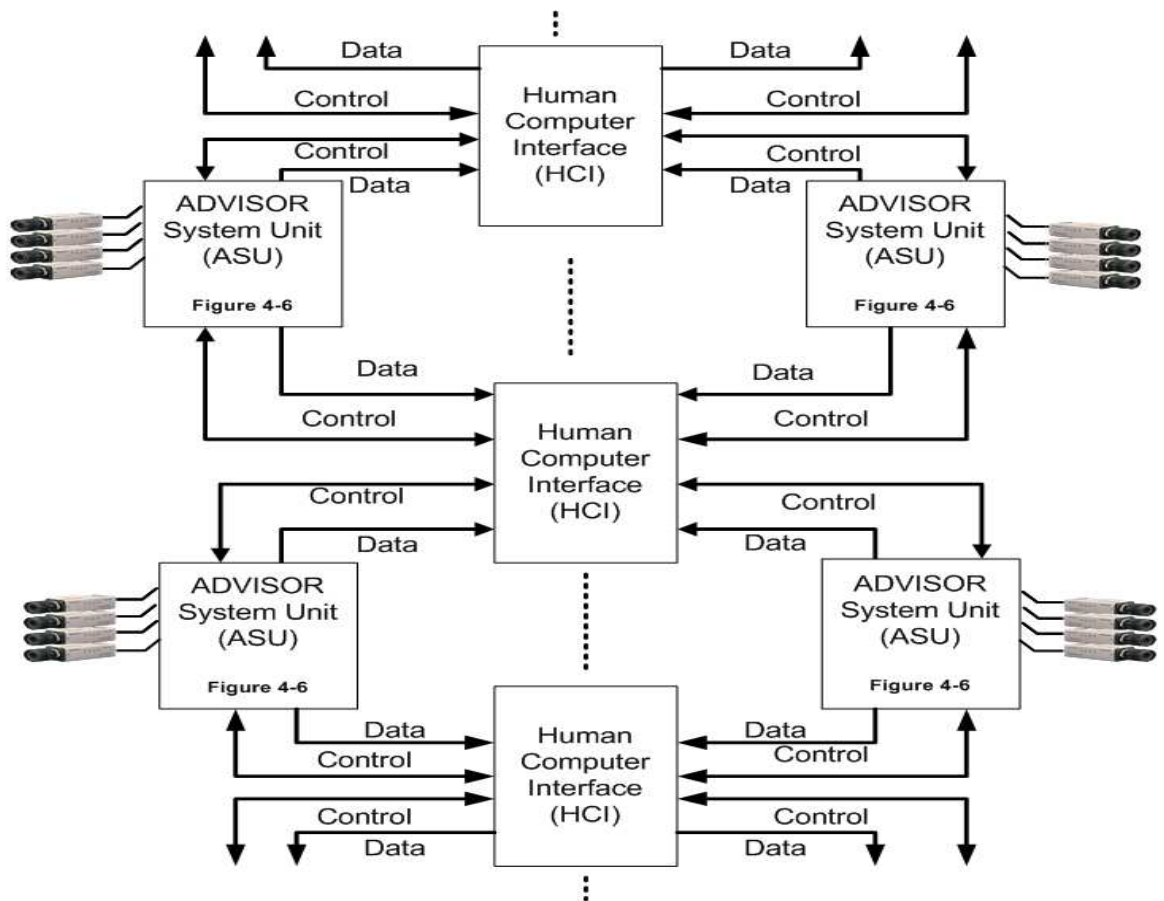


Figure 4-4. Logical view of ADVISOR system.

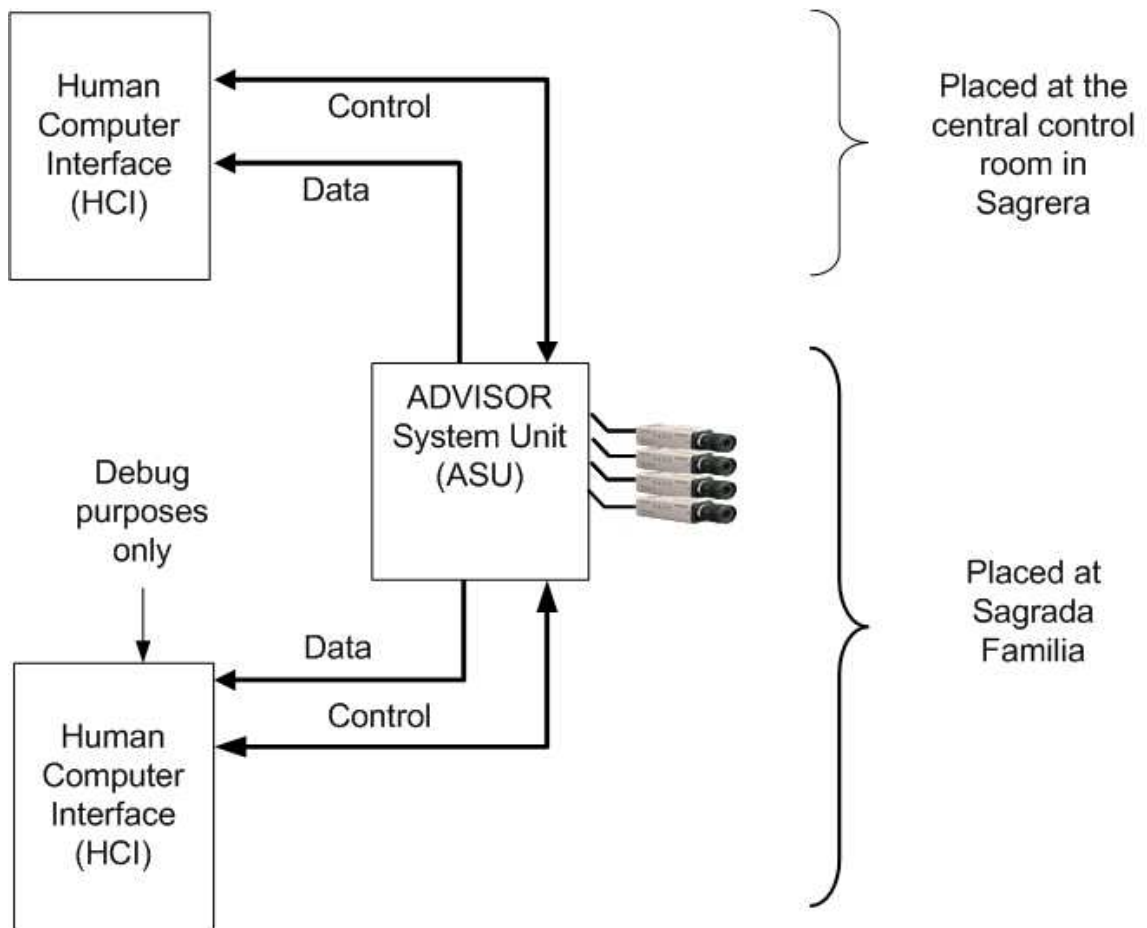


Figure 4-5. The logical view of the ADVISOR Prototype (tested at Barcelona).

To an operator, the ADVISOR system presents itself as a single application that resides on their computer, i.e. the HCI. Therefore, only an HCI is configured to control and start the system up. Moreover, an HCI may dynamically configure selected ASU parameters. The HCI in the ADVISOR Prototype only starts-up the system and requests archived or live images from the system to visualise. Therefore, the HCI of the ADVISOR Prototype is unable to change any configurable parameter of the system but can only change the parameters related to the visualisation of live or archived images.

Each ASU could contain one ASU controller, one SPU module and one or more IPU modules. However, the ASU of the Prototype only contains one IPU module, as can be seen in Figure 4-6. In the ASU module, the SPU and IPU are slaves (i.e. clients) and they are not aware of each other's presence in the ASU. The IPU module contains one IPU controller, one Image Capture module, one Motion Detector module, one People Tracker module and one Crowd Monitor module, see

Figure 4-6. The SPU module contains one SPU controller, one Behaviour Recognition module and one Archive module, see Figure 4-8.

In terms of the communication between the HCI and modules that reside inside the ASU, the HCI communicates directly with all the modules that are inside an SPU module (i.e. the Archive module and the Behaviour module). On the other hand, an HCI can only communicate directly with one of the IPU module's (i.e. Image Capture module). Therefore, an HCI may take live camera feeds, which are in compressed form, from the Image Capture module, it may also take live annotations such as alarm messages raised by the recognition of a given situation, from the Behaviour module. Moreover, HCI may also take recording sequences and annotations from the Archive module. An HCI may search in an Archive module using different criteria: by time, by camera, by type of event, station or date.

4.3.3.1 ASU module

As mentioned before, each ASU operates independently of any other ASU. The ASU can only communicate with the HCI through one bidirectional control channel⁸ and several data channels. The ASU Controller has a management role and its job is to control its SPU and its IPU's. The ASU Controller:

- Supervises the start-up and close-down of the SPU and the IPU's through the SPU/IPU controllers.
- Establishes the appropriate connectivity (channels) between the SPU and the IPU's.
- Provides the primary point of contact with the HCI through a single bidirectional control channel.

⁸ Bear in mind that in ADVISOR, the communication links are called "channels" even though they do not have any connection with the channel protocols of MASCOT/DORIS discussed in chapter 3.

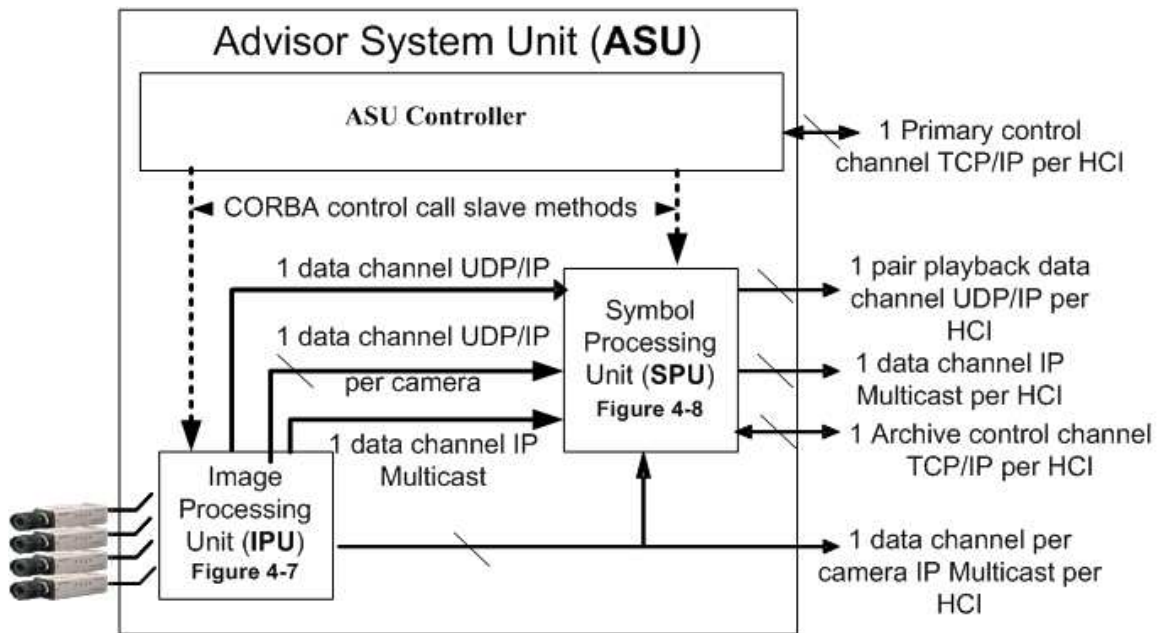


Figure 4-6. Top level design of ADVISOR System Unit (ASU). Note that, the ADVISOR Prototype consists of one HCI that is connected to one ASU.

4.3.3.2 IPU module

Figure 4-7 illustrates an Image Processing Unit or IPU module, whose functionality is based on capturing the output from a number of cameras as sequences of JPEG (Joint Photographic Experts Group) images, and then on running various ‘machine vision algorithms’ on these sequences generating low-level observations. The low-level observations are expressed in XML (Extensible Markup Language) format, and sent through a data channel to the SPU. The IPU module (in ADVISOR system) consists of five distinct components: The IPU Controller, Image Capture module (Image Capture CORBA Object), Motion Detector module (Motion Detector CORBA Object), People Tracker module (People Tracker CORBA Object) and Crowd Monitor module (Crowd Monitor CORBA Object). Although in ADVISOR Prototype, the People Tracker and the Motion Detector module are implemented in the same module. As mentioned before, the IPU Controller is a slave to the ASU Controller, as shown in Figure 4-6 and Figure 4-7. At the start up of the system, the IPU Controller dynamically configures selected IPU parameters, such as camera state, that relates to a single component such as Capture module. Therefore, to configure a parameter of the Capture module (for example), the controller simply calls the appropriate method. This is shown by the dashed lines in

Figure 4-7. Note that as mentioned, the specification of changing parameters dynamically is not implemented in the ADVISOR Prototype, only the dynamic configuration of parameters (at the start up of the system) is implemented.

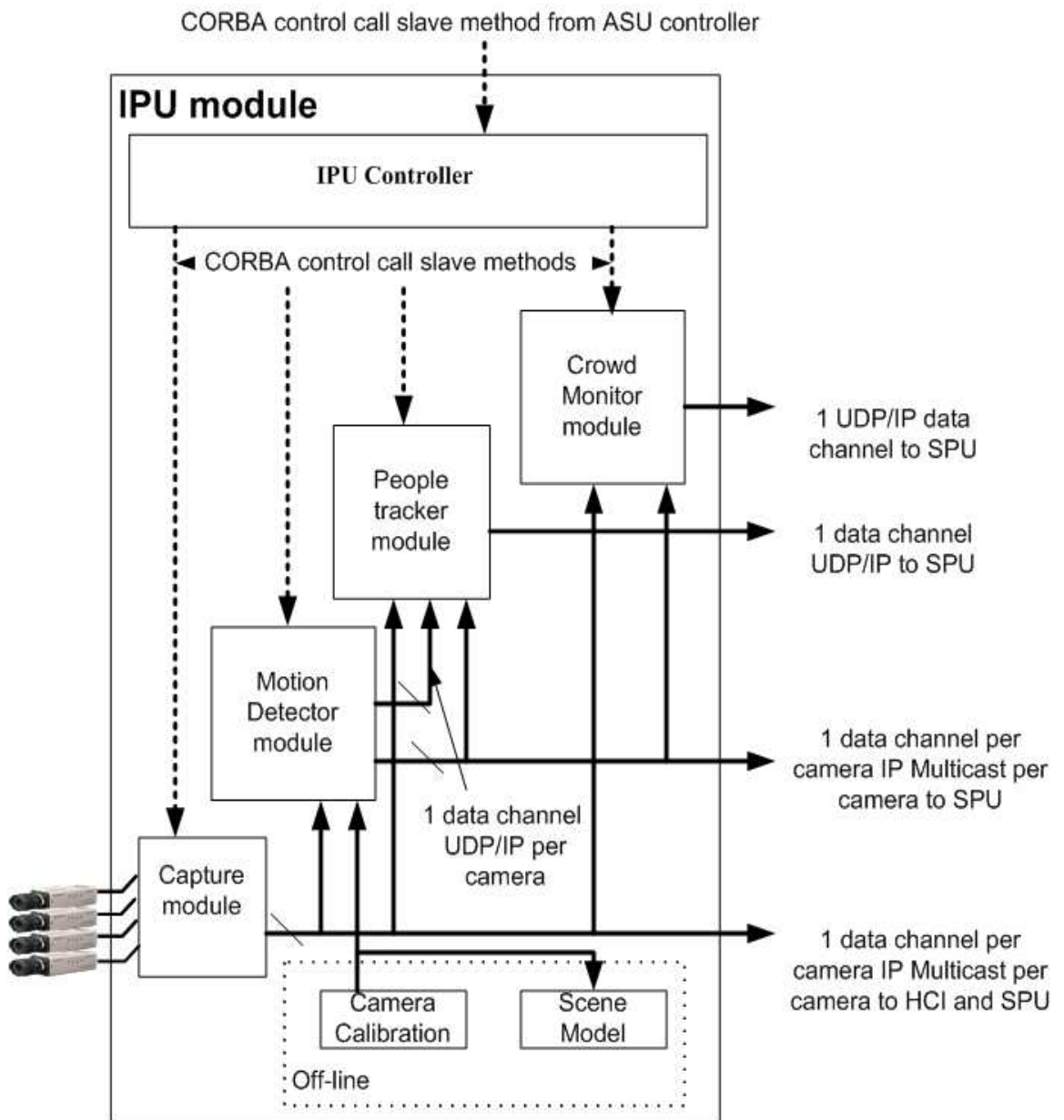


Figure 4-7. Top level design of Image Processing Unit (IPU).

4.3.3.2.1 Description of different parts in the IPU module

Different parts that constitute the IPU module are described by diagrams in Appendix B (pp. 280-293), to give an appreciation of the amount of processing capacity and data requirements that a small distributed surveillance system like the one presented in this case study can require. Bear in mind that this case study is a

surveillance system with a maximum of four cameras. The description of each part is followed by design diagrams representing the implementation of the vision algorithms that appear in each submodule inside an IPU module. Therefore, the diagrams presented in Appendix B (from Appendix B-1 to Appendix B-10) show each of these sub modules as a white box. These diagrams have been made from the information extracted from the specification documents of ADVISOR [ADVISOR 2003]. Therefore, they outline the information that has been extracted from the specification documents. Note that the Tracker module is not represented because it did not appear in these specification documents of ADVISOR.

4.3.3.3 SPU module

Figure 4-8 illustrates the three components inside the Symbol Processing Unit (SPU): a master SPU Controller that manages the rest of components (i.e. The Behaviour Recognition CORBA object and the Archive CORBA object). The SPU stores JPEG image sequences on disk that are sent by the IPU to the Archive module. In the Behaviour Recognition module, machine vision algorithms are run to generate high-level observations from the low-level observations that are sent to the SPU by the IPU. The obtained higher-level observations are also stored in the Archive module, which sends them to the HCI on demand, in XML format, through a single data channel. The last functionality of an SPU module consists in allowing the HCI to do search and retrieval operations with stored image sequences and observations. Various control and data channels are used to support this facility.

As mentioned before, the SPU Controller is slave of the ASU Controller. In the same way as the IPU Controller, the SPU could configure dynamically selected SPU parameters. See discontinuous arrows in Figure 4-8. As mentioned in the previous section, the specification of changing parameters dynamically is not implemented in the ADVISOR Prototype.

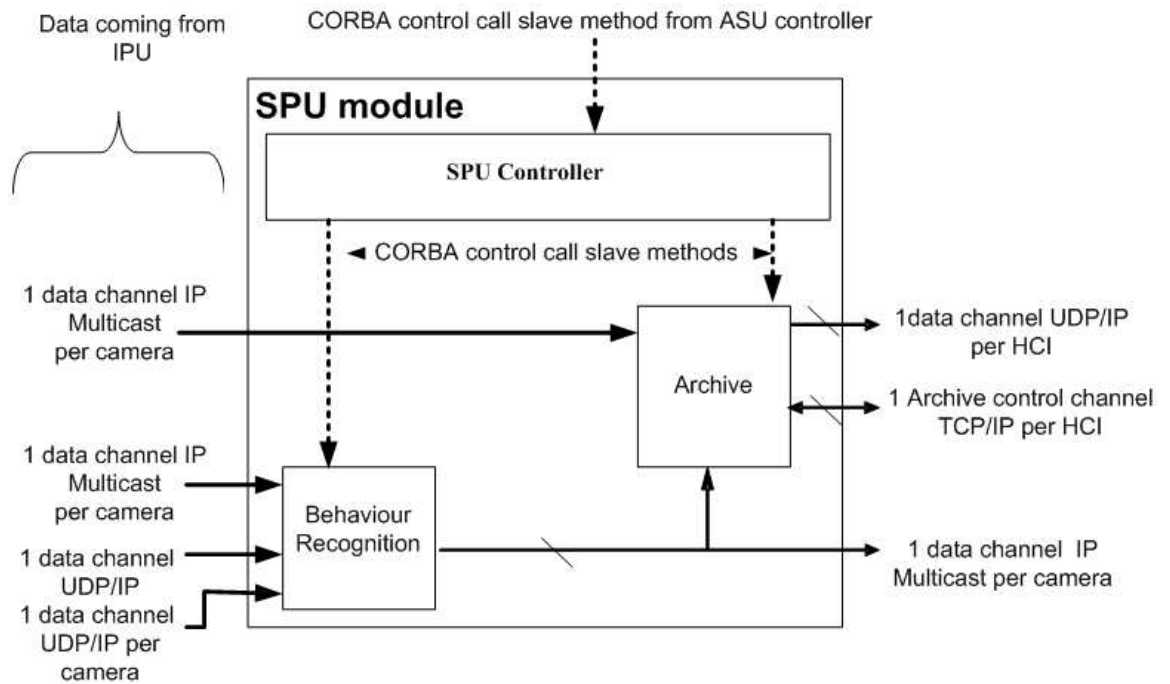


Figure 4-8. Top-level design Symbol Processing Unit (SPU).

4.3.4 Data types in the system and communication between modules

All the communications occurring between an HCI and an ASU could be categorised as either control communication or data communication.

- Control communications, also called transactions, have the following constraints:
 - Only an HCI initiates transactions. All messages are in XML based format.
 - When the HCI requests an ASU to do something, the ASU attempts to do it and responds appropriately. Only an HCI initiates transactions (at the operator's request, even though as mentioned in the specifications, the HCI in ADVISOR Prototype only initiates transactions related to data visualisation, to retrieve archived or live images from the system).
 - No more than one transaction can be in progress.

- As mentioned before, an HCI might dynamically configure selected ASU parameters.
 - A control communication has priority above data communication.
- Data communication consists in large amount of information sent from the modules, usually over a long period of time. The information takes the form of compressed images and annotations. It only goes from ASU to HCI.

Control channels have been designed to transmit control data, sent in XML format, from/to the ASU to/from the HCI. The traffic relating to the control channel consists in command traffic (from the HCI to the ASU), responses traffic (from the ASU to the HCI) and events traffic (from the ASU to the HCI). Events might be configured to trigger alarms. Once an alarm is triggered, the HCI screen is automatically switched to the scene view, where the event is recognised.

Data channels have been designed to transmit data from/to the ASU to/from the HCI/ASU. The following list describes the number of data channels that each module has and also the type and format of this data and from/to which modules the data are sent:

- From the Archive to the HCI: Up to four pairs of channels (one pair per HCI client). Each channel pair had one playback image channel and one playback annotation channel in XML format. The transmission of Playback image channels are realised at five frames per second (fps) in JPEG format.
- From the Image Capture module (IPU) to the HCI: Up to four data channels (one per camera). The transmission of data channels in JPEG format at five fps using YUV colour format, is done through Internet Protocol (IP) multicast communication.
- From the IPU to the SPU: Up to nine data channels of low-level annotation in XML data format. Each output stream has its own data channel going to the SPU; e.g. the output stream resulting from the

detection of people in the scene, is sent to the SPU through data channel (see Figure 4-7). The objects detected and marked as so-called ‘blobs’ are sent through other data channels. The outputs resulting from the detection of crowd situations in the scene are also sent through data channels. Each data channel is linked to specific image input channel (i.e. one channel per camera). Only the data channel that sends the detection of people in the scene, sends the information of all image input channels through only one channel.

Several instances of each type of channel may be active simultaneously. However, at any point in time any given HCI may listen to a maximum of eight data channels. Therefore, as mentioned before, any HCI can only be connected to up to four ASUs (this is shown in Figure 4-4). Consistent with the overall ADVISOR architecture, CORBA has been used in the ADVISOR Prototype as an integration platform of each module. Events, alarms and control data are also sent to the corresponding module through the distributed object computing ability of CORBA. Therefore, the next section presents and discusses the CORBA architecture design used in the ADVISOR Prototype.

4.4 The CORBA architecture design implemented in ADVISOR Prototype using DORIS graphical notation

Following the brief introduction of the main features and components of CORBA, Figure 4-9 illustrates the CORBA elements described in Figure 4-2 that have been used in the design of ADVISOR. Bear in mind that, as mentioned in the Introduction, the DORIS graphical notation is used because CORBA does not have any graphical notation associated to express its architecture designs. The DORIS graphical notation also has been used allowing to compare, using the same graphical tools, the different architecture designs of ADVISOR system using both technologies (CORBA and DORIS). One design decision in ADVISOR is that only static invocations are implemented. Therefore, there is no need to use an interface repository (see Figure 4-2) on the client side. However, the main design decision is to consider the ASU, IPU and SPU controllers as servant objects (i.e. CORBA objects). The other modules, i.e. the IPU and the SPU modules are considered as ‘clients’. In this way, the IPU consists of three client objects (camera object, crowd

monitor object and motion object) and the SPU consists of two more client objects (i.e. archive and behaviour object). The HCIs (i.e. the local and central HCI) are designed as objects that communicate between them or with the archive and behaviour client objects; they do not have any direct relationship with servant objects. This decision is derived from the fact that client objects in the IPU and the SPU modules need information that is needed by the appropriate CORBA objects (i.e. camera CORBA object, crowd monitor CORBA object, motion CORBA object, archive CORBA object and behaviour CORBA object).

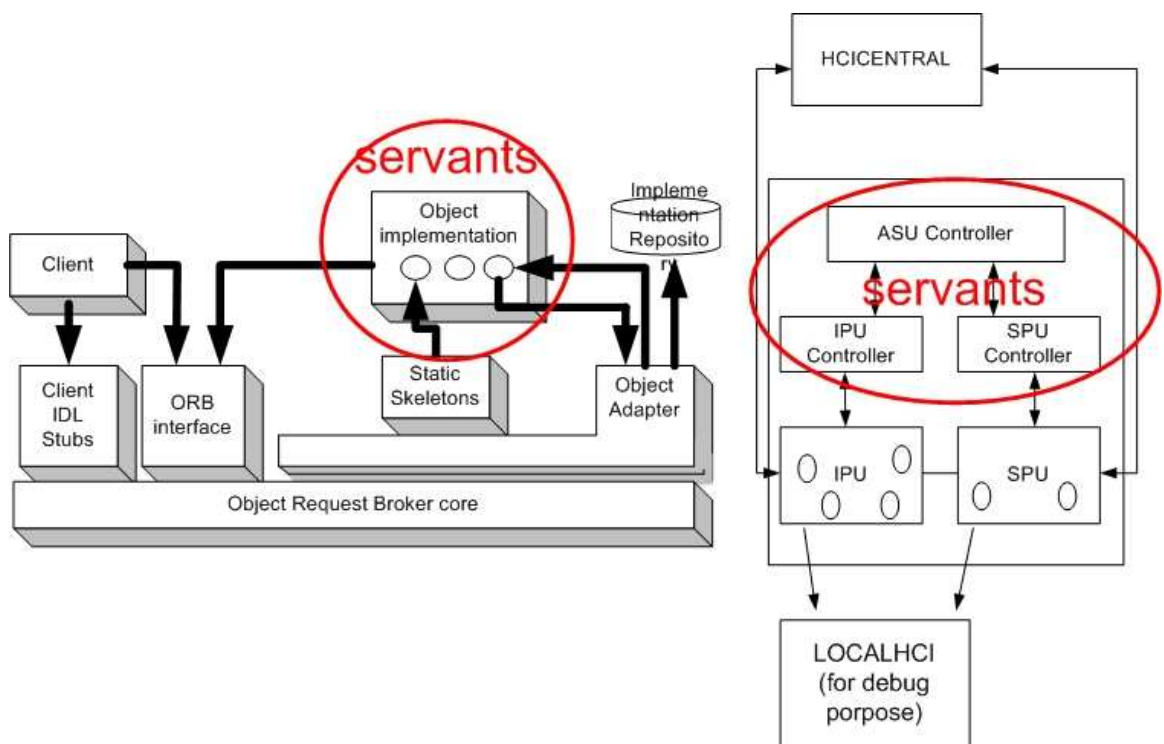


Figure 4-9. Links between the CORBA features illustrated in Figure 4-2 with the CORBA design of ADVISOR.

Figure 4-10 illustrates the representation of ADVISOR CORBA design using the DORIS graphical notation. The client objects are represented as subsystem components grouped in a subsystem called ASUSUBSYSTEM (see top of Figure 4-10). The HCIs objects are represented as activities inside other subsystems called HCICENTRAL and LOCALHCISTATION, which are presented in further sections. Notice that the subsystems inside ASUSUBSYSTEM communicate via IDAs as seen in Figure 4-10 (see the rectangle components that appear inside ASUSUBSYSTEM figure). These components are added because it is not possible to connect directly two active components (i.e. each of the subsystems that appear

inside ASUSUBSYSTEM) directly using MAscot Design Generator (MADGE) tools which follows the RTN principles. One of the principles established in RTN states that the communication between two active components is always established through a passive element called IDA. Therefore, MADGE does not allow designing architectures connecting two active elements directly.

On the other hand, the servants are represented as activities inside the subsystem called CORBA_SUBSYSTEM (see bottom of Figure 4-10). The Object Adapter (i.e. POA) is represented by two other activities inside the CORBA_SUBSYSTEM called OA_IN and OA_OUT. These two activities receive and send the requests from the client objects in ASUSUBSYSTEM. The implementation repository, whose function is to store information about the locating servants, is used by the OA_IN activity and is represented by a “datarepository” component called IMPLREPOSITORYSUBS subsystem (see Appendix C-22, pp. 306, Figure C-22).

Finally, the ORB interface and the ORB core (see Figure 4-10), which as mentioned represent the object “bus” of CORBA, whose functionality and implementation is transparent to the CORBA designer, is represented by components inside the subsystem called COM (see section 4.4.1). This subsystem consists of a group of distributed protocols introduced in chapter 3. Some of these protocols represent the client-server relationship used in CORBA.

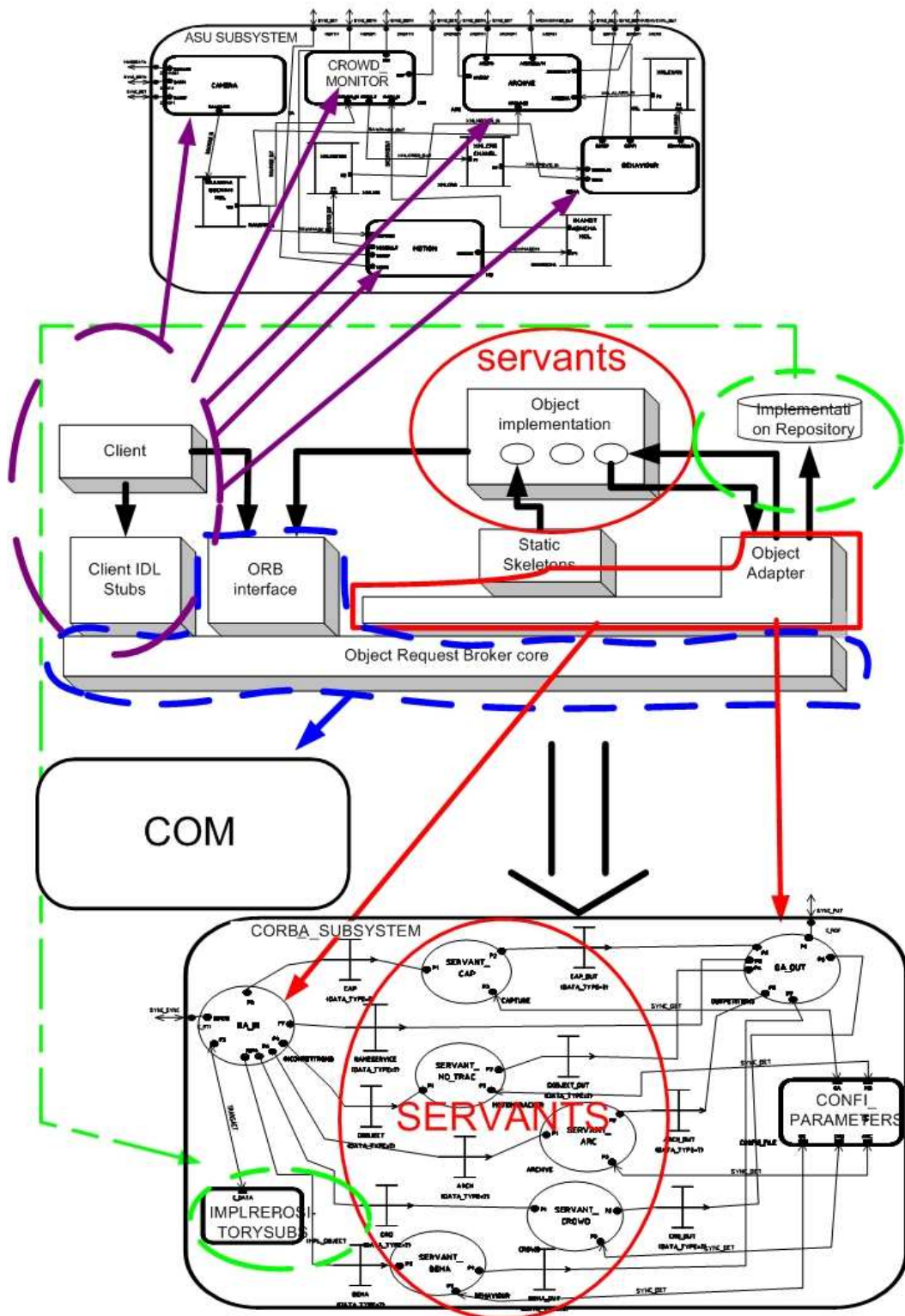


Figure 4-10. The CORBA design of ADVISOR using DORIS graphical notation. ASU SUBSYSTEM (Appendix C-14, pp. 298), COM (Appendix C-13, pp. 297), CORBA SUBSYSTEM (Appendix C-15, pp. 299), IMPLREPOSITORYSUBS (Appendix C-22, pp. 306), CONF_PARAMETERS (Appendix C-21, pp. 305), CROWD_MONITOR (Appendix C-20, pp.304).

4.4.1 ORB and COM subsystem

As mentioned in the previous section, the COM subsystem (see Figure 4-10) represents the ORB CORBA bus. Figure 4-11 presents the configuration of the COM subsystem using a DORIS notation. One of the main differences between both technologies (CORBA and DORIS) is based on the communication mechanisms. On one hand, DORIS provides, as discussed in chapter 3, an extensive taxonomy of protocols that allows the designer to choose the communication mechanisms that are best suited for that application. The CORBA's architecture is based on different layers [Mowbray and Zahavi 1995]; one of these layers is called the communication layer that is handled by the ORB. The communication layer deals with the communication in distributed environments between the components in the application layers. Thus, the ORB technology deals with the communication between objects in any CORBA application. Therefore, the designer is unaware of the type of communication mechanisms used. From a commercial point of view, this fact may be taken as an advantage, but from a system engineering point of view, it is always important to know what and how the system is designed to have a great understanding of the system (not everything may be suitable for each application). Therefore, we tried to illustrate the ORB architecture design by using only the DORIS graphical notation. In other words, how the ORB communication mechanism should be established (its architecture) for ADVISOR application (the Prototype) to be able to compare in section 4.7 both technologies. Once again, some elements presented in this section and in section 4.5 have been added for tool constraints. The COM subsystem consists of three other subsystems called COMSUBSYSTEM, MULTIDISTRIBUTION and DISTRIBUTSIGNAL. The composition of these subsystems is included in Figure 4-11, although to see the details of each subsystem it is best to refer to Appendix C-13, C-19, C-17 and C-18 respectively. The functionality of the COM subsystem is to transmit signals coming to and/or from CORBA_SUBSYSTEM and HCIs to and/or from the ASUSUBSYSTEM. Moreover, the COM subsystem de-multiplexes/multiplexes the signals coming to and/or from the CORBA_ SUBSYSTEM to and/or from the ASUSUBSYSTEM.

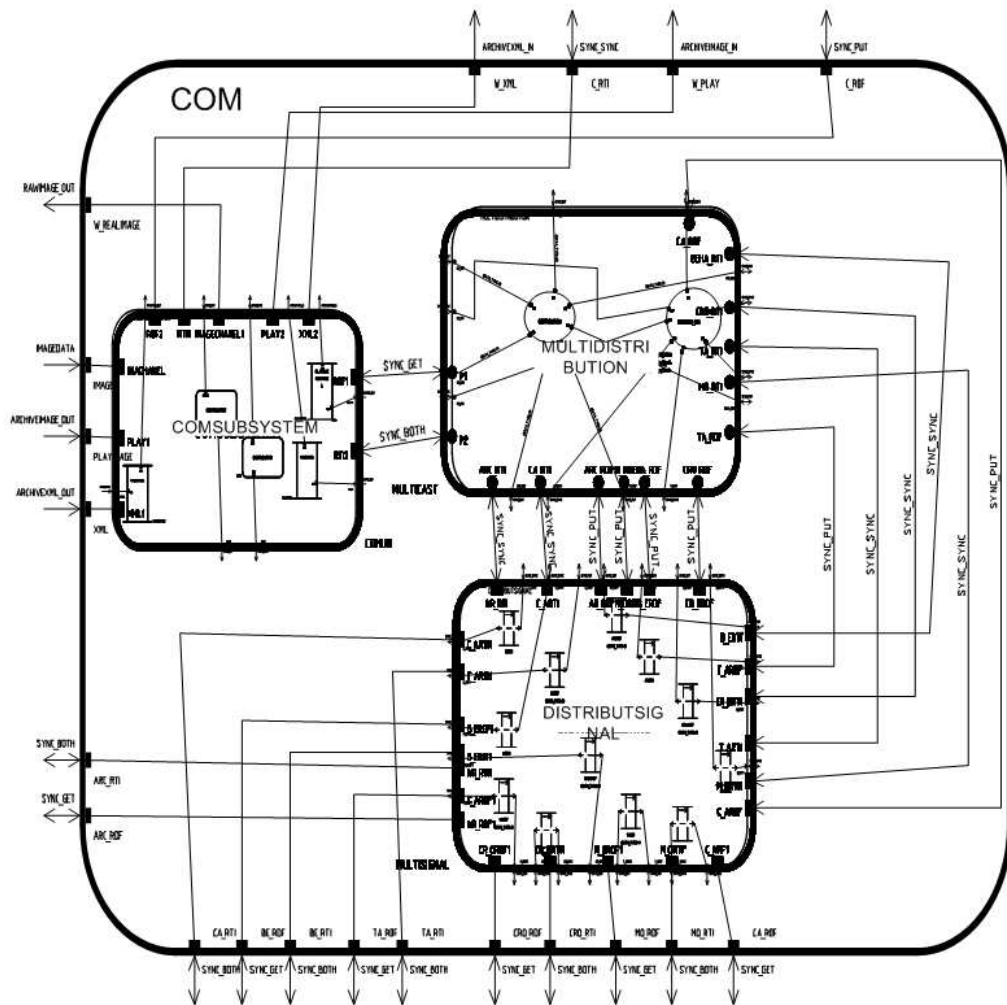


Figure 4-11. The configuration of COM subsystem (Appendix C-13, pp. 297). See Appendix C for more detailed view of each of the components that appear in this figure: COMSUBSYSTEM (Appendix C-19, pp. 303), MULTIDISTRIBUTION (Appendix C-17, pp. 301), DISTRIBUTESIGNAL (Appendix C-18, pp.302).

COM deals with two types of signal coming from the CORBA_SUBSYSTEM: the signals coming from the OA_IN activity and the signals coming from OA_OUT. OA_IN activates the corresponding servant whose function is to send the information that is in CONFIG_PARAMETERS subsystem (see Figure 4-10) to the corresponding client. Therefore, OA_IN activates an activity and OA_OUT sends the required data to the client. These two interactions are represented by two different protocols. The first interaction is represented by a RTI protocol and the second interaction is represented by a RDF protocol (please see Protocol Taxonomy in chapter 3). The use of RTI implies a “thread activation” on the server side i.e. the

client does not send data but a request of a thread activation. Therefore, when OA_IN receives a request from the client side, it activates the corresponding servant. Once the servant is activated and does the work, the client can receive the data that it needs through the RDF protocol. The use of this protocol implies that the client sends a request of data and waits until it receives the data from OA_OUT activity.

Figure 4-12 presents the components used in COM to transmit the signal coming from OA_OUT activity. The bottom of the Figure 4-12 illustrates the idea that the functionality of all these components is reduced to the use of one RDF protocol. Nevertheless, the graphical design presents all these components for many reasons: firstly, because it is a distributed solution (i.e. CORBA_SUBSYSTEM resides in one computer and ASUSUBSYSTEM resides in other computer) and therefore the protocol needs to be stretched (see DISTRIBUTRDF). Secondly, because the signal is demultiplexed (there are more than one client and server). Finally, because of the tool constraints (e.g. it is necessary to use four subsystems instead of grouping all these components in one subsystem like in Figure 4-12).

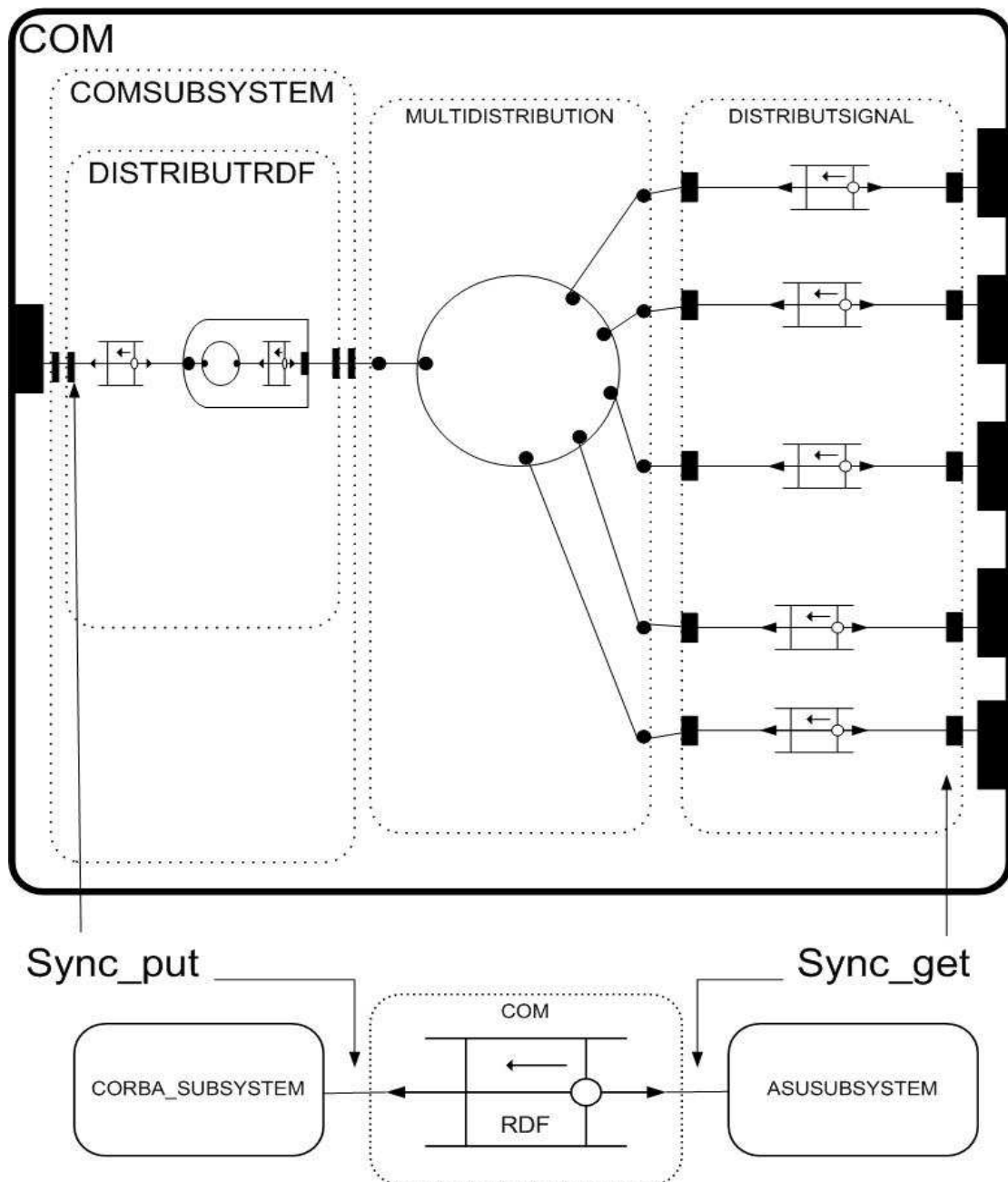


Figure 4-12. The whole path of one of the signals coming from CORBA_SUBSYSTEM to ASUSUBSYSTEM. This signal comes from OA_OUT inside CORBA_SUBSYSTEM (see interface called Sync_put) to one of the objects inside the ASUSUBSYSTEM (see interface called Sync_get).

4.5 The ADVISOR Prototype architecture using CORBA platform technology in DORIS notation

The ADVISOR Prototype discussed in this chapter was demonstrated in the Barcelona Underground in March 2002. As mentioned before, it consisted of two physical nodes: one node was placed in the Central Transport Control Room and it

consisted of one HCI unit. The other node was located in one of the underground stations (Sagrada Familia) and it consisted of one ASU unit and another HCI that was installed on a standalone PC, as shown in Figure 4-5. The ASU unit was made up of four CPUs: one CPU hosted Capture module, another CPU hosted Crowd Monitor module, another hosted Archive module and the last CPU hosted the Motion and Tracker modules altogether.

In section 4.3.1, the overall requirements of the ADVISOR system have been presented. In section 4.4 and 4.4.1, the CORBA architecture design used in ADVISOR system has been presented using the graphical DORIS notation (as mentioned, used just as a graphical tool). As mentioned in the Introduction, in sections 4.5 and 4.6, two different ADVISOR system architecture designs are presented, which have been created in this thesis. In section 4.5, the ADVISOR system architecture design using CORBA is discussed and illustrated using the DORIS graphical notation. In section 4.6, a new ADVISOR system architecture design is presented using this time RTN concepts and also the DORIS graphical notation. Both architectures are created from the same requirements taken from the ADVISOR Prototype presented early in this work. The reason behind this is to depict, in section 4.7 using the same graphical notation, the differences between the two architectures, which use different conceptual principles.

A variation in both architecture designs of the presented ADVISOR Prototype has been introduced to discuss properties such as distribution; it would have been rather difficult to discuss with only two nodes (one data processing node and one HCI) as it is on ADVISOR Prototype. The variation consists of using four nodes distributed in different physical places instead of two nodes (see the blue line on the left of Figure 4-13).

4.5.1 The decomposition of ADVISOR architecture design

ADVISOR has been created in a way that reflects the human hierarchical structure of transport surveillance systems, which have a central control node, situated usually in the central control room of the transport system, and local control units located across the area covered by the transport network. Figure 4-13 illustrates the

ADVISOR system and ADVISOR Prototype and also the decisions that have been taken to present the designs of ADVISOR Prototype using MADGE tools (i.e. ADVISORSYSTEM in section 4.5 and in section 4.6). Note that, as mentioned in section 4.3.2, in section 4.5 and 4.6 the ADVISOR Prototype system (with some added variations) is used instead of ADVISOR system. Moreover, in Figure 4-13, a blue line represents one of the variations added into the illustrated design (as mentioned at the end of the previous section). One variation consists in representing four nodes in ADVISORSYSTEM instead of the two nodes that the ADVISOR Prototype has. The other variation is to consider an HCI central like in ADVISOR system but also, to consider a local HCI in each ASU node, representing the HCI that ADVISOR Prototype has for debugging purposes.

Therefore, the modules used to represent the design of ADVISORSYSTEM are the following:

- The HCICENTRAL module, which represents the HCI central in ADVISOR system (see left of Figure 4-13).
- The ASUSUBSYSTEM module, which represents the IPU and SPU modules of ADVISOR system.
- The LOCALHCISTATION that represents the ASU, IPU and SPU controllers and the local HCI that the ADVISOR Prototype uses for debugging.
- The LOCALHCISTATION and ASUSUBSYSTEM that are grouped to constitute the LOCALDATAPROCESSING module, which represents the ASU module in the ADVISOR system.

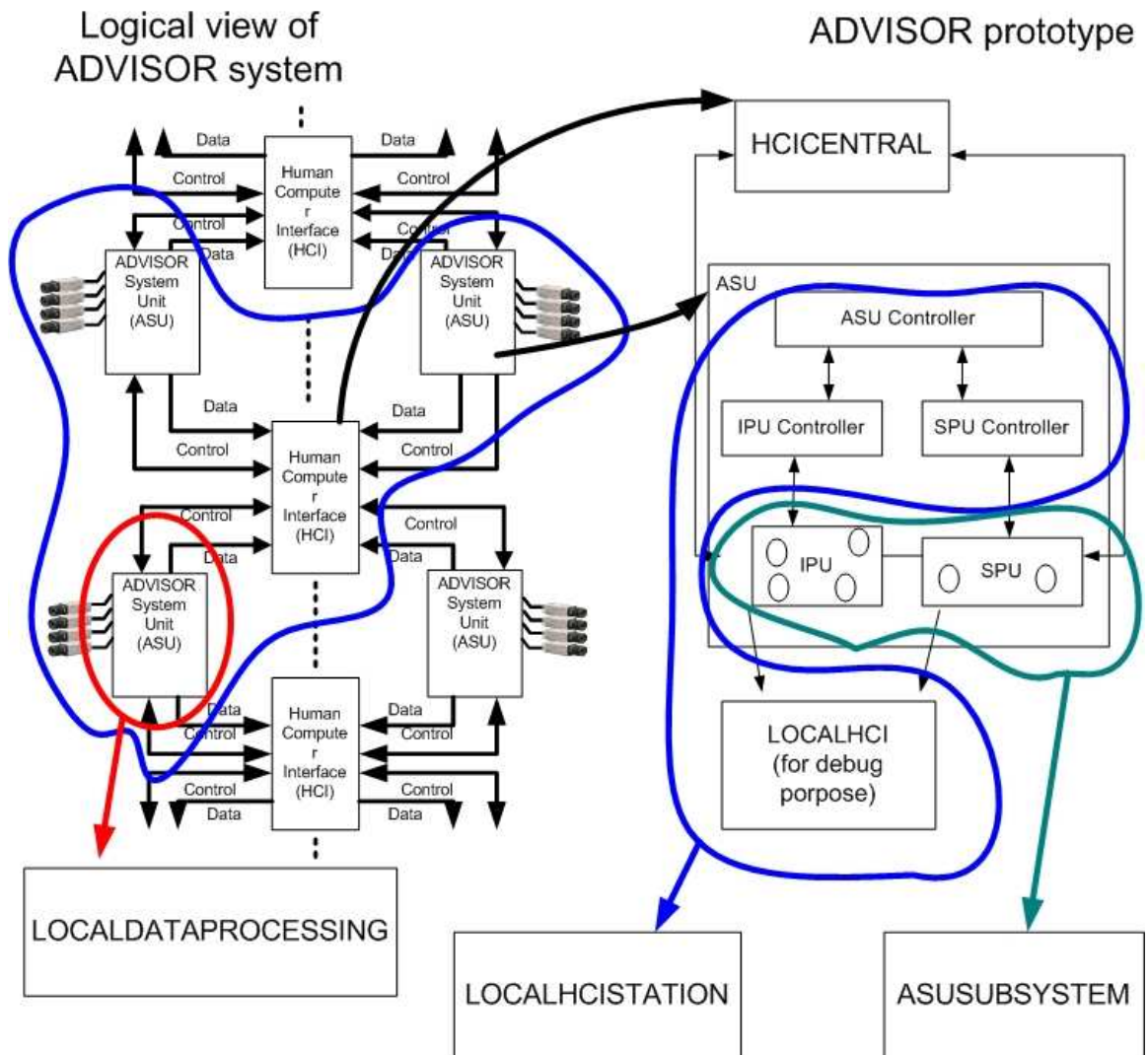


Figure 4-13. The links between the modules used below to express the architecture design of ADVISOR Prototype using MAGDE tool. Moreover, this figure illustrates the differences between the ADVISOR system and the ADVISOR Prototype (see also Figure 4-4 and Figure 4-5).

Figure 4-14 illustrates the whole system architecture design using MADGE and also shows the modules that are discussed in this section. Some of these modules illustrate how CORBA creates elements and manages them to establish transparent communications between client and server objects within the application. The arc arrows illustrated in Figure 4-14 indicate the communication between modules which belong to different levels of the architecture.

The COM and COMUHCISUBSYSTEM subsystems (see Figure 4-12 and Appendix C-13, pp. 297 and Appendix C-4, pp. 288) are designed inside

LOCALASU and LOCALDATAPROCESSING respectively. COMUHCISUBSYSTEM (see Appendix C-4, pp. 288) is a communication subsystem that appears in the design because of the mentioned tool restrictions. Therefore, the appearance of COMUHCISUBSYSTEM is unreadable in Figure 4-14. As mentioned, COM (see Figure 4-12) is a communication template subsystem, which groups all the protocols used to model the communication between the submodules that represent the client objects and the server CORBA objects.

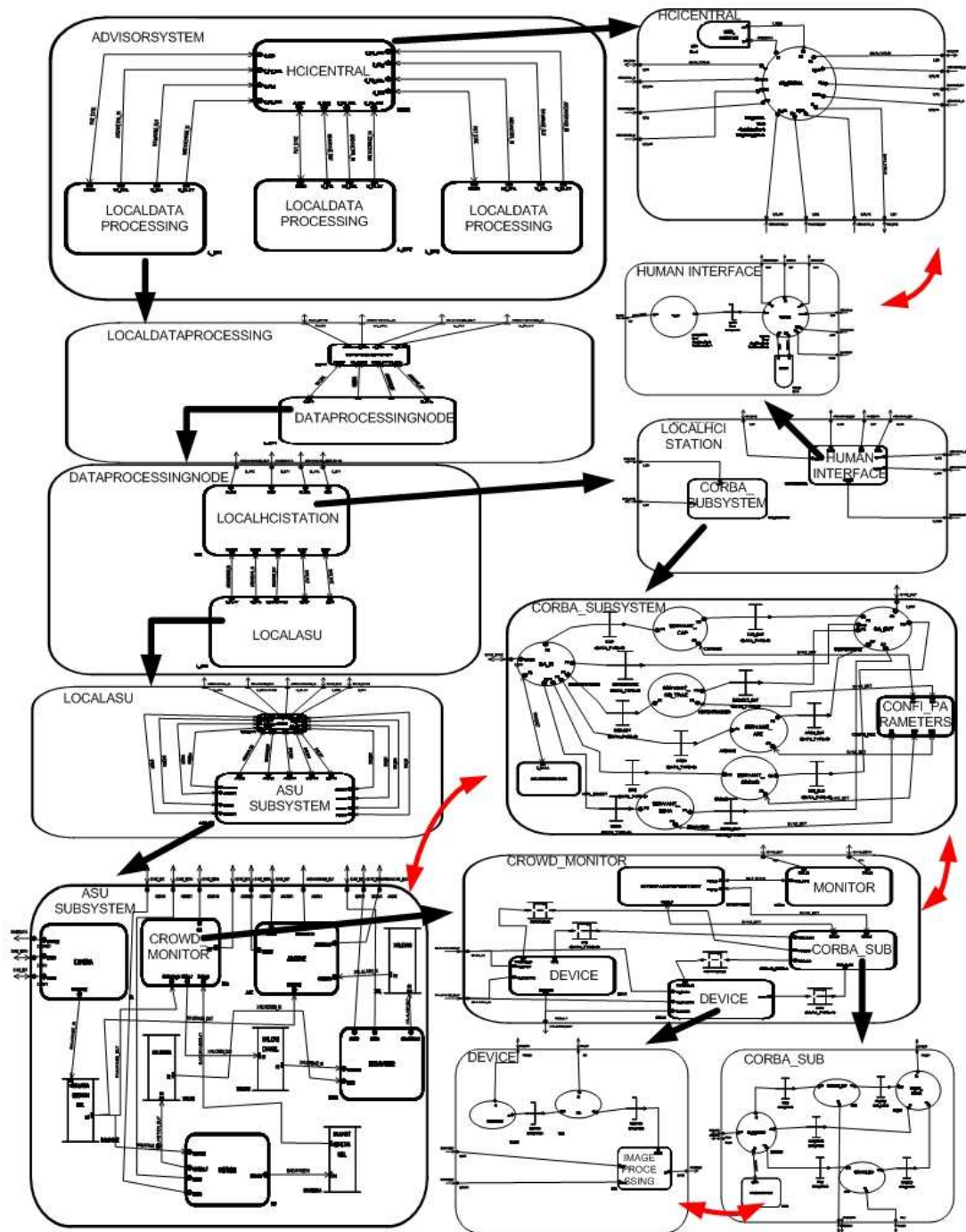


Figure 4-14. Most of the levels of decomposition of the ADVISOR Prototype system. The figure also illustrate the modules that represent the CORBA solution for ADVISOR Prototype. All the subsystems that are represented in this figure are described in more detail in the main text. All the subsystems can also be found, in clearer separate diagrams, in Appendix C. The bidirectional arrows illustrate the data communication flow between modules of different levels. Bear in mind, that some modules have been introduced due to tool restrictions even though they are not needed to model CORBA (e.g. COMUHCISUBSYSTEM in Appendix C-4, pp. 288).

The following figures illustrate the designs in a hierarchical functional manner because DORIS notation imposes a scale up/down design structure. Table 4-2 shows this structure by grouping the components in hierarchical order and also indexes each component with a level number to help the reader in the perception of this structure. The figures have been generated using MADGE that is the CASE tool that allows diagrammatic capture of RTN designs.

ADVISOR using DORIS notation			
Architeturational level structure	Name of the component	Figure Number	Number level
First level	ADVISORSYSTEM	Figure 4-15 (Appendix C-1, pp.285)	1
Second level	HCICENTRAL	Figure 4-16 (Appendix C-2, pp.286)	1,1
	LOCALDATAPROCESSING	Figure 4-17(Appendix C-3, pp.287)	1,2
Third level	COMUHCISUBSYSTEM	Refer Appendix C-4, pp.288	1.2.1
	DATAPROCESSINGNODE	Figure 4-17 (Appendix C-5, pp.286)	1.2.2
	PLAYBACKCHANNEL	Refer Appendix C-6, pp.290	1.2.1.1
	IMAGECHANNEL	Refer Appendix C-7, pp.291	1.2.1.2
	XMLCHANNEL	Refer Appendix C-8, pp.292	1.2.1.3
	DISTRIBUTRDS	Refer Appendix C-9, pp.293	1.2.1.4
	LOCALASU	Figure 4-17 (Appendix C-10, pp.294)	1.2.2.1
	LOCALHCISTATION	Figure 4-17 (Appendix C-11, pp.295)	1.2.2.2
	RDSLINK	Refer Appendix C-12, pp.296	1.2.1.4.1
	COM	Figure 4-12 (Appendix C-13, pp.297)	1.2.2.1.1
	ASUSUBSYSTEM	Figure 4-22 (Appendix C-14, pp.298)	1.2.2.1.2
	CORBA_SUBSYSTEM	Figure 4-19 (Appendix C-15, pp.299)	1.2.2.2.1
	HUMANINTERFACE	Figure 4-21 (Appendix C-16, pp. 300)	1.2.2.2.2
Fourth level	MULTIDISTRIBUTION	Refer Appendix C-17, pp. 301	1.2.2.1.1.1
	DISTRIBUTSIGNAL	Refer Appendix C-18, pp. 302	1.2.2.1.1.2
	COMSUBSYSTEM	Refer Appendix C-19, pp. 303	1.2.2.1.1.3
	CROWD_MONITOR	Figure 4-23 (Appendix C-20, pp.304)	1.2.2.1.2.1
	CONFI_PARAMETERS	Figure 4-20 (Appendix C-21, pp. 305)	1.2.2.2.1.1
	IMPLREPOSITORYSUBS	Refer Appendix C-22, pp.306	1.2.2.2.1.2
Fifth level	DISTRIBUTRDF	Refer Appendix C-23, pp. 307	1.2.2.1.1.3.1
	DISTRIBUTRTI	Refer Appendix C-24, pp. 308	1.2.2.1.1.3.2
	MONITOR	Figure 4-24 (Appendix C-25, pp.309)	1.2.2.1.2.1.1
	DEVICE	Figure 4-25 (Appendix C-26, pp. 310)	1.2.2.1.2.1.2
	CORBA_SUB	Figure 4-26 (Appendix C-27, pp.311)	1.2.2.1.2.1.3
	INTERFACEREPOSITORY	Refer Appendix C-28, pp. 312	1.2.2.1.2.1.4
	RPCLINK	Refer Appendix C-29, pp. 313	1.2.2.2.1.2.1
	RDFLINK	Refer Appendix C-30, pp. 314	1.2.2.1.1.3.1.1
	RTLINK	Refer Appendix C-31, pp. 315	1.2.2.1.1.3.2.1
	IMAGEPROCESSING	Figure 4-27 (Appendix C-32, pp. 316)	1.2.2.1.2.1.2.1

Table 4-2. The following figures in their respective level are indexed, assigning a level number to each figure to assist in following the hierarchical designs. For a clearer representation of the figures see Appendix C.

4.5.2 First level of the Architecture design: ADVISORSYSTEM

Figure 4-15 presents ADVISORSYSTEM, which is a template of the ADVISOR Prototype system. This template is composed of four instances of templates that are called HCICENTRAL and LOCALDATAPROCESSING respectively. The template is depicted by having the name in the middle of the graphical component and the instance is represented by a name written in the bottom-right corner outside the corresponding template. Accordingly, ADVISORSYSTEM consists of one instance called HCICC from a HCICENTRAL template subsystem, and three instances of the LOCALDATAPROCESSING subsystem template called L_DP1, L_DP2 and L_DP3.

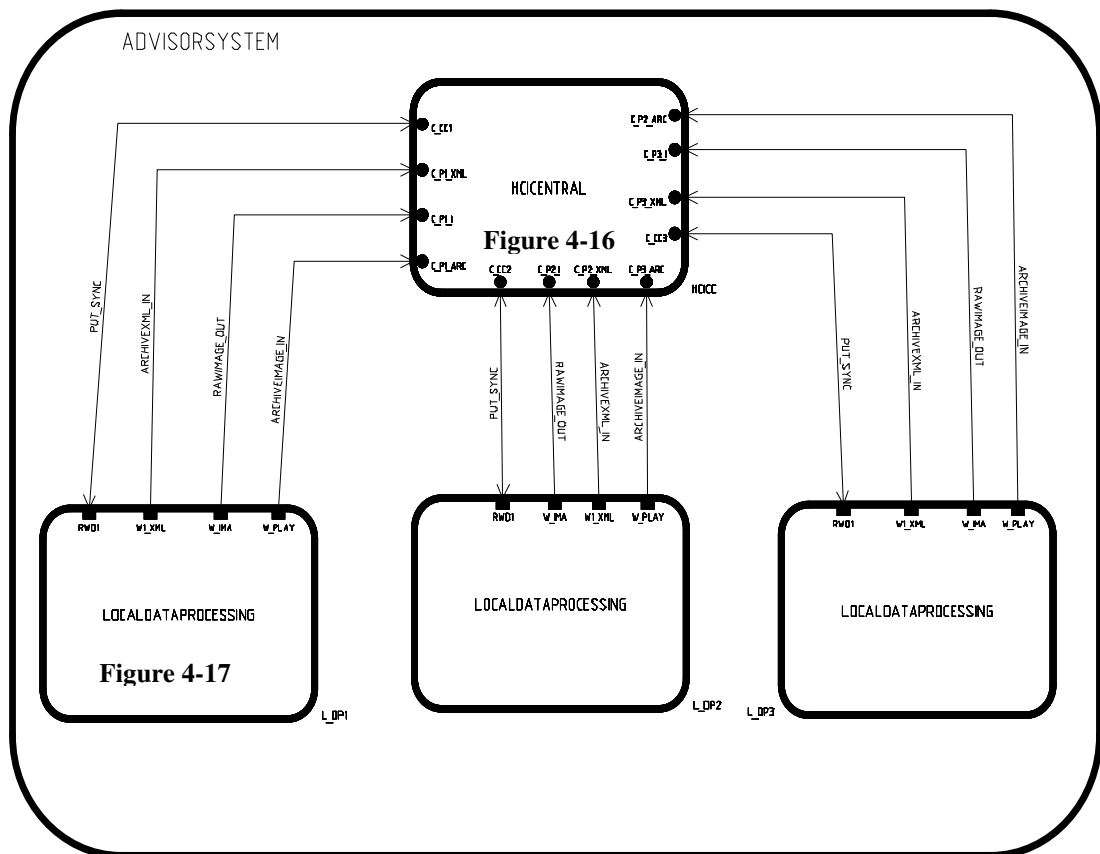


Figure 4-15. ADVISOR Prototype system using DORIS notation. This figure represents the first level of ADVISORSYSTEM. This ADVISORSYSTEM consists of three human interface subsystems and one central human interface subsystem.

In this section, MADGE has been used only as a CASE tool. MADGE imposes RTN principles in the diagram designs. Therefore, some of the components that appear in Figure 4-15 have been created to satisfy the constraints that MADGE tool

imposes. One constraint that MADGE imposes, following RTN rules, consists in not allowing connection of two active components directly. Bear in mind that active components in RTN have ports as terminators, and that passive components are characterised by having windows as terminators. Therefore, in Figure 4-15 although all the instances represented in ADVISORSYSTEM diagram are active components (nodes). The HCICENTRAL has been featured with ports. The rest of the nodes (i.e. LOCALDATAPROCESSING instances), even though are active components, have been featured with windows because as mentioned, MADGE does not allow to connect directly two active components. HCICENTRAL has been featured with ports to illustrate that HCICENTRAL has the functionality of a server when it deals with control signals (see PUT_SYNC interface in Figure 4-15). At the same time, the LOCALDATAPROCESSING has been featured with windows because acts as a client. Therefore, the HCICENTRAL provides (puts) control data to the client (i.e. LOCALDATAPROCESSING). However, HCICENTRAL may act as a client when requires (gets) non-control data from LOCALDATAPROCESSING, which then acts as a server. Following subsections discuss the functionality of these components and also present further levels of decomposition of the ADVISOR Prototype architecture design presented in section 4.5.

4.5.3 Second level of the Architecture design: HCICENTRAL

Figure 4-16 presents the HCICENTRAL subsystem. HCICENTRAL represents the central control unit in the system. It enables interactions with the user through a server component called USER_INTERFACE (see Figure 4-16). It captures control signals coming from the user and it distributes these signals to the local control units such as L_DP1 (see Figure 4-17), if required. HCICENTRAL also deals with the user interface, displaying the live images from the installations where the cameras are located or displaying recorded events.

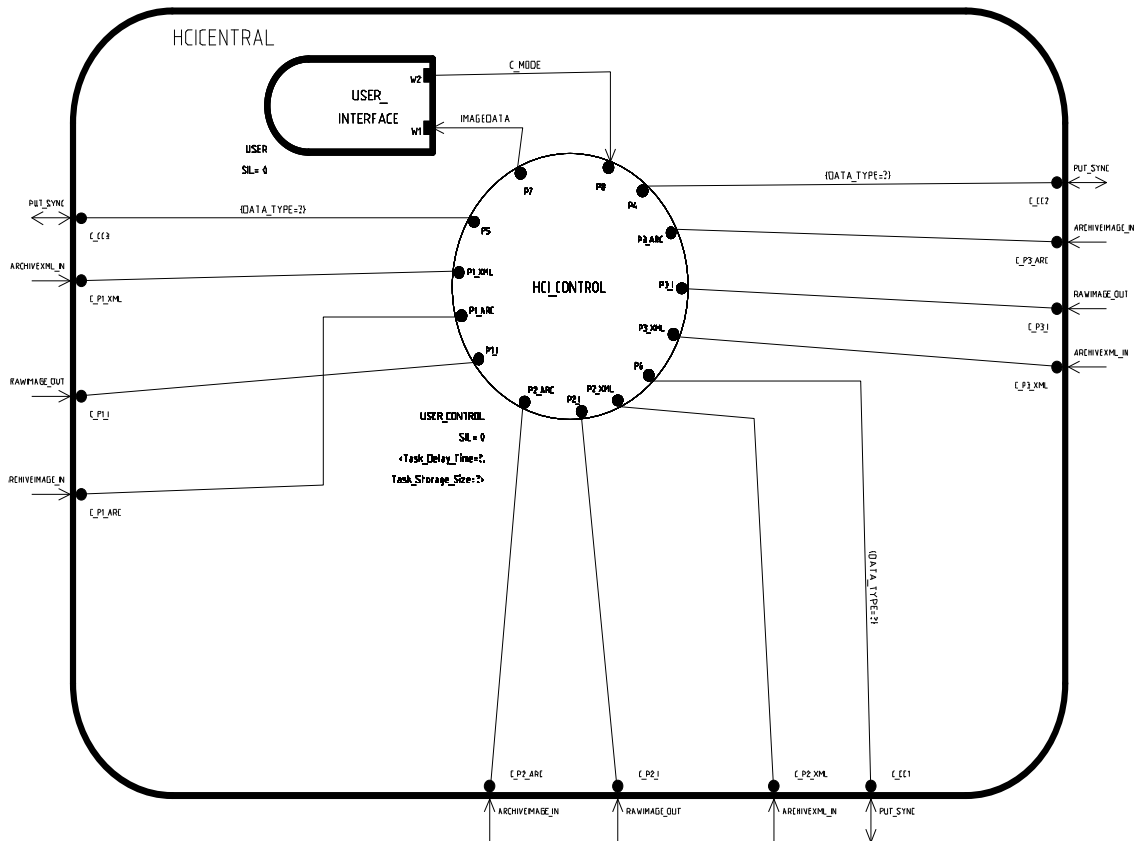


Figure 4-16. The second level of decomposition of the ADVISOR Prototype system. It represents the internal composition of the HCICENTRAL subsystem. HCICENTRAL deals with the control signals coming from the central control user.

4.5.4 Second level of the Architecture design: LOCALDATAPROCESSING

Figure 4-17 presents the decomposition of LOCALDATAPROCESSING, which as illustrated in the figure, consists of primarily one subsystem called DATAPROCESSINGNODE. The other subsystem (COMUHCISUBSYSTEM) is unreadable and smaller than DATAPROCESSINGNODE because, as mentioned, it has been added due to tool constraints (it is not possible to connect directly two active components). At the same time, COMUHCISUBSYSTEM is represented as small as it illustrates Figure 4-17, because it represents the following idea in OO and CORBA; the communication between objects (active components) is represented by a simple link (line that connects both objects). The definition of the link is not important (usually become transparently for the designer) in OO and CORBA. Therefore, COMUHCISUBSYSTEM that represents “the link” between the two active components (DATAPROCESSINGNODE and HCICENTRAL) is

shown very small. Nevertheless, by representing with DORIS notation the ADVISOR Prototype using CORBA, it is possible to discuss how this “link” may be defined. COMUHCISUBSYSTEM consists of three channel protocols: IMAGECHANNEL (see Appendix C-7, pp. 291), PLAYBACKCHANNEL (Appendix C-6, pp. 290) and XMLCHANNEL (Appendix C-8, pp. 292) that communicate HCICENTRAL with the HUMANINTERFACE subsystems.

In this case study, the designs have been created after the implementation. Therefore, it is known that the nodes communicate remotely, therefore the template substitution has been used and IMAGECHANNEL, PLAYBACKCHANNEL and XMLCHANNEL are three distributed channel protocols that stretch out the channel protocols. COMUHCISUBSYSTEM also consists of a DISTRIBUTEDRDS (see Appendix C-9, pp. 293) communication subsystem that stretches out the RDS protocol used to transmit control data between HCICENTRAL and HUMANINTERFACE (see RDSLINK element also in Appendix C-12, pp. 296).

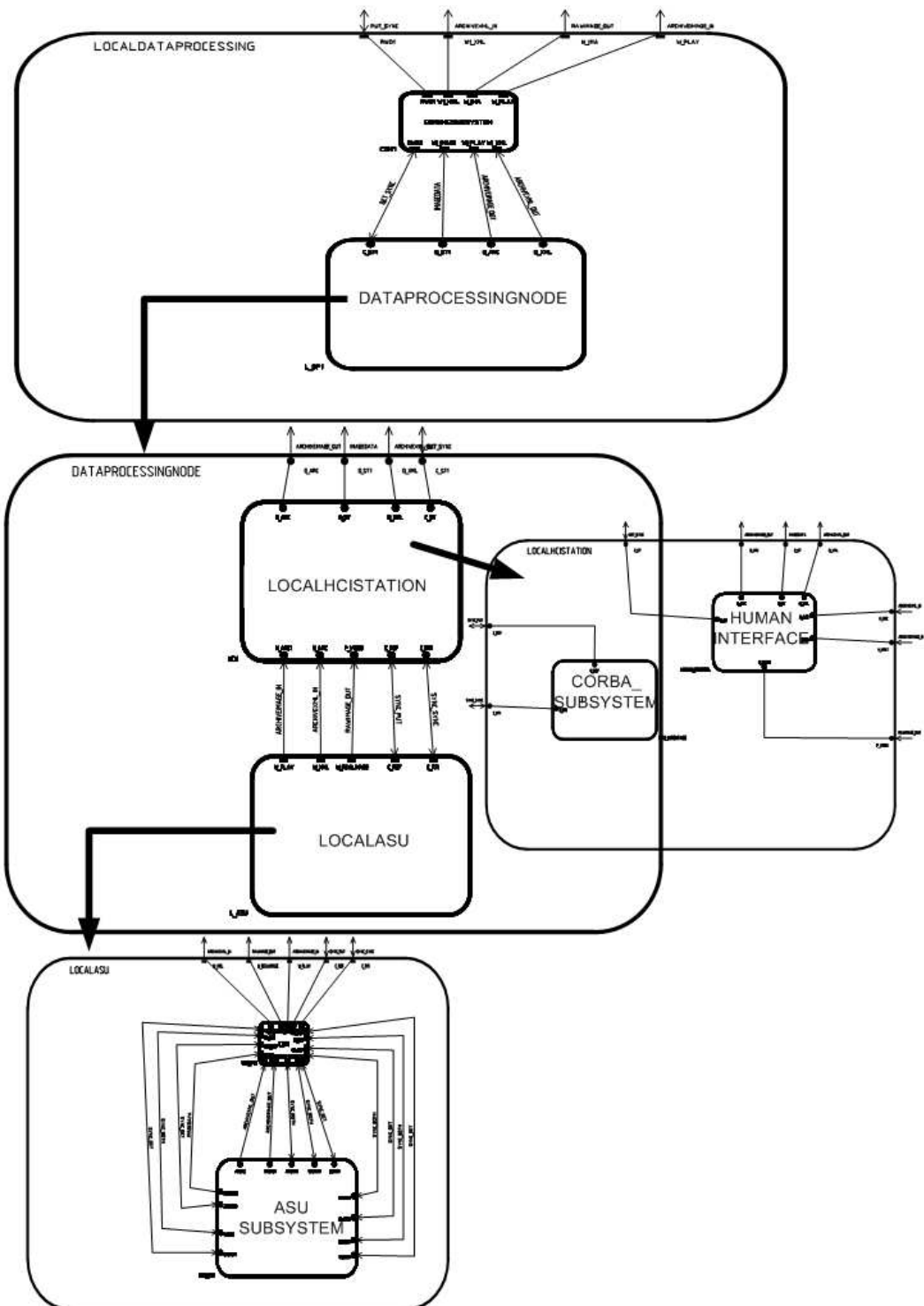


Figure 4-17. Three levels of decomposition of the ADVISOR Prototype system (following the hierarchical DORIS notation). It starts with the internal composition of LOCALDATAPROCESSING (Appendix C-3, pp. 287) subsystem, which is presented in the second level of this hierarchical structure. This is followed by the decomposition of DATAPROCESSINGNODE (Appendix C-5, pp. 289), which corresponds to the third level and it finishes with the decomposition of LOCALASU (Appendix C-10, pp.294) and LOCALHCISTATION (Appendix C-11, pp. 295) that are the third level of the hierarchical structure.

4.5.5 Third level of the Architecture design: CORBA_SUBSYSTEM, HUMANINTERFACE and ASUSUBSYSTEM

The DATAPROCESSINGNODE subsystem consists of two subsystems: LOCALASU and LOCALHCISTATION, see Figure 4-18. LOCALASU consists of two subsystems: COM (see Figure 4-11) and ASUSUBSYSTEM (see

Figure 4-22) which represents a group of processes inside LOCALDATAPROCESSING, whose functionality is to carry out the image processing tasks. As mentioned in section 4.5.4, the communication between these image processing tasks is represented by channel protocols (e.g. IMAGECHANNEL). The multicast⁹ communication is represented by connecting more than one interface to the same window). On the other hand, LOCALHCISTATION subsystem in Figure 4-18 (see also Appendix C-11, pp. 295) represents a local control unit and consists of two subsystems CORBA_SUBSYSTEM (see Figure 4-19) and HUMANINTERFACE (see Figure 4-21).

⁹ “[...]is the delivery of information to a group of destinations simultaneously using the most efficient strategy to deliver the messages over each link of the network only once and only create copies when the links to the destinations split...[.]” [Wikipedia 2001].

location of the objects known. Another strategy is called daemon¹⁰ strategy, where the ORB is implemented on a dedicated daemon process that mediates between clients and servants. The other strategy called application-resident, provides ORB as a shared library that is linked with CORBA applications. In this strategy, the ORB functionality runs in the same context as the client and the servant. In ADVISOR Prototype, the application-resident strategy has been used for implementing CORBA ORB. Therefore, the computing node called LOCALHCISTATION is represented by a group of several CORBA objects (sharing the same executing context). The designed CORBA objects are called: Capture, Motion, Crowd, Archive and Behaviour CORBA objects.

In the ADVISOR Prototype, these client objects (i.e. the software modules defined in ADVISOR system (see Appendix B, pp.280), which implement the image processing algorithms required) are located in different CPUs. These software modules are represented in

Figure 4-22 as the subsystems called CAMERA, CROWD_MONITOR, ARCHIVE, MOTION, and BEHAVIOUR respectively. The clients need to know the necessary information (in the ADVISOR Prototype this information is defined as the Internet Protocol (IP) address of each software module that is hosted in a separated PC). The reason of this needed information is because the software modules, which represent each of them an “image processing” algorithm, in ADVISOR prototype are implemented to communicate with the rest of the modules directly via sockets and therefore, the IP address is needed. The way the clients obtain this information is through the Naming Service and it is explained in next paragraph.

The clients request to CORBA objects their IP addresses through CORBA methods. The CORBA objects are presented in Figure 4-19. To initiate the request to the correct CORBA object incarnated as servant object, any of these clients needs to know the reference of the specific servant object. This reference is provided by Naming Service in the ADVISOR Prototype CORBA implementation. As mentioned in section 4.2, the functionality of the ORB is defined through its

¹⁰ A daemon is a standalone operating system process that runs in the background and provides some services

services. Therefore, we represent the service used in ADVISOR Prototype to illustrate the functionality of the ORB. The implementation of this service is shown in Figure 4-19 and it works in the following way. The OA_IN template activity receives a request, i.e. the reference of a servant, from one of the clients. The OA_IN searches this reference in IMPLREPOSITORYSUBS (see Appendix C-22, pp. 306). Once OA_IN obtains the reference, it sends the reference through the NAMESERVICE instance channel to OA_OUT. The OA_OUT sends back the reference to the client that previously asked for the reference in ASUSUBSYSTEM. Once the client has the reference, it is able to call the CORBA method to the specific servant. OA_IN also routes the petitions to the servants. Then, when the client requests an IP address from a servant, the servant obtains the requested IP address from CONFIG_FILE instance (inside the CONFI_ PARAMETERS subsystem, see Figure 4-20). The servant sends the obtained IP address back to the OA_OUT. Then, the OA_OUT activity sends this information to the corresponding client in the ASUSUBSYSTEM through the COM subsystem.

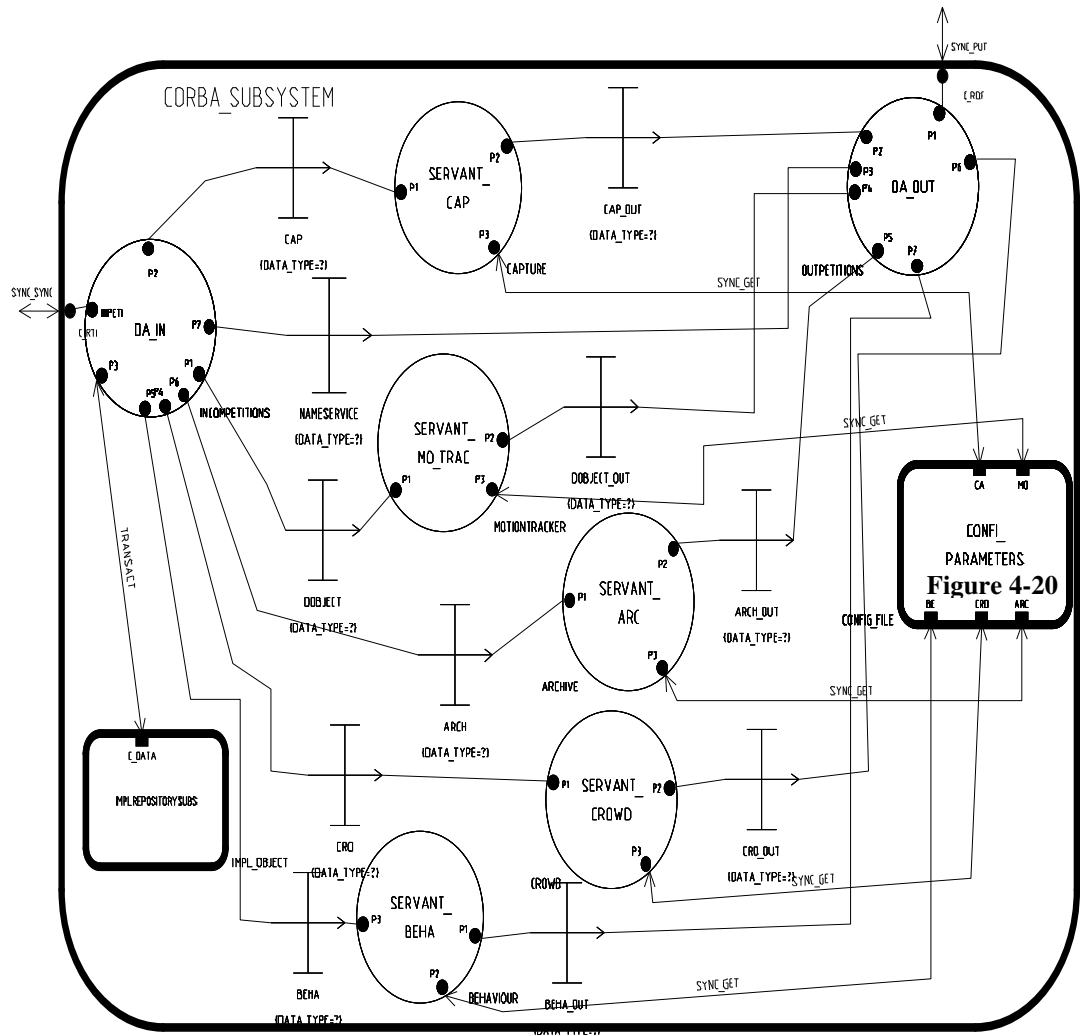


Figure 4-19. Following the hierarchical MASCOT/DORIS notation, CORBA_SUBSYSTEM represents the fifth level of the ADVISOR Prototype system. CORBA_SUBSYSTEM subsystem illustrates the communication functionality of an ORB in CORBA technology.

4.5.5.1.1 Fourth level of the Architecture design: CONF_PARAMETERS

The CONF_PARAMETERS subsystem represents a datarepository, which servants from CORBA_SUBSYSTEM access to get the information that the clients require. In CORBA_SUBSYSTEM there is another subsystem that has a similar functionality called IMPLREPOSITORYSUBS (see Appendix C-22, pp.306). IMPLREPOSITORYSUBS as introduced in section 4.4.1, is also a datarepository that stores the references or IDs of the different servants that are used in this CORBA implementation, allowing then, the client to communicate with a servant without having prior knowledge of the ID of the servant.

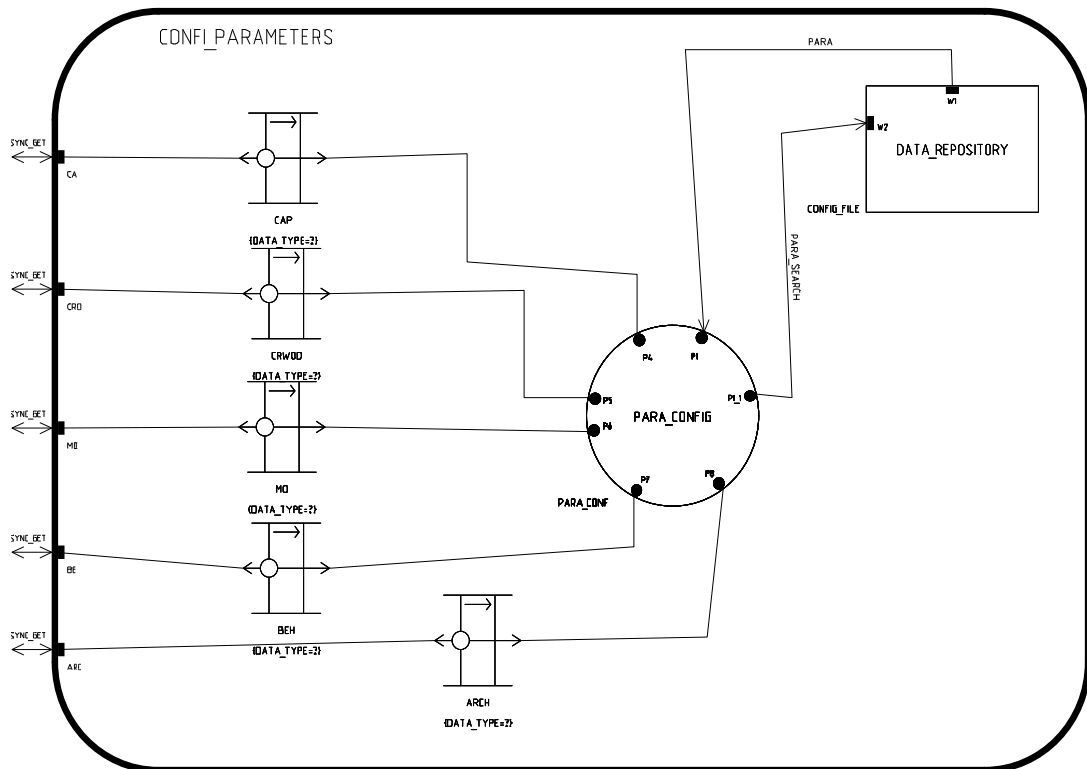


Figure 4-20. CONF_PARAMETERS (the sixth level of the ADVISOR Prototype decomposition) is introduced in the fifth level of the hierarchical structure in CORBA_SUBSYSTEM. CONF_PARAMETERS gives the configuration parameters required for other components outside the subsystem.

4.5.5.2 Third level of the Architecture design: HUMANINTERFACE

HUMANINTERFACE represents the control unit process of the HCISTATION, which interacts with the user through the server component called SCREEN (see Figure 4-21). HUMANINTERFACE also makes possible the communication between ASUSUBSYSTEM and HCICENTRAL. HUMANINTERFACE deals with the control signals that are coming from the HCICENTRAL subsystem and with the image data and XML results that are coming from the ASUSUBSYSTEM. Depending on the control signals coming from HCICENTRAL, the HUMANINTERFACE subsystem sends back to HCICENTRAL live images from the CCTV, or archive images, or events that have been archived or events that just have occurred. HUMANINTERFACE also does the same with the control signals coming from the user through the SCREEN server component.

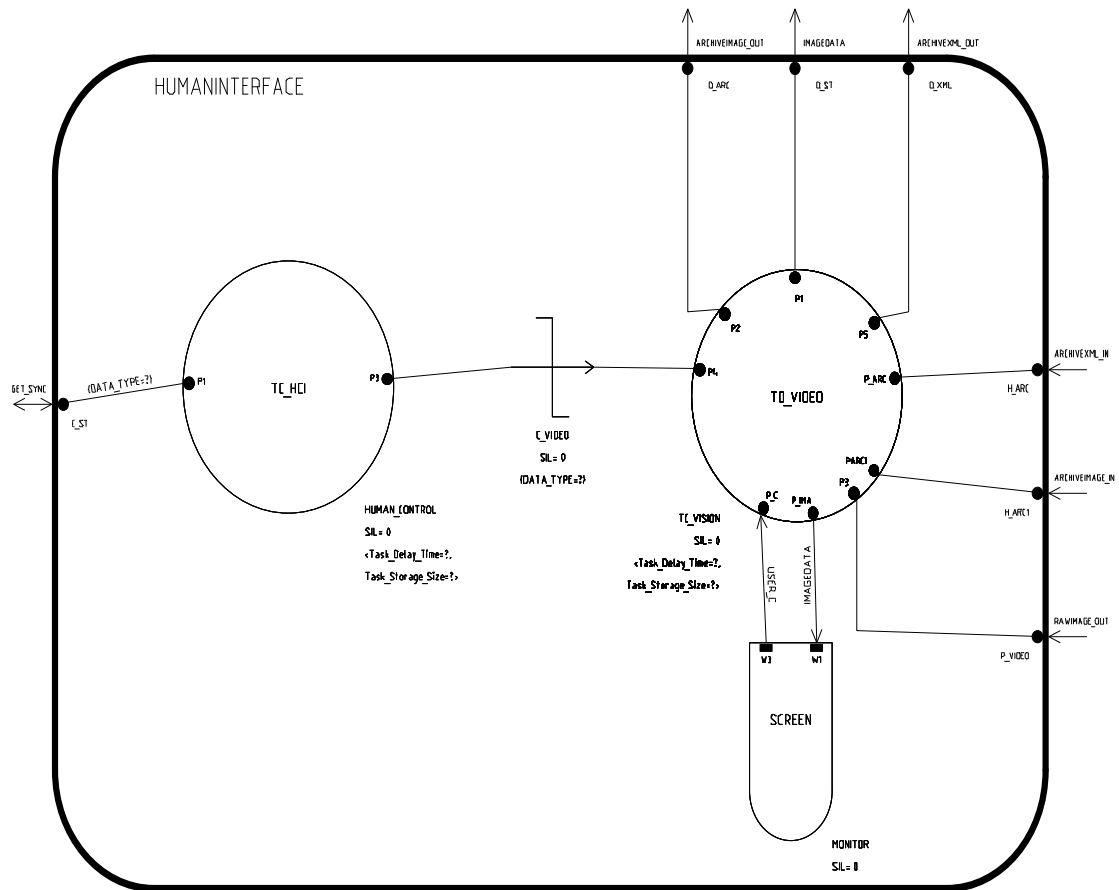


Figure 4-21. HUMANINTERFACE subsystem presented in the third level of the hierarchical structure of the ADVISOR Prototype system. HUMANINTERFACE is introduced in LOCALHCISTATION subsystem. HUMANINTERFACE deals with the control signals coming from the HCICENTRAL and it also deals with local control signals coming from the user in the local HCI subsystem.

4.5.5.3 Third level of the Architecture design: ASUSUBSYSTEM

The ASUSUBSYSTEM module accomplishes the required image processing tasks as mentioned. ASUSUBSYSTEM consists of five active subsystems: CAMERA, CROWD_MONITOR, ARCHIVE, MOTION and BEHAVIOUR. Accordingly, the final decomposition of ASUSUBSYSTEM may be seen as a network of activities that perform image processing algorithms, which are described in the Appendix B, pp. 280. In this section, only the decomposition on further levels of CROWD_MONITOR module is presented (see Figure 4-23, Figure 4-24, Figure 4-25 and Figure 4-26).

The CAMERA subsystem inside the ASUSUBSYSTEM (see

Figure 4-22. ASUSUBSYSTEM is presented in the third level of the decomposition of ADVISOR Prototype system. ASUSUBSYSTEM is composed of image processing subsystems.

4.5.5.3.1 Fourth level of the Architecture design: CROWD_MONITOR

The CROWD_MONITOR subsystem is also designed using CORBA. See Figure 4-23. CROWD_MONITOR handles two DSPs (Digital Signal Processing) that perform a low level processing optical flow algorithms to the obtained images from CAMERA subsystem. It applies afterwards, some algorithms to the outputs from the DSPs and sends the results in XML format to the BEHAVIOUR subsystem. CROWD_MONITOR is composed of five subsystems; MONITOR (see Figure 4-24), two DEVICES (see Figure 4-25), INTERFACEREPOSITORY (see Appendix C-28, pp.312) and CORBA_SUB (see Figure 4-26).

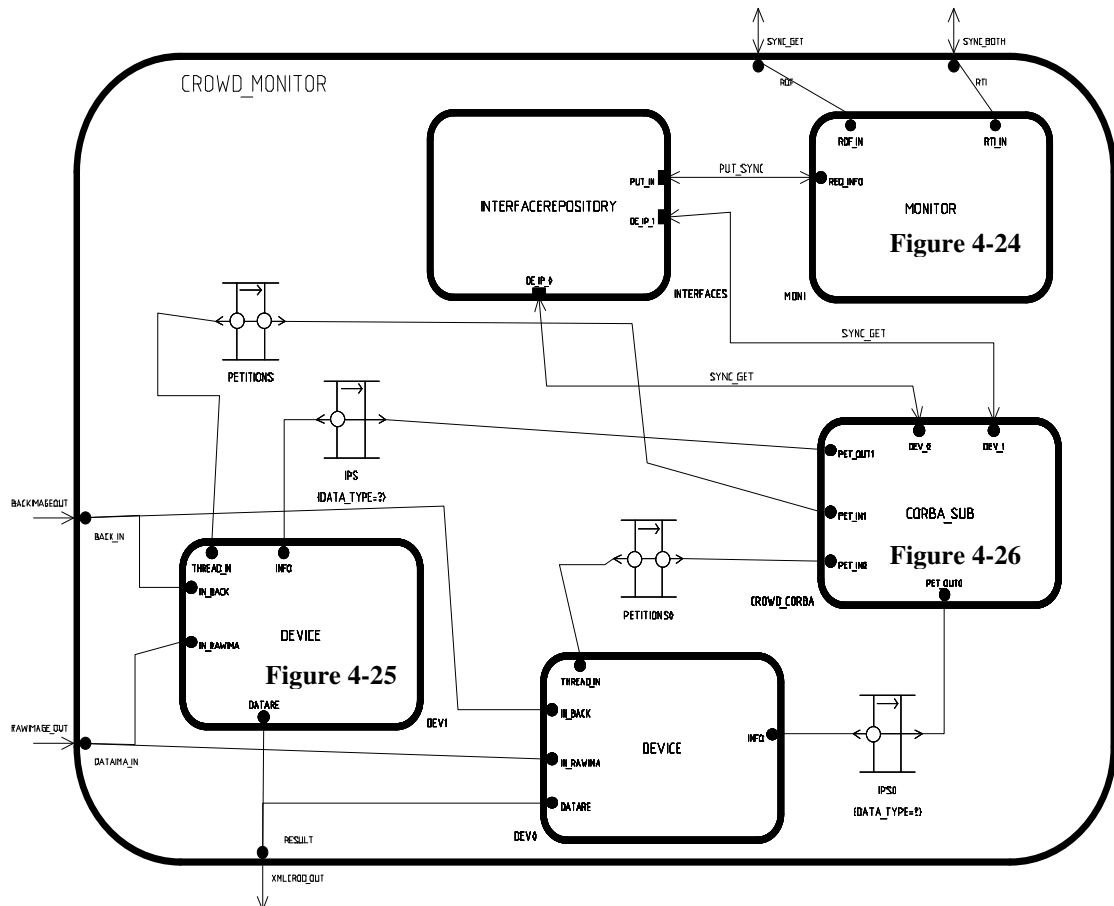


Figure 4-23. The CROWD_MONITOR subsystem (the fourth level of ADVISOR Prototype system). It performs an image processing task detecting crowds.

4.5.5.3.1.Fifth level of the Architecture design: MONITOR

The MONITOR subsystem gets the information that has requested from the LOCALHCISTATION (from CORBA_SUBSYSTEM) and puts this obtained

information to the INTERFACE-REPOSITORY in CROWD_MONITOR, see Figure 4-24.

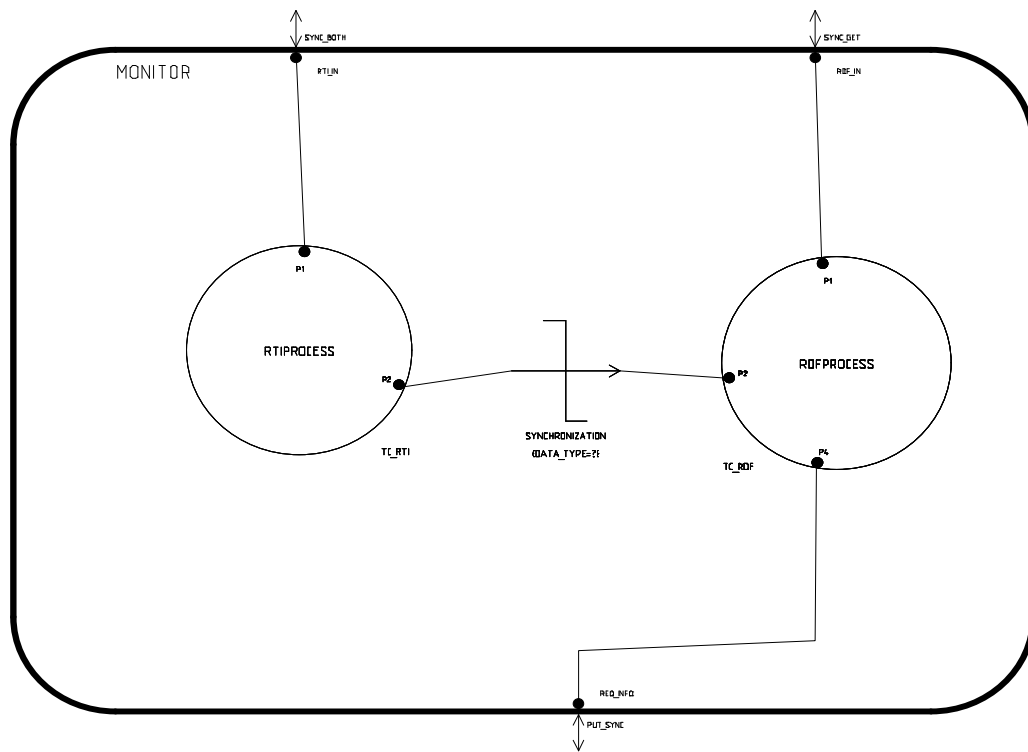


Figure 4-24. The MONITOR subsystem (the fifth level of ADVISOR Prototype system) deals with control signal coming from upper levels and petitions from lower levels.

4.5.5.3.1.2 Fifth level of the Architecture design: DEVICE

DEVICE, which acts as a client object, is composed of two activities and the IMAGEPROCESSING subsystem (see Figure 4-27). The INTERFACES activity activates the corresponding servant and the PA activity obtains the IP address from the activate servant (see Figure 4-25).

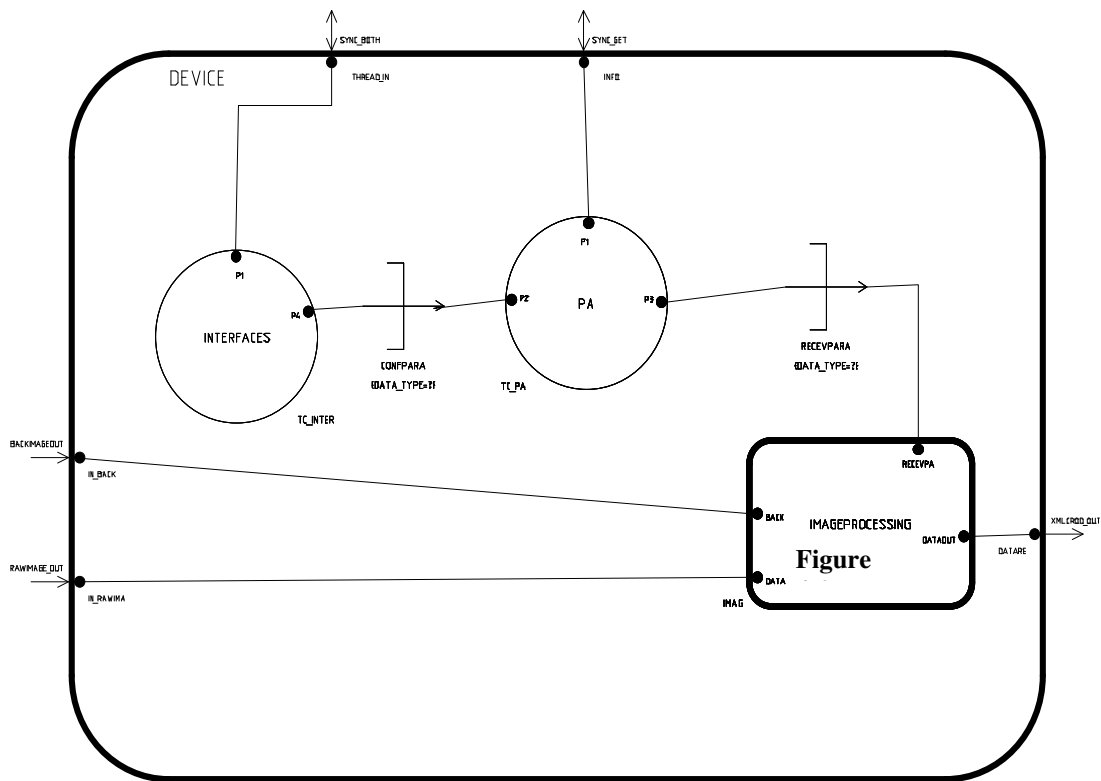


Figure 4-25. DEVICE (fifth level of the decomposition of the ADVISOR Prototype system) is a subsystem that deals with the control signals coming from upper levels. It also deals with the signals of a subsystem where the low-level image processing tasks are carried out.

4.5.5.3.1.3 Fifth level of the Architecture design: CORBA_SUB

CORBA_SUB like CORBA_SUBSYSTEM represents the CORBA ORB design used in the ADVISOR Prototype. It has two CORBA objects that deal with the requests coming from the DEVICE subsystems. As mentioned, MONITOR in CROWD_MONITOR (see Figure 4-23) acts as a client object, requesting the IP information from LOCALHCISTATION (see Figure 4-17). Once MONITOR receives the information, it sends it to INTERFACEREPOSITORY, which stores the IP address information obtained previously by MONITOR subsystem. The SERVANT_DEV in CORBA_SUB then sends the IP address to the DEVICE client, every time that the DEVs require it.

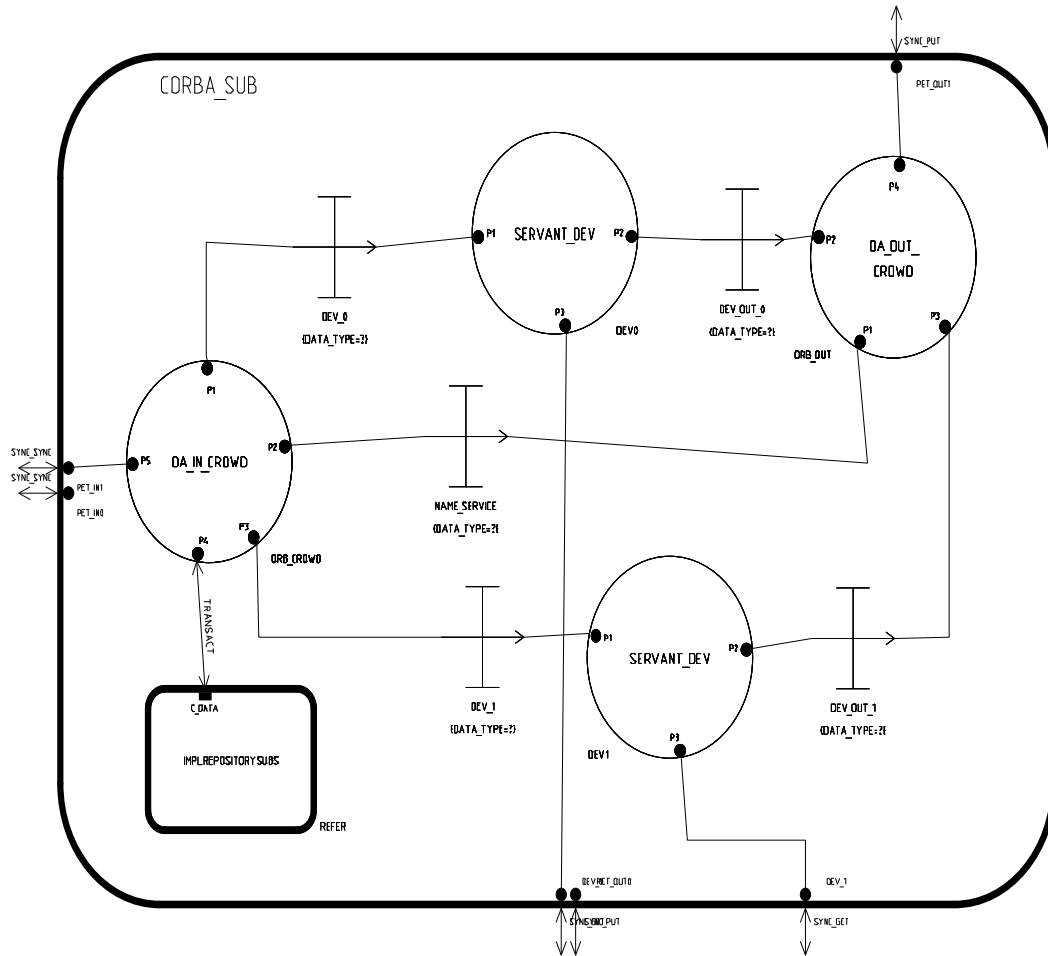


Figure 4-26. CORBA_SUB subsystem (fifth level of decomposition of the ADVISOR Prototype system) illustrates the communication functionality of an ORB in CORBA technology like in CORBA_SUBSYSTEM.

4.5.5.3.1.4Fifth level of the Architecture design: IMAGEPROCESSING

IMAGEPROCESSING (see Figure 4-27) interacts with the DSPs through the server components called DSP, providing the raw images to the DSPs and obtaining the results. After applying defined thresholds to the obtained results from the DSP, IMAGEPROCESSING sends the XML results to the BEHAVIOUR module.

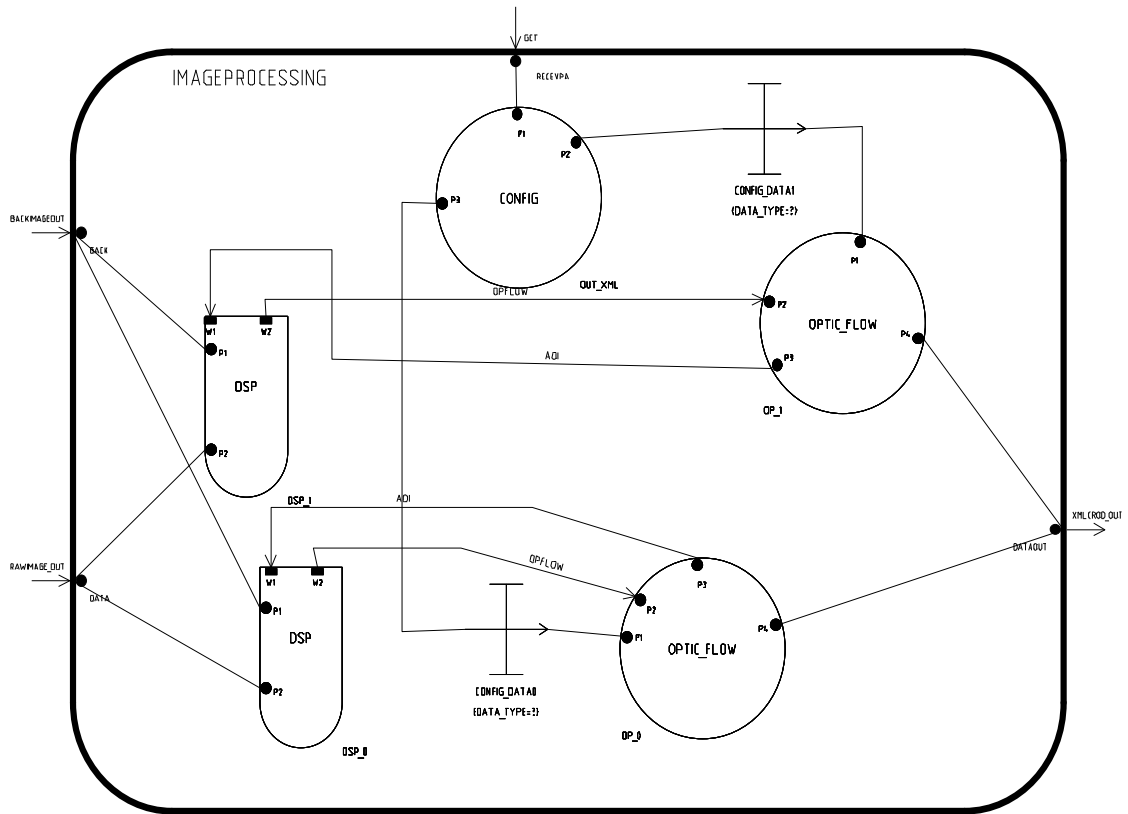


Figure 4-27. The IMAGEPROCESSING (the fifth level of decomposition the ADVISOR Prototype system) subsystem, which performs the image processing tasks.

4.6 The ADVISOR Prototype architecture using the DORIS method and its concepts

This section presents the ADVISOR Prototype system design, with the mentioned variations, applying RTN concepts using MASCOT-3 and DORIS extensions, to continue the comparison between CORBA and MASCOT-3/DORIS approaches, established in Table 4-1. In Table 4-3, the hierarchical structures of the ADVISOR designs are presented, indexing each design component with its corresponding figure.

ADVISOR using RTN concepts			
Architectural level structure	Name of the component	Figure Number	Number level
First level	ADVISORSYSTEM	Figure 4-28 (Appendix C-33, pp. 317)	
Second level	COMMUNICATION	Figure 4-29 (Appendix C-35, pp. 319)	1,
	HCICENTRAL	Figure 4-30 (Appendix C-34, pp. 318)	1,
	HCINODE	Figure 4-31 (Appendix C-36, pp. 320)	1,
Third level	CROWD_MONITOR	Figure 4-32 (Appendix C-37, pp. 321)	1.3.1

Table 4-3. The following figures in their respective level are indexed, assigning a number level to clarify the hierarchical designs.

4.6.1 First level of the architecture design: ADVISORSYSTEM

Figure 4-28 presents the design of the ADVISOR system, which consists of a communication element called COMMUNICATION (see Figure 4-29) and four computing nodes: one HCICENTRAL subsystem (see Figure 4-30) and three HCINODE subsystems (see Figure 4-31). The HCICENTRAL subsystem communicates with the rest of the modules through the composite IDA called COMMUNICATION. A single communication element is used to communicate HCICENTRAL with the rest of HCINODEs because all identical interfaces in each HCINODE subsystem are connected to the same window; e.g. the RAWIMAGEOUT interface in each HCINODE subsystem is connected to the same window called CA_IN in the COMMUNICATION element. MADGE allows the connection of a number of interfaces to the same window if the interfaces are the same type. The IDA, then, will provide a mechanism to deal with each interface separately (i.e. multicast).

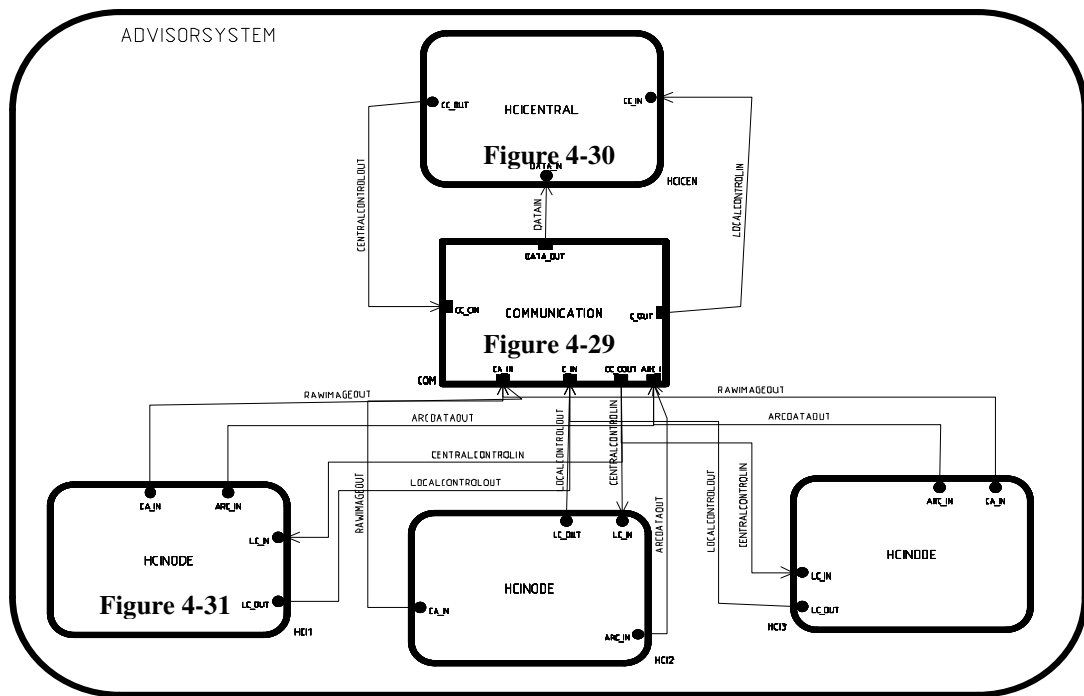


Figure 4-28. This figure represents the ADVISOR system using RTN concepts.

ADVISORSUBSYSTEM (Appendix C-33, pp. 317) represents the first level of the design system. The system is composed of four computing nodes and one communication element.

The following subsections discuss the functionality of the components illustrated in Figure 4-28 and also present further levels of decomposition of the ADVISOR architecture design presented in section 4.6.

4.6.2 Second level of the architecture design: COMMUNICATION

Figure 4-29 represents the decomposition of the composite IDA element COMMUNICATION. This element has two data inputs coming from the HCINODE that are transmitted to HCICENTRAL using a distributed form of a channel, which consists of a simple channel connected to an active agent (e.g. the TRANSCANAL activity in Figure 4-29) that stretches out the data to another channel. In this section, the idea of template substitution has also been used because it is known from the implementation that the communication between the HCICENTRAL node and the HCINODEs is remote. In this case, TRANSCANAL also multiplexes the data coming from two different interfaces (RAWIMAGEOUT and ARCDATAOUT) to a unique interface called DATAIN. On the other hand, the TRANSGN and TRANSGN_IN activities only stretch out the route of control data

coming from the HCICENTRAL to the HCINODE and the returned signals coming from a HCINODE to HCICENTRAL.

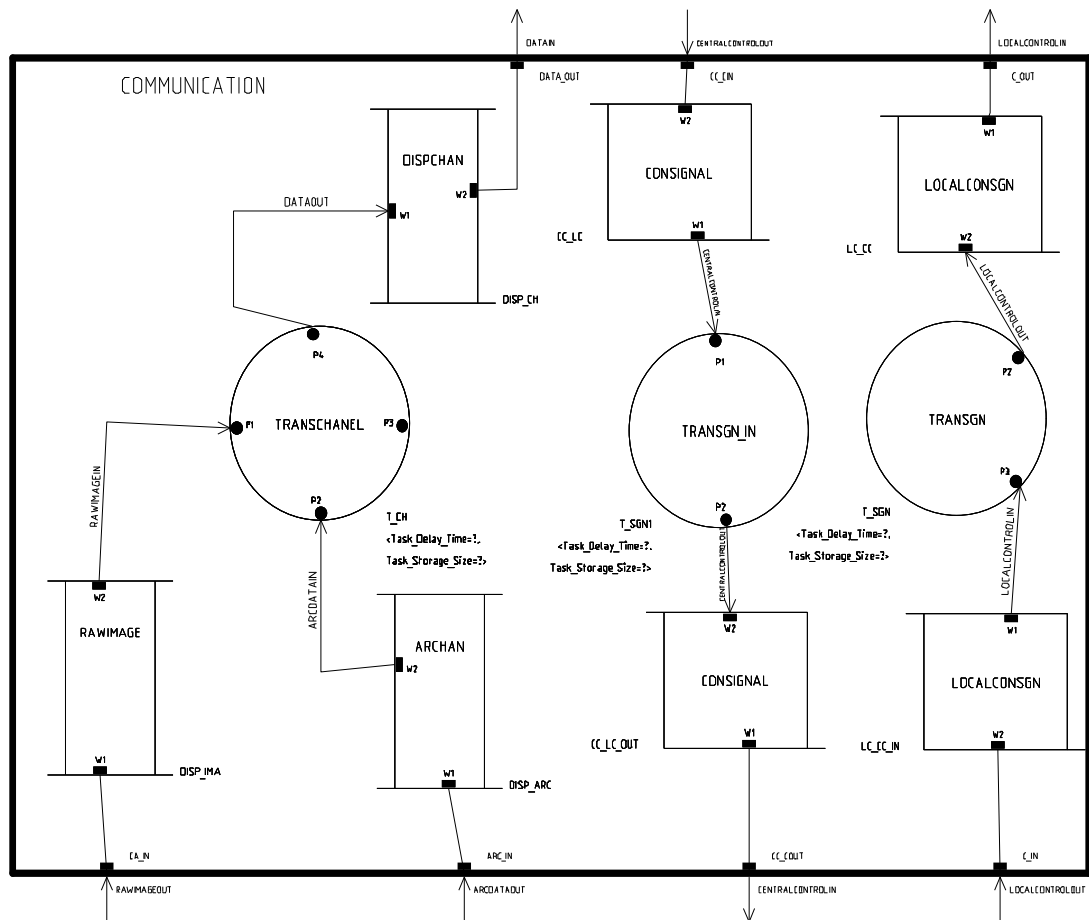


Figure 4-29. COMMUNICATION (Appendix C-35, pp. 319) composite IDA (second level of decomposition of the ADVISOR Prototype system), is shown in the first level of the hierarchical structure of ADVISOR. This communication element links the HCICENTRAL subsystem with each HCINODE.

4.6.3 Second level of the architecture design: HCICENTRAL

Figure 4-30 presents the internal composition of HCICENTRAL subsystem. This subsystem consists of two activities: the TC_HCI activity and the DISPLAY activity. TC_HCI merely deals with control signals coming from any HCINODE or from the server element called SCREEN, which allows the subsystem HCICENTRAL to interact with the environment (in this case, it allows the interaction of the HCICENTRAL subsystem with a user). TC_HCI may also interact with the DISPLAY activity through a signal protocol to manage, if needed, what should be displayed on the screen. Therefore, the DISPLAY activity sends to

the SCREEN server, if required, the received data from any HCINODE, to be displayed in the console.

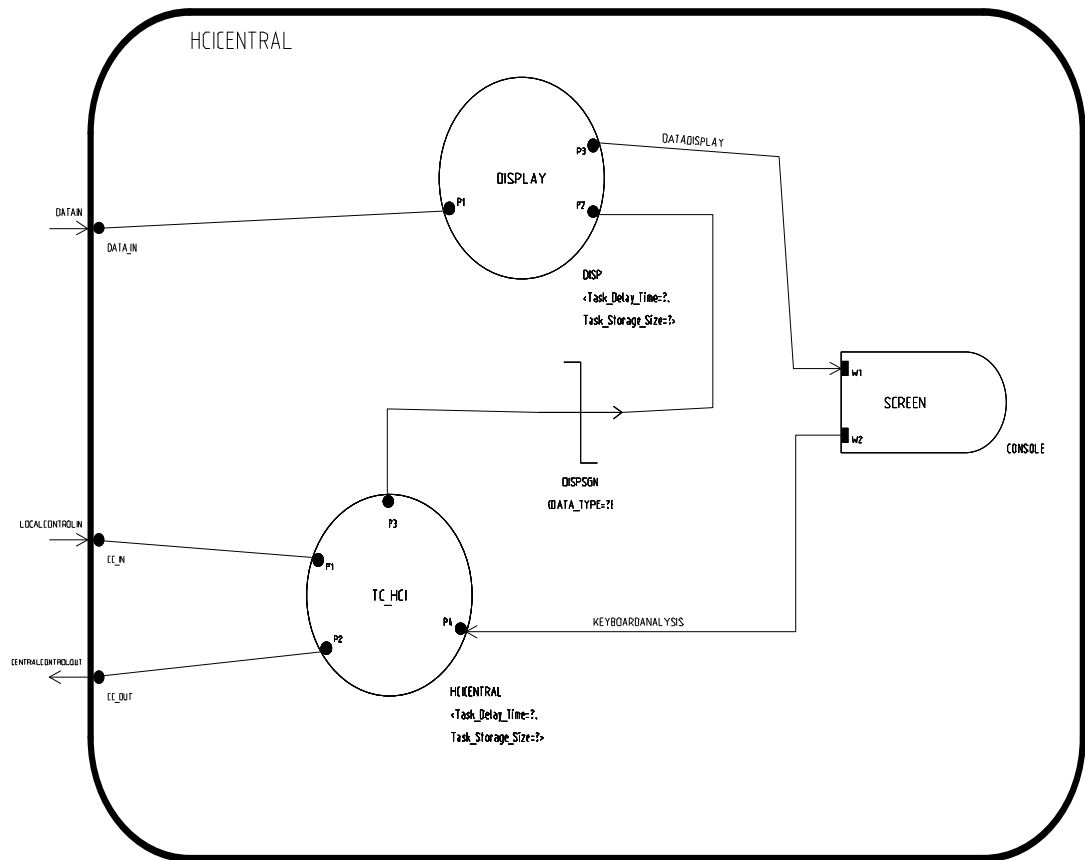


Figure 4-30. The HCICENTRAL (Appendix C-34, pp. 318) subsystem purely displays, if a user requires, data coming from any HCINODE. It also deals with control signals coming from any HCINODE.

4.6.4 Second level of the architecture design: HCINODE

Figure 4-31 represents the image processing node. Following the structure of ADVISOR, HCINODE is composed merely of five image processing tasks: capture and digitalisation, motion detection and background subtraction, crowd motion detection, behaviour analysis and finally archiving. Capture and digitalisation of the CCTV images operations are carried out by the CAMERA subsystem. Motion detection and background subtraction operations are performed in the MOTION subsystem. Another image processing task consisting in the detection of crowd situations is carried out in the CROWD_MONITOR subsystem (see Figure 4-32). The creation of natural language messages with the results coming from the CROWD_MONITOR and MOTION subsystems is performed in the BEHAVIOUR

subsystem. The ARCHIVE subsystem merely stores the results coming from the CAMERA and CROWD_MONITOR subsystems. HCINODE also consists of a control subsystem called HCILOCAL, whose internal composition is merely the same as the HCICENTRAL subsystem. This subsystem deals with control signals; it distributes to the different subsystems the control signals coming from the HCICENTRAL subsystem, and it also may send control signals coming from the local user to the HCICENTRAL subsystem. HCILOCAL also displays data coming from the CAPTURE, BEHAVIOUR and ARCHIVE subsystems to the local console.

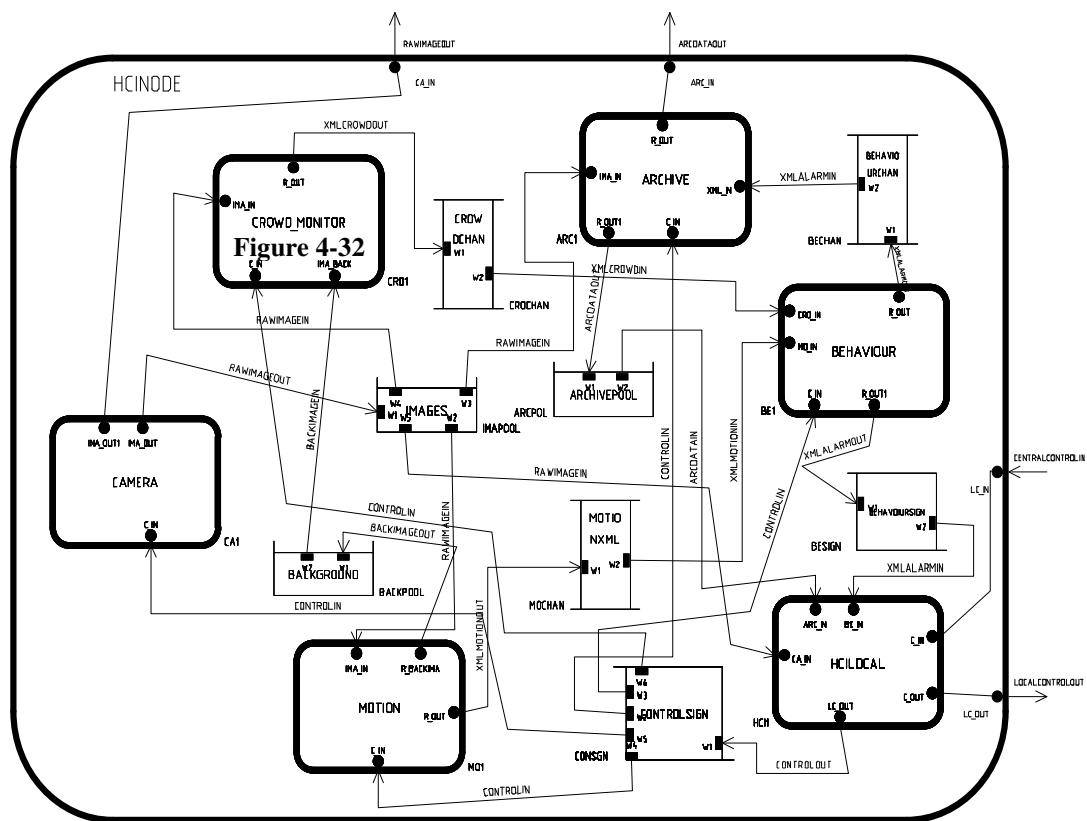


Figure 4-31. The HCINODE (Appendix C-36, pp. 320) is the second level of the hierarchical structure of the ADVISOR Prototype system. It is composed of six subsystems, which communicate between them through IDAs: channels, pools and signals.

4.6.5 Third level of the architecture design: CROWD_MONITOR

In this section, as in the previous section 4.5, only the decomposition of CROWD_MONITOR is presented (see Figure 4-32). Figure 4-32 presents CROWD_MONITOR subsystem, which consists of two activities and two server components called DSP1 and DSP0 that interact with a dedicated hardware.

CONTROL_CONFIG activity receives control signals coming from HCILOCAL subsystem such as changing thresholds parameters or changing AOIs and sends them back to OPTIC_FLOW activity or DSP servers, depending on the type of control signal to be sent. CONTROL_CONFIG activity also receives the digitalised images and the background images from CAPTURE and MOTION subsystem respectively and sends them to each Digital Signal Processing (DSP) dedicated hardware, where specific image processing algorithms are applied to these images. Each DSP server sends the results to the OPTIC_FLOW activity, which after performing specific operations, sends the final results to the BEHAVIOUR subsystem.

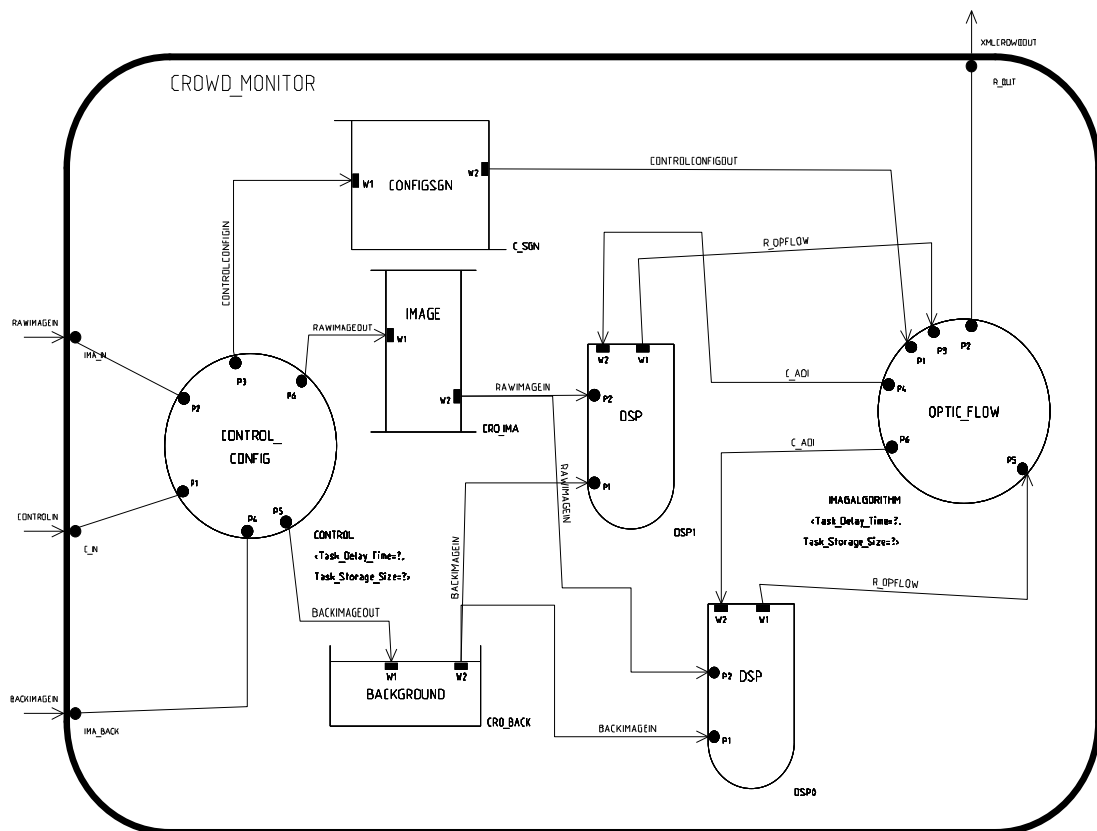


Figure 4-32. The CROWD_MONITOR (Appendix C-37, pp. 321) represents the third level of the hierarchical structure of the ADVISOR Prototype system. It is composed of two activities and two servers which communicate between them through a channel, a pool and a signal.

4.7 Comparison between the two architectures

In section 4.5 and 4.6 the designs of the ADVISOR system, expressing different solutions have been presented. In this section, the discussion of design differences between both approaches is focused on the functional partition of the system and on

the divergence of architecture designs. Therefore, the following figures present graphical dissimilarities at some design levels of ADVISOR architecture solution using the two approaches.

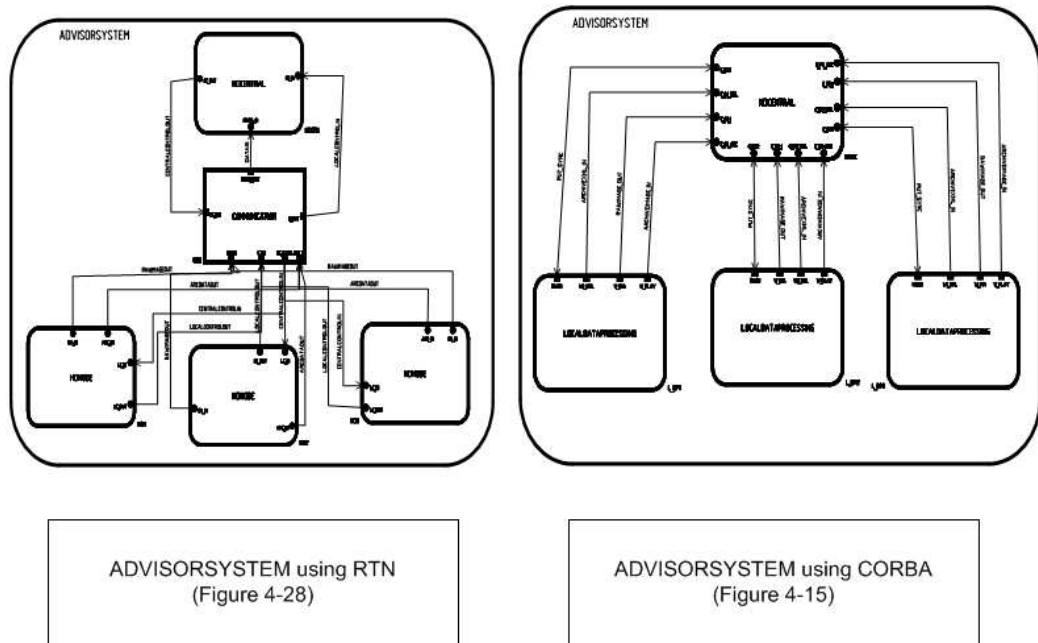


Figure 4-33. Comparison at a first level of design of ADVISOR Prototype system using the two approaches.

The functionality at the first level of the architecture in both approaches is the same because they are coming from the same specifications and, in this case study, from the same implemented system (see Figure 4-33); i.e. the ADVISOR system design presented in this chapter, is subdivided primarily into four active nodes (see Figure 4-15 and Figure 4-28). One of these active nodes represents the primary control node, i.e. the HCICENTRAL (see Figure 4-16 and Figure 4-30), which is the node that interacts with the user of the system and with the rest of the data processing nodes. The functionality of the rest of the nodes is divided into local control functions and data (e.g. images) processing functions. In this section, nodes that have control functionality are called central or local control nodes, and nodes that have data processing functionality are called data processing nodes.

4.7.1 Communication techniques

Even though the functionality at the first level of the architecture design is the same in both approaches, the architecture design in the following levels is quite different,

mainly because of the communication techniques. In RTN, communication is symmetric (both components that communicate have the same roles). It uses an extra explicit component (IDA) to allow this independent communication. RTN solution presents independence between pairs of communicating nodes (i.e. data processing nodes and control node). This is represented in the design by using only one IDA for the communication of HCICENTRAL (the control node) and the rest of the HCINODE (data processing) nodes. In CORBA, communication is asymmetric (the components that are communicating have different roles i.e. usually one is a client and the other is a server). In the presented CORBA architecture design, there is a certain dependency between the control component and the client components, which is expressed in the design by drawing each time the communication links between each HCICENTRAL-LOCALDATA-PROCESSING pair of nodes (see Figure 4-33).

Therefore, in the ADVISOR Prototype architecture design using CORBA, there is an explicit coupling of control signals between data processing nodes and control nodes; the data processing nodes need some setup data from the local control nodes in order to work. Therefore, the data processing nodes require these data at the start-up time, e.g. to be able to communicate with the other modules. In contrast, in the ADVISOR Prototype architecture design using RTN concepts, the data processing nodes do not have this control coupling with the control nodes because the activities in RTN work independently from each other. The activities are only aware of sending and receiving information from their ports. Therefore, even though they can receive control signals from their local control nodes, they do not depend on this control information to work.

Another difference between the two approaches in the architecture design is illustrated in Figure 4-34. Even though the functional design of data processing nodes in both approaches is the same i.e. ASUSUBSYSTEM, the design of the local control node and the design communications between the ASUSUBSYSTEM and its local control node (i.e. LOCALHCISTATION) are different in each approach. The LOCALHCISTATION in RTN (called HCILOCAL), has the same functional decomposition as HCICENTRAL (see Figure 4-30 and Figure 4-34); i.e. two thread activities and a server component. One of these activities deals with data control

and the other activity with the visualisation of data coming from the data processing node and also interacts with the central control node. The design of the LOCAL-HCISTATION using CORBA is more complicated; as seen in Figure 4-34. The control data between the local control node and the data node ASUSUBSYSTEM, is done through the CORBA_SUBSYSTEM. The servants inside CORBA_SUBSYSTEM represent the control functions and the CORBA clients in ASUSUBSYSTEM represent the data functions. The other subsystem, i.e. HUMAN INTERFACE, visualise the data coming from the CORBA clients and also interacts with the central control node.

On the other hand, it may be stated that the last point is strongly dependant upon the implementation of CORBA. In other words, CORBA systems may be defined as peer-to-peer systems (i.e. all nodes have identical capabilities and responsibilities and all the communications are symmetric) or end-systems rather than client-server systems like ADVISOR Prototype. CORBA provides a service called Event Service, to obtain symmetry in the communication. Event Service allows the application to use decoupled communication between parts rather than strict client-to-server synchronous request invocations. The basic architecture of Event Service [Henning and Vinoski 1999] consists in Supplier and Consumer Modules, which can play passive or active roles, and Event channel, which plays the role of mediator. An event data can be delivered from the suppliers to the consumers with a decoupling of physical knowledge. Note that conceptually this idea is very similar to the simple communication model in RTN illustrated in section 3.5.3 chapter 3.

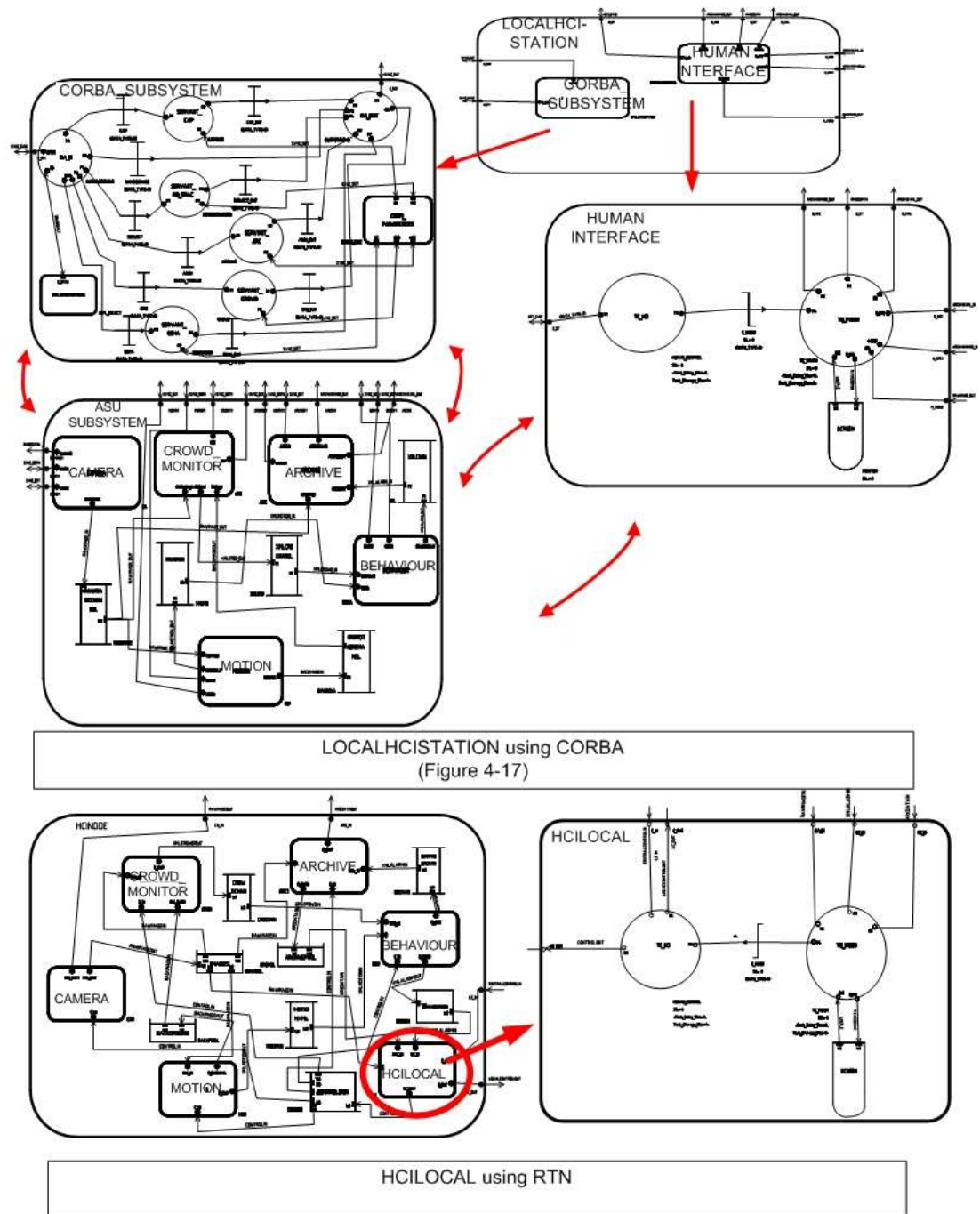


Figure 4-34. Comparison of LOCALHCISTATION subsystem designs using the two approaches.

In CORBA, as mentioned, the communication is established transparently to the designer (through the ORB layer), but as illustrated in Figure 4-34 and Figure 4-35, this communication requires extra components e.g. CORBA_SUBSYSTEM, making the architecture design more complex than the illustrated architecture design using RTN concepts (section 4.6). Moreover, the CORBA architecture

design of e.g. CORBA_SUBSYSTEM also illustrates the complexity of these interactions between different components inside the subsystems. Some of these activities require a tight relation with other activities, producing some coupling that it is not necessary as seen in section 4.6 using RTN. This is the result of having these dynamic interactions between the objects that constitute the CORBA design system, which are needed to allow the integration of objects to the system without extra effort. However, allowing the integration of objects to the system without adding effort to the programmer implies more dynamic interactions between objects are created, even though they are transparent to the programmer (as illustrated in Figure 4-19, Figure 4-20, Figure 4-23 and Figure 4-26). Some of these interactions may be strongly coupled and in some systems, such as real-time surveillance systems, such coupling may have costly effects such as producing deadlock situations.

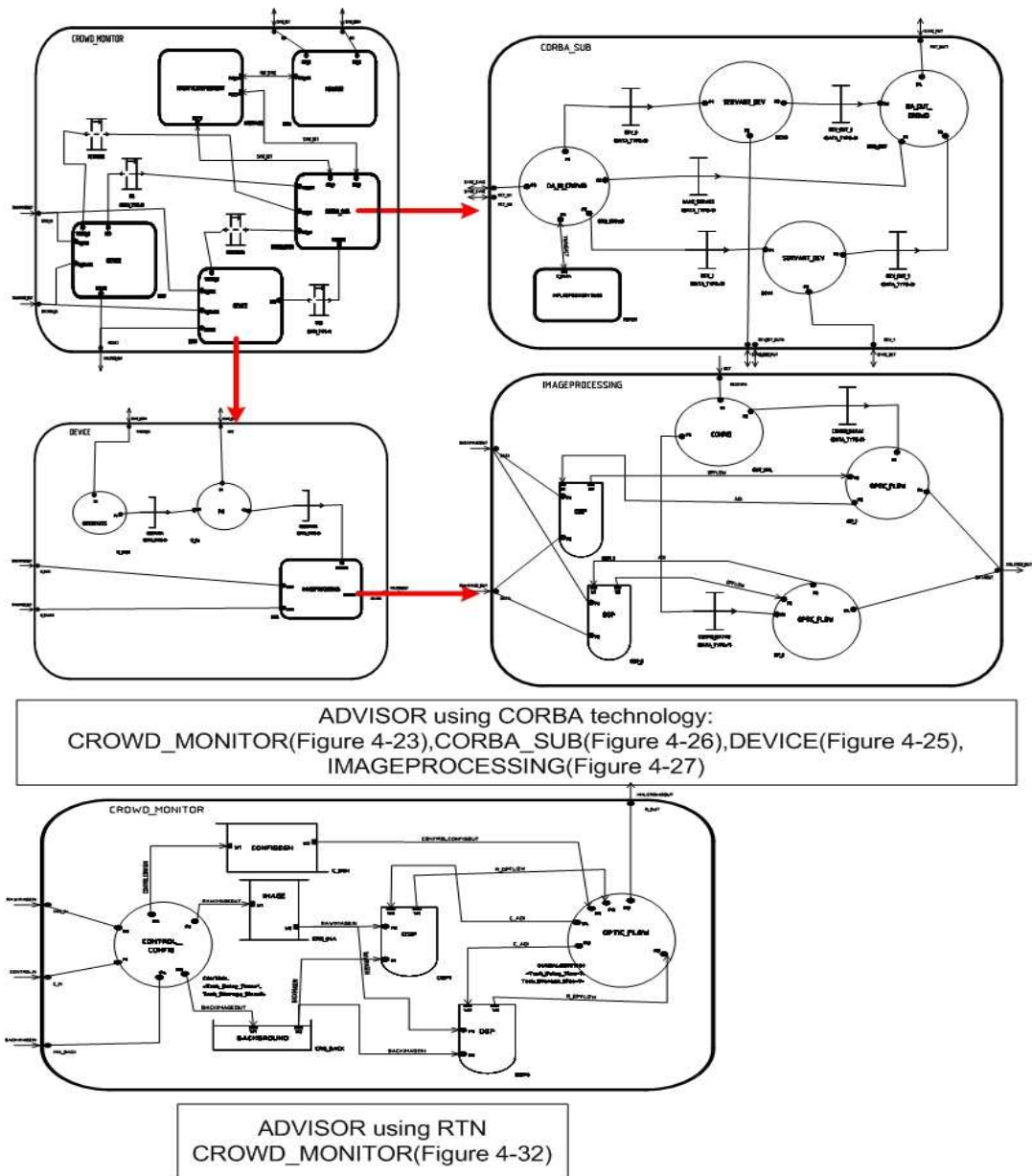


Figure 4-35. Comparison of CROWD_MONITOR subsystem designed using both approaches.

4.7.2 Concurrency and distribution

The key issue for developing concurrent systems is focused on structuring the system into the right number of concurrent tasks. It is also focused on giving the mechanisms to support inter-communication tasks and on allowing tasks to synchronise their operations (producer/consumer problem) and the access to shared data (mutual exclusion). It is also important to assure support for concurrent execution in the programming language or by the Operating System. In a single processor environment, tasking may provide an improvement in performance by allowing I/O operations to be executed in parallel. In a multi-processor

environment, tasking may improve the performance by allowing different tasks to execute in parallel on different processors.

A distributed application is a concurrent application. Thus, in a distributed application there are necessarily multiple threads of control. As mentioned in the previous chapter, any design in RTN is conceived as a network of concurrent threads called activities. Therefore, RTN bounds the resources by knowing beforehand the resources that are going to be required, because when the network is finally designed, the number of threads is automatically determined by the number of activities that appear in the design. These threads or activities communicate through passive components called IDAs that provide the necessary mechanisms to allow the inter-communication between activities, and the synchronisation of the access data (see the Taxonomy of the protocols in chapter 3). To apply the basic communication principle between different components in a distributed system, RTN uses a template substitution. Template substitution is a technique applied once the network is instantiated and mapped into hardware at the building process time. If the activities that communicate through an IDA are distributed in different places; the designed IDA is substituted by a new IDA template that allows the distribution by stretching¹¹ the already designed IDA.

In a conventional OO distributed application, each server object when distributed should operate in a different thread of control, because a distributed server object may have multiple concurrent clients, see e.g. Figure 4-19. Therefore, it is necessary to apply synchronisation mechanisms to control concurrent access to shared objects being distributed, which are provided by the threads themselves and not by external components as in RTN. CORBA, which is a standard for a distributed object systems, allows either single-thread or multi-thread ORB architecture. Therefore, if the single-thread ORB architecture is chosen, even though a distributed object may have more than one client, requests are forced to be processed in a sequential mode instead of concurrent mode as they naturally should

¹¹ As mentioned in chapter 3, the stretching technique in RTN, consists of introducing an activity in the IDA, which moves the data from the IDA to the next one, and therefore projecting the IDA to the other side. See extension taxonomy protocols in chapter 3 or e.g. IMAGE CHANNEL, PLAYBACKCHANNEL or XMLCHANNEL templates (in Appendix C-7, C-6 and C-8), which are stretched by using activities between simple IDA channels and forming by this means, the new templates called e.g. IMAGECHANNEL, PLAYBACKCHANNEL or XMLCHANNEL.

be; as mentioned before, a distributed system should be concurrent system. If the chosen CORBA architecture is multi-thread, CORBA provides three different design approaches: thread-per-request, thread-per-connection and pool-of-threads. Thread-per-request, as its name specifies, creates a thread at every incoming request; if there are many requests the server application may run out of resources because it has to deal with many threads. In thread-per-connection approach, a thread is created for every connection, forcing a server application to deal with many threads if the server has many clients, and it may incur in a thread creation overhead if the petitions are too frequent. In the last approach (pool-of-threads), at the start-up period of a system, a pool of threads is created; any non-busy thread can deal with a request: if all threads are busy the incoming request is queued. From the three approaches discussed, this is the most distributed approach and it follows the concept in RTN to bound the resources by knowing beforehand the resources that are going to be needed.

4.7.3 Run-time

Run-time support mechanisms, which are necessary in the construction of any real-time concurrent system, are focused on providing task scheduling policies, as well as mechanisms to support task communications and synchronisation. Also, these mechanisms need to provide support in the management of I/O interrupts and memory. The run-time support for concurrent tasks may be provided by the run-time support system provided by the concurrent language used or by the kernel of the operating system (e.g. in RTN this is called the MASCOT machine). These concurrent languages also handle task scheduling. On the other hand, if the run-time support is provided by a kernel, the kernel provides the task scheduling, it also provides the mechanisms for communication tasks and the synchronisation. In RTN, the scheduling strategy is left to the designer to allow the optimal algorithm for the application to be used, even though the MASCOT kernel machine usually applies a co-operative scheduling policy.

In RTN, the MASCOT machine provides primitive operations for timing, synchronisation and control of the execution of activities. The scheduler (in the MASCOT machine) controls the execution of the activities and also it allocates the

processing time for each activity [IECCA and MUF 1983b]. MASCOT-3 provides two primitive operations for timing: DELAY (i.e. specifies the period of time, which the activity may be stopped) and TIMENOW (i.e. returns the value of time). The synchronisation only takes place at the access to IDAs and servers (i.e. in the access procedures). This synchronisation achieved by four primitive operations that provide the mechanisms for mutual exclusion of competing processes (JOIN and LEAVE primitives) and cross-stimulation of co-operating processes (STIM and WAIT).

In CORBA, the run-time mechanisms to control the execution of clients and servers are specified in the ORB run-time properties, making these mechanisms highly dependent on each vendor. TAO is considered [CORBA 2005], at the time of writing, the high-performance real-time ORB for applications with deterministic and statistical Quality of Service (QoS). The TAO ORB Core uses multi-threaded, pre-emptive strategy with priority-based connection. The TAO's I/O subsystem assigns priorities to real-time threads. In [Marsden and Fabre 2001] it is illustrated with empirical results how latency, throughput and the CPU processing overhead behaviour drifts using the same real-time ORB middleware (TAO) architecture in different real-time operating systems such as VxWorks, LynxOs and other operating systems with real-time extensions like Windows NT, Solaris or Linux. TAO uses Real-Time Event Service from CORBA to alleviate some restrictions with CORBA standard invocations.

To apply efficient scheduling strategies, it is important to determine the boundaries of the endsystems (in CORBA terminology) or components of the system, to avoid non-deterministic behaviours from these components. In RTN designs, there is a restriction to the dynamic creation of components to bound the non-determinism, enforcing the minimisation of dynamic resources scheduling at run-time in the designs. Moreover, RTN designs are also thought of as multi-processor configuration to reduce process contention and with distributed shared memory to avoid memory access contention (dynamic invocations in DORIS notation implies 'datarepository' elements, see Figure 4-20 and IMPLREPOSITORY in Appendix C-21, pp.305). On the other hand, in CORBA designs, there are no restrictions to the dynamic creation of components. In Figure 4-19, it is difficult to bound the

dynamic resources consumed in the interactions between the ‘servant’ activities inside CORBA_ SUBSYSTEM with the ‘client’ activities outside the subsystem. Note that in Figure 4-21 or Figure 4-26, the ‘servants’ activities have been drawn to express the existence of these servants, but they could be created dynamically. Therefore, it is not possible to determine e.g. how many servants are going to be created and active at certain point of time, arisen a possible non-deterministic behaviour in the boundaries of the CORBA_ SUBSYSTEM.

4.7.4 Development aspects

In MASCOT and DORIS, the development of an application from its design to its creation is defined in three stages [IECCA and MUF 1987c]: the status progression, the system building and the development configurations which includes a mapping process to distributed hardware.

In the status progression stage, there are two main features: the modules that facilitate the elaboration of the design and the creation of an application software and a database which has an important contribution in the creation of these modules. In the status progression, a formal recognition of the development of the modules is carried out. The status value associated to each module provides a measure of this recognition. They are five different status values: registered, partially introduced, fully introduced, partially enrolled and fully enrolled. Once all the modules that constitute a system are fully enrolled it is possible to move to the second stage. Therefore, system building starts from a fully enrolled system template, and it produces a representation of this system in an executable form [IECCA and MUF 1987c]. There are different strategies employed in this stage and the target configuration for which the application needs to be built is considered: e.g. the number and type of processors available, the accessibility of memory from each processor and other requirements. In the last stage (i.e. development configurations) different hardware configurations appropriate for the MASCOT software might be discussed.

As mentioned in the introduction of this chapter; at the time of writing, CORBA does not provide any appropriate development environment. Even though there are

some tools provided by different companies such as Rational Rose and ArtiSAN and recommended by the OMG, to help design CORBA applications, basically using UML, they do not provide an environment to develop the application from the design to the execution.

4.8 Summary

This chapter has further compared RTN and OO approaches. The comparison in chapter 4 has been conducted by means of a case study. Therefore, a distributed real-time surveillance system solution called ADVISOR has been presented. This comparison focuses on the architecture design viewpoint for a distributed real-time system; issues such as communication, distribution, concurrency and run-time have been discussed. ADVISOR used a CORBA approach, which is an OO-based technology, as a solution to the design of its distributed architecture. Therefore, to continue the comparison discussed in chapter 3, this chapter has based the comparison between DORIS (the latest extension of RTN) and CORBA. It has been shown that even though CORBA may be a suitable solution for some real-time distributed applications like telecom systems, allowing an integration of different language platforms, it presents for the system requirements like ADVISOR, a more complex architecture design than RTN, as reflected in the figures presented here. The communication CORBA design of ADVISOR also illustrates that there is a strong coupling between the server and the client components (objects). In contrast, RTN avoids this coupling by using specific communication components that provide decoupling of the connected components. RTN designs attempt to create a network as deterministic as possible by explicitly defining the number of components constituting the real-time network system at run-time. It has also been shown that RTN/DORIS provides a full development environment for the creation of software applications but not CORBA. As discussed, RTN solutions are intended for concurrent, distributed, real-time complex applications. RTN gives the principles and the tools to create them. For these reasons, RTN can provide the basis for the creation of a framework to help the development of distributed real-time surveillance systems. To explore this further, in chapter 5 a proposed generic distributed real-time surveillance system using RTN is presented.

5 Design of a Real-Time Distributed Surveillance System with multiple cameras

5.1 *Introduction*

As mentioned in chapter 2, one of the requirements of 3GSSs consists in the on-line processing of data streams in real-time. Nowadays, this on-line processing may be possible to achieve thanks to the use of low cost imaging devices and embedded devices like Digital Signal Processors (DSPs) and to the steady increase of general-purpose computing power. As concluded in chapter 4, to meet real-time requirements, these systems should manipulate data streams in concurrent environments, designed by taking into account scheduling and synchronization issues. In the visual surveillance field, until now this has been mainly solved by building specialised systems using ad-hoc designs and implementations which sacrifice flexibility and performance [François and Medioni 2001]; issues which are important in large scale systems. This chapter proposes a generic, extensible modular software architecture design of a 3GSS using RTN/DORIS.

In this chapter, the designed system is presented graphically, following its hierarchical structure. Thus, the system is illustrated level by level to finish with the final network of activities and passive elements used to communicate (like IDAs) or to storage information (called repository data elements). In section 5.2, the first level of the design is illustrated by presenting the functional definition of the system. Moreover, all the RTN/DORIS elements used to design the system are also introduced in this section to give a reference to the reader. In section 5.3, the different functional definition of the modules that compose the system are presented and discussed. These modules are grouped in three main parts depending on their functionality; data processing, control and feedback parts. A distinction between the different types of data that are used in the system is also presented in this section.

In section 5.4, the design of the system architecture is presented. Note that the proposed designed system expresses how the software structure of the system (the system architecture) should be designed, but the physical structure of the system is not discussed. In section 5.5, different topologies for multimedia system

applications are presented, based on the networking literature. After this introduction to network topologies, section 5.5 presents a discussion on different network topologies that were proposed for the system design. After the discussion, a final network topology for the software structure proposed in previous sections 5.3 and 5.4 is then presented. Following this section, section 5.5.2 discusses the representation and design in RTN of a specific traffic behaviour (multicasting) heavily used in surveillance systems. In section 5.6, the management of the Quality of Service (QoS), which is an important issue for distributed multimedia systems, is discussed. There, QoS is discussed in terms of bandwidth and the selection of scheduling policies. Finally, section 5.7 finishes this chapter by summarising the obtained conclusions from the creation of the design of a large-scale surveillance system using RTN.

5.2 *First level of the system design*

This section describes the functionality of the proposed system architecture design focusing on functional definition (section 5.2.1) and then on a brief discussion of the RTN components used in this design (section 5.2.2).

5.2.1 Functional definition of the system

Figure 5-1 illustrates the functional definition of the system. It presents the system as a network of three functionally different types of subsystems or nodes:

- The Data Processing Unit node (DPU): is the node where the sensors (e.g. cameras) are connected. Most of the on-line and off-line data processing coming from the sensors is done in these nodes.
- The Communication Control 0 node (CC0): this node interfaces users with a DPU. Therefore, a user can change a configuration parameter of a DPU through a CC0. The CC0 node also stores the information of all DPUs connected to the same CC0 and allows the user to visualise the outputs coming from any DPU node that is connected to the CC0.
- The Communication Control 1 (CC1): this node visualises any output coming from any DPU. The CC1 node also provides storage for the information of all CC0 nodes that are connected to the CC1.

Similar to ADVISOR in chapter 4, in the functional design of the system there is a *local* node (CC0) that visualises the data processing outputs from the nodes that are connected to it, and there is another *central* node (CC1), which can visualise the outputs coming from all data processing nodes. Therefore, even though the nodes are functionally independent, the system has a hierarchical structure from the visualisation and the structural organisation (system user control) points of view. Figure 5-1 illustrates this hierarchical network structure. The system is designed as having three different levels. The top-level is represented by CC1 nodes, the second level by CC0 nodes and the low-level of the hierarchy is represented by DPU nodes. DPUs only process the signals coming from the sensors. The rest of the nodes (i.e. CC0 and CC1) analyse the alarms and visualise the data coming from the sensors. The system follows the user hierarchical structure that wide-area surveillance systems have (there are local control operators that survey a local area). In first upper level (CC0), the operators control one *zone*, i.e. more than one *local area*. In the following upper level (CC1), the operators survey *all zones*. Figure 5-1 describes two levels of surveillance control through CC0 and CC1. The distinction of two main functionalities in the system i.e. data processing expressed in the design through a DPU node and visualisation and organisational control through the design of CCx¹² nodes, is a similar idea that appears in some research work reported in [Marcenaro et al. 2001] and in [Christensen and Alblas 2000].

In [Marcenaro et al. 2001], the functionality of the system is mainly divided by sensor and hub nodes. All sensor nodes are connected to hubs. They present an empirical discussion in terms of performance and bandwidth allocation, about the distribution of the data processing tasks in the sensor or hub nodes. The authors state that in a surveillance system that is constituted of several cameras (with embedded DSPs) connected to a hub and which remotely sends processed data to an operator, it is better (empirically proved) to allocate the embedded low/high-level signal processing tasks performed in the system, on the hub node if the number of cameras is less than two (for a high processing power of the cameras (i.e. the

¹² In this chapter, because the functionality is similar in CC0 and CC1, these nodes are generalised by the term CCx.

embedded DSP of the cameras) like e.g. 450 MHz). It is also better to allocate the tasks in the hub node if the number of cameras is less than six (for low processing power cameras such as 200MHz). Otherwise, it is better to allocate the tasks on the cameras rather than in the hub.

[Christensen and Alblas 2000], as mentioned in chapter 2, present the design of a surveillance system with three cameras. The design consists of three “crunchers” nodes that realise the low-processing of the signals coming from the three cameras. Thus, each node is attached to a camera. In that work, the functionality of the design system is also divided in two main parts: the “crunchers” nodes and the “hub” or “data analyzer” node. Each cruncher is also attached to a local database. The nodes are communicated between them in a fault tolerant way using a mesh network structure. There is another (i.e. the “data analyzer”) node that analyses the processed data coming from the crunchers and this node is connected to a global database. Apart from all this nodes, there is a monitor node that is connected to one “cruncher” and that allows to visualize the signals coming from the crunchers. In this case, the authors follow the idea of embedding the low-level tasks including a local archive in each camera (three in their case) and to allocate the high-level processing tasks, including a global archive, to a “hub” node.

The network structure of the system proposed here consists in a hybrid of three different network architectures, which is discussed in more detail in section 5.5. The design decision of the network structure comes from a compromise between fault tolerance and scalability. Note that, even though in Figure 5-1 only CC0 and CC1 appear, the design of this system is intended to be as scalable as possible. Therefore, it is possible to scale the system by introducing a CC2 node, which introduces another level in the structure of the system, allowing the integration of another hierarchical level in the system. However, this might not be necessary nor advisable, because the system can, in fact, grow without introducing another level (with the consequent complexity that this implies).

The main difference between what is proposed in [Marcenaro et al. 2001] and [Christensen and Alblas 2000] and what is proposed here, lies on the functional definition of the “hub” node. In [Marcenaro et al. 2001] and [Christensen and

Alblas 2000] the functionality of the “hub” node is to concentrate the signals coming from “sensor” nodes, and to apply some high-level processing tasks (depending on the number of sensors that are attached and on their power capacities). In this proposed design, the “hub” node or CCx node does not only concentrate the signals but visualises and controls the signals coming from the “sensor” nodes or DPU nodes. Moreover, the CC1 nodes also control the signals coming from the CC0s.

Furthermore, [Christensen and Alblas 2000] work relates to the design of a specific system using three cameras with high processing capacity, while the design proposed in this chapter is independent of the number of cameras and their power capacity, because these matters are more appropriately dealt with in the physical mapping phase instead of the design phase. Therefore, note that in the proposed design, there is no discussion of the physical distribution of the tasks at this level of the design because as, mentioned in chapter 3 and chapter 4, the design phase of a system using Real Time Networks is transparent to the physical distribution of the tasks. This is in fact, a strength of the method.

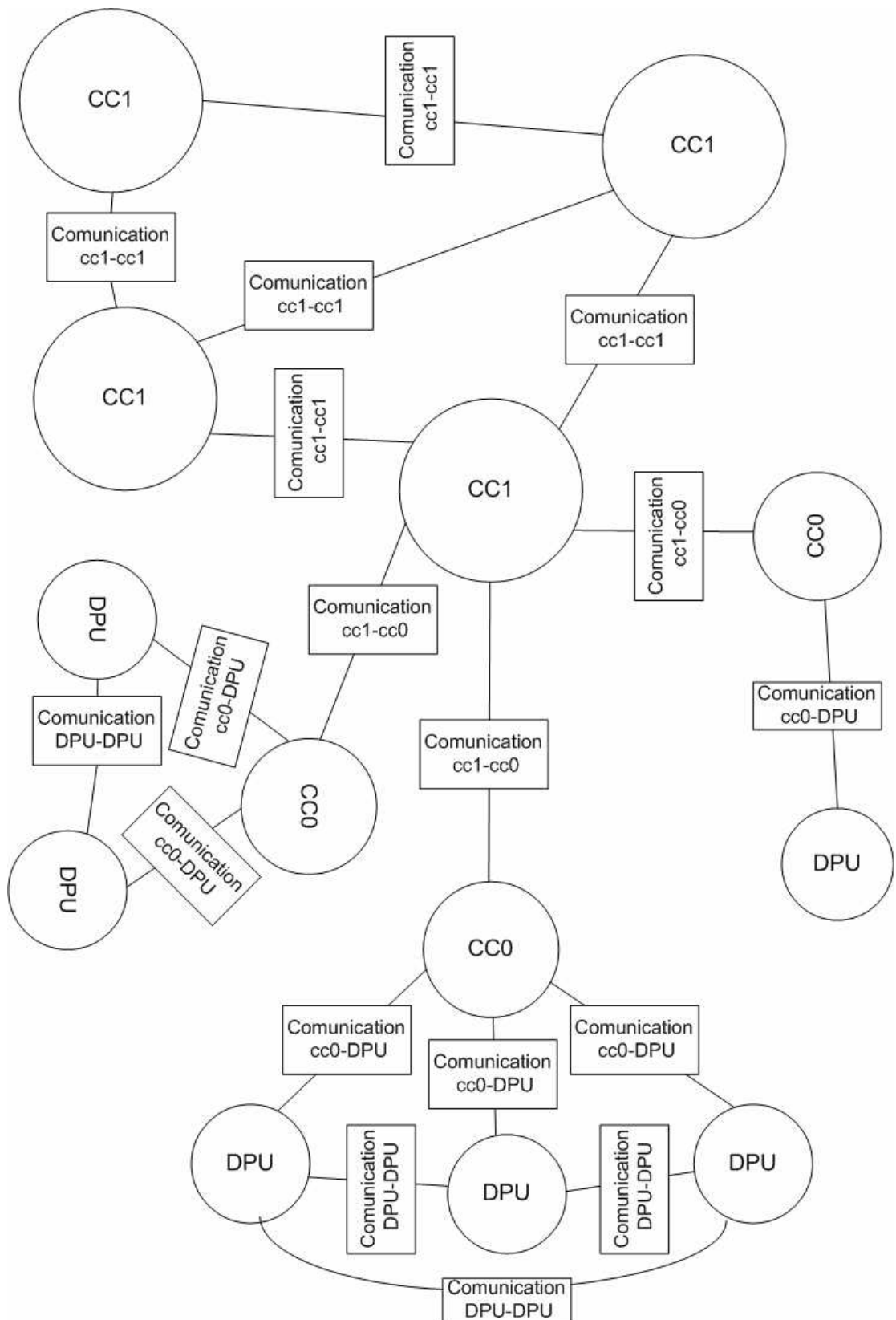


Figure 5-1. Functional representation of the system.

5.2.2 RTN components used in the design

Table 5-1 lists some of the components used in the design of the system. The component that represents the system as a whole is called LSIVSS (an acronym for Large Scale Intelligent Visual Surveillance System). Inside this system there are other subsystems, which represent the previously mentioned CC0, CC1 and DPU nodes. The design of the system is intended to be as modular as possible. The subsystems are designed as groups of elements that have the same functionality and that are independent of the rest of elements than do not belong to the same subsystem. The only connexion to the outside of the subsystem it is done through ports, windows and servers. The server components are used to represent the interaction of the system with the outside world. The system interacts with the user through a server called e.g. Screen, which represents a computer to visualise the outputs, and a server called e.g. Kboard, which represents a computer to allow the user to send commands to the system.

RTN COMPONENTS	MAIN DESIGN TEMPLATES NAMES
System	LSIVSS
Subsystem	CC0, CC1,VISUAL
Activities	OD_OR
Protocols	A_E(signal), I_C(pool), S_I(channel), TRIGGER (flash)
Protocols	S_DATA_IN(RPC), S_DATA_OUT(RDS)
datarepository	Local_DPU_Info, LAR, DPU_info_Module.
Server	Screen, Kboard, camera

Table 5-1.Summary of some of the RTN components that appear in the proposed design system.

Therefore, one of the design decisions has been to group each node in different subsystems representing the different functionality of each node in the initial functional design. The CC0 node is represented by the subsystem called CC0, the CC1 node is represented by the CC1 subsystem and the DPU node is represented by the subsystem called DPU. Inside each of these subsystems there are more subsystems. For example, the subsystem called VISUAL (part of CC0) represents the visualisation part of the system. The description of the different components of each subsystem is presented in section 5.3.

Another important design decision corresponds to the final number of activities into which each subsystem is decomposed. As mentioned in chapter 4, a network that is designed with too many activities may carry out a penalty in the performance of the system, because each activity that represents a thread also involves a context switch. To design the system with too many activities implies a system with a possibly unnecessary number of context switches which may reduce its performance. Even though there seems to be no method that allows the calculation of the number of activities that is required in a specific application design, as a rule of thumb [IECCA and MUF 1983b] advice that the activities should have few ports. If an activity has more than two input and output ports it may be necessary to decompose the activity further. However, some of the designed activities that are presented in this chapter have more than two inputs or ports, because in order to carry their work, the activities need more than two inputs (which it is quite common for the design of surveillance systems architectures) or because the output resulting from these activities should be sent to different parts of the system.

In the design presented in this chapter, each required image processing algorithm such as motion detection, tracking or behaviour recognition has been represented by an activity. Therefore, some of these activities have more than two input ports or more than two output ports, such as the activity that represents the motion detection algorithm. This algorithm for example requires (as an input) the image from which the algorithm has to detect the motion parts, and it also requires (as an input) the background image to be able to extract the motion components (please see the diagrams in Appendix B). Therefore, the activity that represents the motion detection algorithm has at least two input ports.

The actual nature of the communication between activities or/and subsystems determines the types of protocol should be used. It is possible to reflect such needs clearly in the design, because RTN provides a rich family of possible protocols (e.g. compared to CORBA). Therefore, the use of the protocols that appear in Table 5-1 (e.g. signal and flash) to communicate the subsystems and the activities, becomes another important design decision. [Haveman 1997] illustrates through mathematic analysis the difference in time performance between using a pool or a signal

protocol to communicate two subsystems. The protocols used in the proposed design are the signal, flash data, channel, pool, constant, RPC and RDS protocols (please refer to Figure 3-3 and Figure 3-5 to see the taxonomy of the RTN protocols). As presented in Figure 3-3, the writer is never blocked but the reader may be blocked using a signal protocol. Therefore, this protocol is used when the data to transmit between activities or subsystems is sporadic such as alarm events or control signals. Flash protocol, as mentioned in chapter 3, is a variation of signal protocol and represents a signal protocol with zero buffer capacity. In the pool protocol, neither the writer nor the reader are blocked, then this protocol is used with periodic signals such as the input signals coming from the sensors or background actualisations. Moreover, the use of a pool protocol implies full asynchronous communication between the activities and therefore the use of this protocol provides temporal independence between the communicating activities. Finally, in the channel protocol, either writer or reader can be blocked. This is used to represent message passing communication between the activities, or used when a synchronisation on the communication is required such as the communication of configuration data parameters.

The last component used in this proposed design is the datarepository component (Table 5-1). This component is used as an archive to permanently store data such as the information of each node (location, ID) and to storage images coming from the sensors and events detected by the system. To put and obtain data stored in this datarepository component in a synchronous manner other types of RTN protocols are used i.e. RDS and RPC (please see Figure 3-5).

5.3 Functional description of different parts of the design

In terms of requirements, surveillance systems should transmit video effectively, should allow the visualisation of video scenes to aid real-time monitoring, should facilitate the extraction, processing and access of objects and events in real-time, and should recognise scenarios. All these requirements are grouped in different parts in the proposed system design.

The system has five different constituting parts: monitoring units, data processing and archive units, communication units, control units and feedback units. Apart from section 5.3.1, which describes the type of data used in this system, the following sections describe in detail each of these parts of the system but not the communication unit because each communication unit is defined separately as a group of some of the protocols mentioned in the previous section.

5.3.1 Classification of data used in the system

To design a system it is important to define and to classify the type of data that the system requires and uses. Apart from the input data coming from the outside world, there is some data that it is used inside the system, which may suffer transformations while is used or transmitted through the different parts that constitute the system. These transformations may be important and therefore they are required to be stored even when the system is not working. This type of data is defined as *persistent* data. If these transformations are used as intermediate data between modules then these data are defined as a *volatile* data. As mentioned in chapter 2, there is a growing research on establishing standard formats data for surveillance systems. For example, in ADVISOR the input images data coming from the sensors or from the archive are JPEG images while the rest of the data corresponding to the description of events or the system information setup are defined as XML streams. A poor selection of the input format data of the system may reduce the performance of the system. In [Mähönen and Saaranen 2000] the authors classify the use of different image formats such JPEG or MPEG2, MPEG4 for different multimedia applications.

5.3.1.1 Persistent data and local storage

Different types of persistent data that exist in the system, produce the creation of subsystems that are different depending on the use of this data and on the number and the ways in which this persistent data needs to be accessed. There is persistent data that it is used only by some algorithms that are implemented in the system; e.g. a background estimation algorithm produces the background of an image, which is required by other algorithms such as motion detection or tracking. Therefore, the background image data should remain in the system permanently so that other components in the system can consult and obtain the data. The background

estimation algorithm not only creates the background image but it also actualises it. Therefore, this type of data such as background image data, should not only remain in the system but should be constantly updated. In this design, the pool protocol is used to store this type of persistent data (because the reader cannot destroy the data (i.e. reference data) but the writer can always update, destroying previous values).

Another type of persistent data that should remain in the system but in this case which is not necessarily updated, is data that has been obtained from an off-line process such as the 3D scene model of the place from where sensors are capturing the data. In the case that the sensor is a camera, the camera calibration parameters may be considered as persistent data, which also do not require to be updated (if the camera is fixed). For this type of persistent data, the constant protocol is used, because once the data is inserted in the protocol only the reader can interact with the data without being able to destroy the data.

The persistent data that not only is used by the system but by the user too, is stored in the archive component illustrated in Figure 5-2. This component represents the archive of images and events recorded by the system that the user can consult and visualise. As seen in Figure 5-2, the LAR component has one incoming signal and two output signals. The *Trigger* signal is used to register error signals from the archive component such as “the archive is full” (see *Gar_Data* in Appendix D). The *AsynchIn*, as the name implies, is used to insert data in the datarepository component in an asynchronous manner. The signal called *AsynchOut* is used to retrieve data, also in an asynchronous manner.

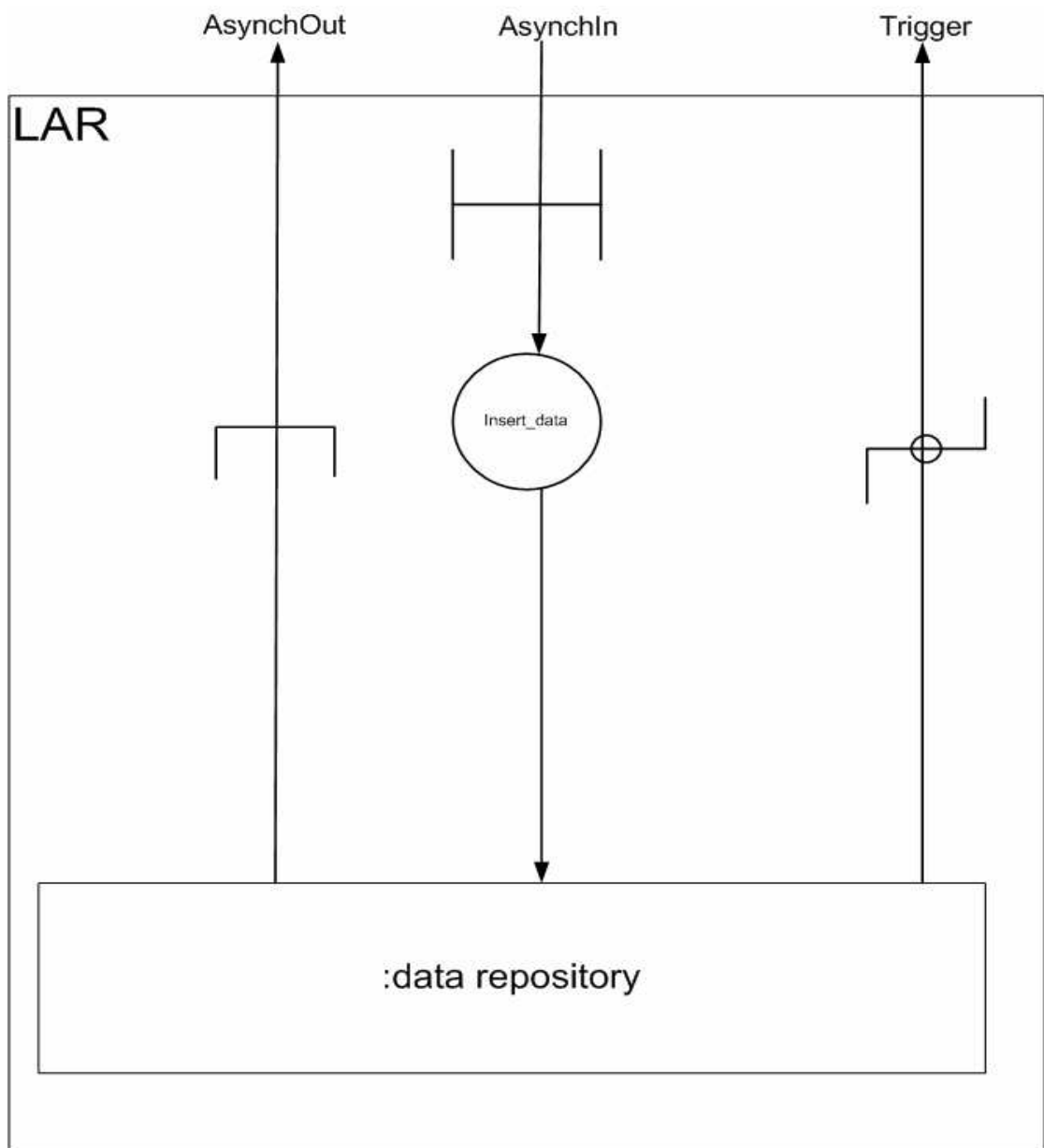


Figure 5-2. Local ARchive (LAR) component. Note that in Appendix D this subsystem the signal *Trigger* does not appear.

Another type of persistent data is used either by the user or by the system to have information of the organisational structure of the system includes information such as: the number of cameras in the system, the number of DPUs, the number of CC0 and the number of CC1 nodes, or the links of coverage areas between DPUs (this is explained below). As mentioned, several designed subsystems contain different data repository components that store this type of information (Figure 5-3, Figure 5-4, Figure 5-5 and Figure 5-6) as explained in some more detail now.

Figure 5-3 presents the Local_DPU_info_MODULE subsystem. This subsystem has a data repository component that contains the local information of a DPU node. As shown in Figure 5-3, this local information consists of the ID of the DPU node, the number of sensors (e.g. cameras) connected to the DPU node, camera parameters, parameters needed by some algorithms like thresholds and a description of the zones covered by the cameras. If there is a zone covered by more than one camera, i.e. a link between cameras, the link information and the cameras ID that are linked are stored. Finally, the link between DPUs and the ID of the DPUs linked is also stored. All this information is used by the CC0 node to which a DPU is connected and it is also used by different activities that a DPU contains. Therefore, there are two different types of access to the data repository component. One of the accesses is done by the three activities that appear in Figure 5-3, i.e. CIUcc, CIUdpu and L_ToDPU that deal with requests to *get* and *put* information. The requests to CIUcc and CIUdpu come from the CC0 and the requests to L_ToDPU come from another DPU. The CIUcc is waiting for a sporadic control signal coming from the CC0 asking for some information stored in the data repository. Once the CIUcc receives the signal, it gets the required information and sends it back to the CC0. At the same time the CIUdpu inserts the information that receives from the CC0. The other type of access to the data repository is coming from the activities that the DPU contains (see *Asynch out* and *Asynch in* signals in Figure 5-3). This is similar to the access processes in the local archive illustrated in Figure 5-2.

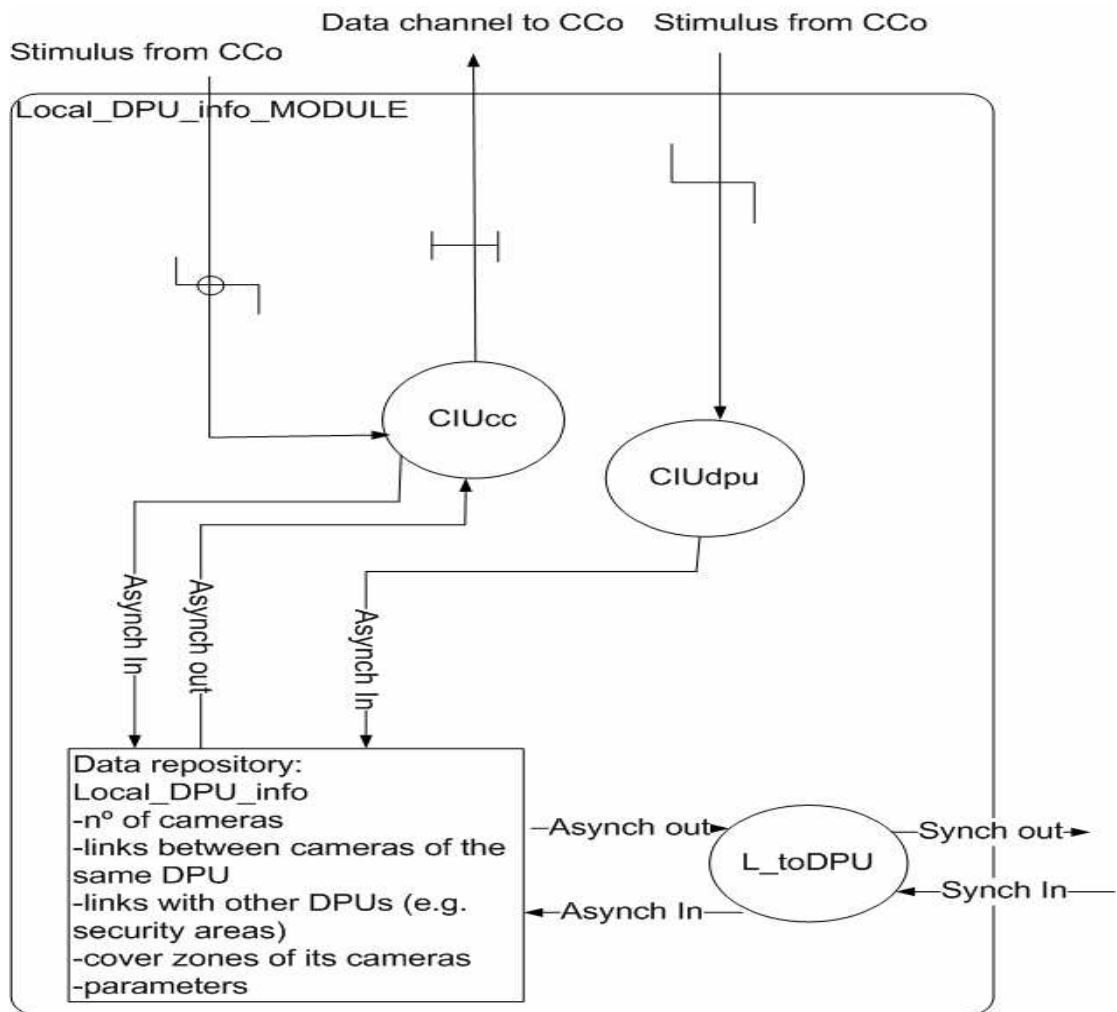


Figure 5-3. The design of LOCAL_DPU_info_MODULE.

Figure 5-4 presents the subsystem called `DPU_info_MODULE`. This subsystem deals with the information needed by the `CC0` node. The data information that this subsystem retains is similar to the data information stored in `Local_DPU_info_MODULE`. However, data such as the camera parameters or constant parameters that some algorithms in a DPU require, are not stored in the data repository component of the `DPU_info_MODULE` subsystem, but instead are stored in the `Local_DPU_info_MODULE` module. In the `DPU_info_MODULE` subsystem, the data is inserted by the two activities that appear in Figure 5-4; i.e. `CItoDPU` and `addInfoDPU`, and the data is retrieved by a `CC1` node through the `ToCC1` activity, (see Figure 5-4). `CItoDPU` sends sporadic signals to each DPU that is connected to the `CC0` node. Once `CItoDPU` sends one signal to one DPU, it checks periodically when the data information coming from that DPU has arrived (through the data channel, see Figure 5-4). The data information consists in the ID

of the DPU, the number of cameras attached to this DPU, the links that this DPU has with other DPUs and the areas that this DPU shares with other DPUs. Therefore, the functionality of the CltoDPU activity is to collect and store all the information of the DPUs that are attached to the CC0 node. Once this information is received, the CltoDPU stores this data in the data repository component. In parallel, the addInfoDPU activity may receive data coming from a CC1 node, for example if a user in the CC1 node needs to change a parameter such as a threshold or a camera position in a specific DPU, the signal containing this change information is sent to the addInfoDPU activity (through the ToaddInfoDPU interface, see Figure 5-4). Afterwards, addInfoDPU stores the received information and at the same time it transmits the information to the corresponding DPU. At the same time, CC1 may retrieve data from the CC0 node (e.g. number of DPUs that are connected to the CC0 node) in asynchronous manner (please see *Asynch Out* and *Asynch In* from ToCC1 activity in Figure 5-4).

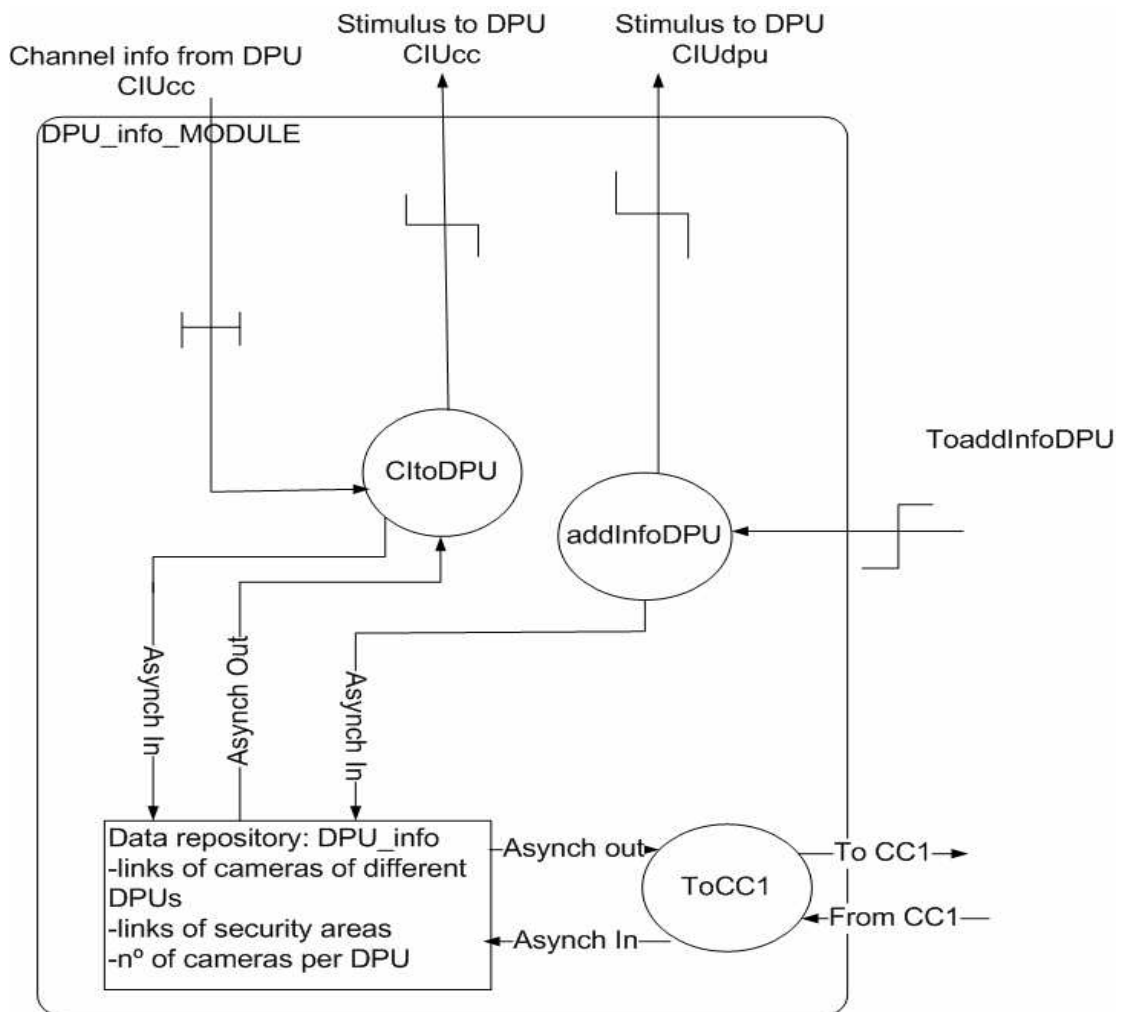


Figure 5-4. Decomposition of DPU_info_MODULE subsystem.

Figure 5-5 illustrates the design of the CC0_info_MODULE subsystem. This subsystem deals with data that is used by CC1 nodes. The data information, that the datarepository component of this subsystem stores, consists of: number of CC0 connected to the CC1 node, the ID of each CC0 and the DPU information of the each DPU connected to this CC0. There are two types of access. One of the accesses is done by the activity called CIUcc. This activity sends a signal sporadically, to each CC0 node that is connected to the CC1 node asking for the information of the CC0 (i.e. the information stored in the DPU_info_MODULE subsystem). Then, CIUcc checks periodically for the arrived data. Once the data is in the buffer, CIUcc reads the data and stores it in the datarepository component. The other type of access is used by another subsystem that it is contained in the same CC1 node (i.e. VISUAL_CC subsystem), which is discussed later on. This access is the same as the local archive shown in **Figure 5-3** (see CC0_PetiVisual activity in **Figure 5-5**).

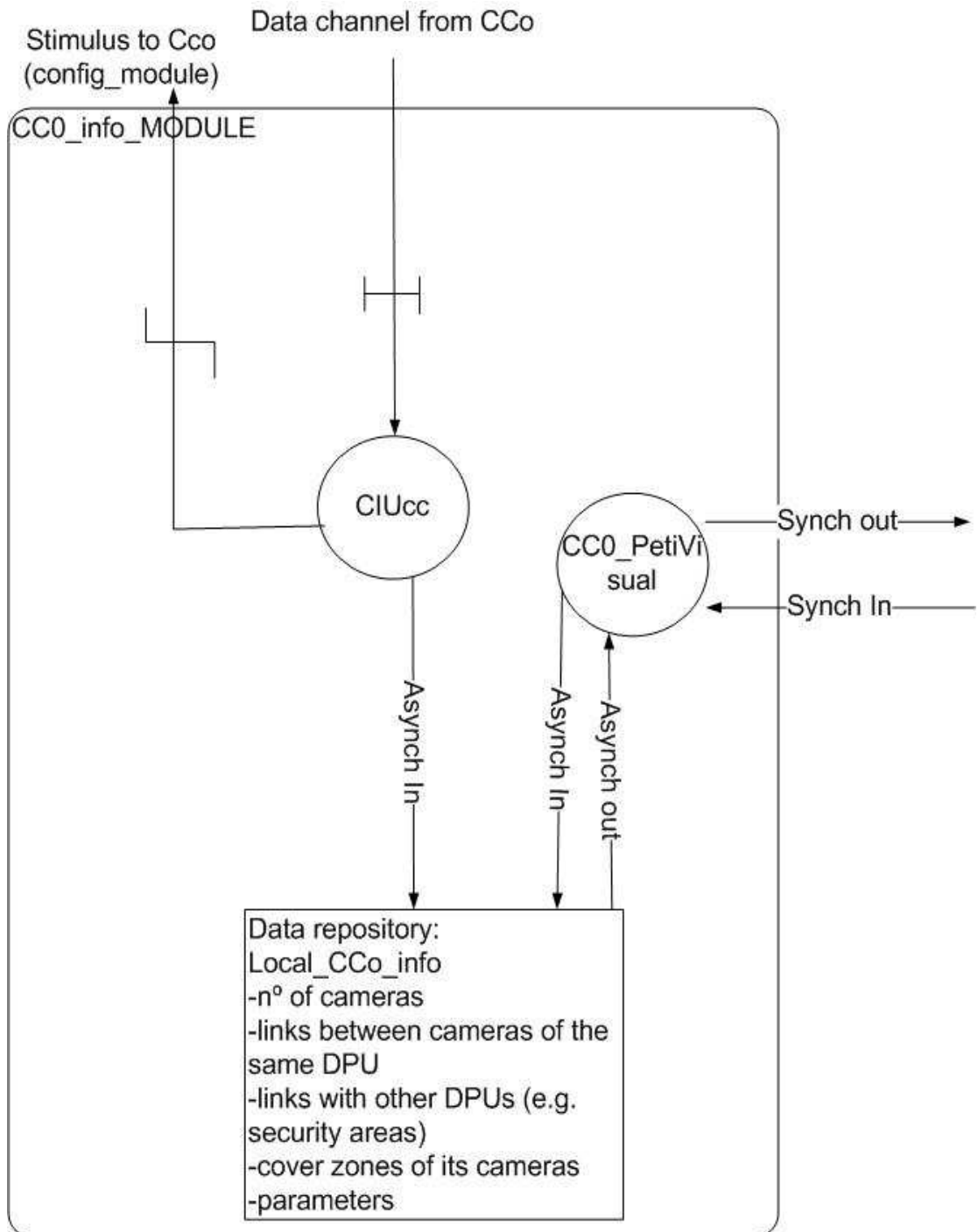


Figure 5-5. Design of CC0_info_MODULE.

Figure 5-6 represents the subsystem called **CC1_info_MODULE**. This subsystem also deals with data that is required by CC1 nodes. The information archived in this subsystem corresponds to information of the CC1 node itself, information about other CC1 nodes that the CC1 has been able to retrieve and also information about the DPUs that are connected to these neighbours CC1. This information is stored because it may be used when the CC1 node needs to get some data from a DPU that

is not connected directly to it, but connected to another CC1. All these datarepository subsystems and their connections are needed to communicate for example, a Station A, which is attached to a CC1_1 node (that surveys *zone 1*), with a CC1_2 node (which surveys *zone 2*, but requires information from Station A). Note that, one or more DPUs are located in Station A.

At the same time, this information may be used if the system is scaled and upper level control nodes are added such as CC2. A CC2 subsystem would use afterwards this information to get the information of the organisational structure of the system and to know which CC1 nodes are connected and consequently which CC0s and DPUs are connected directly to the CC2.

There are three different types of access. One of the accesses is done by the activity called TPC. When this activity receives sporadically a signal requiring information, TPC retrieves the information from the datarepository component and sends it back to the CC1 node that asked for the data. In parallel (i.e. the second type of access), a CIUdpu activity puts in asynchronous manner the data that it had previously required from another CC1 node. This information consists of the ID and the location of a DPU. This information is used by the CC1 node to connect to the DPU. Therefore, when CIUdpu gets the information, it not only stores it, but it also sends it back to another subsystem in a CC1 node (i.e. the VISUAL_CC subsystem) through PetiVisual activity, which uses this information to connect with the specific DPU that is attached to another CC1 node. The third type of access is the same as the local archive access that **Figure 5-3** and **Figure 5-5** illustrate. See *Asynch In* and *Asynch out* in **Figure 5-6**.

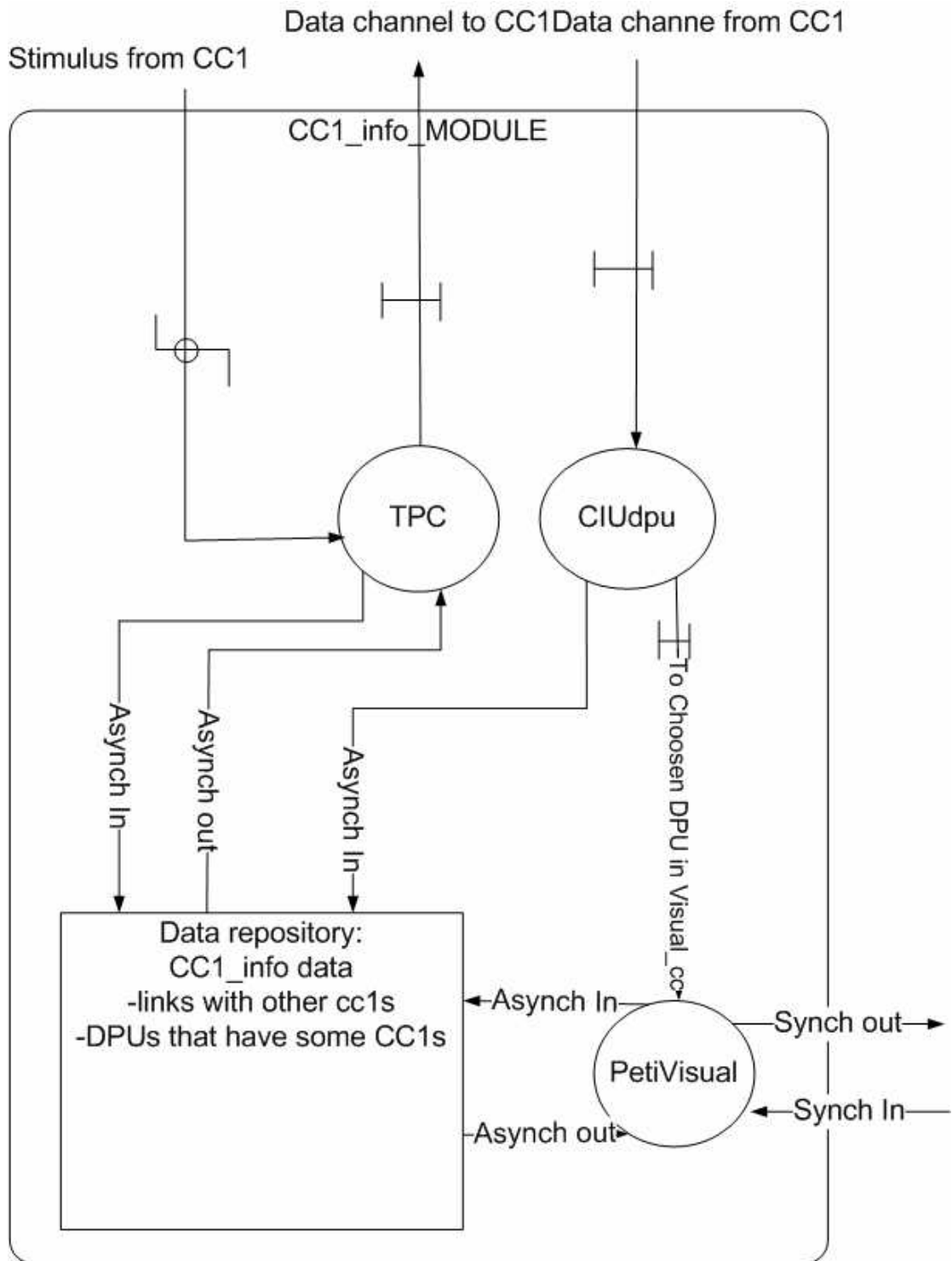


Figure 5-6.Design decomposition of CC1_info_MODULE.

The last type of persistent data that is found in the system is, as Figure 5-7 illustrates, the data information about the profiles of any user connected to the system. This data information is important to control access to system information by different types of users. Moreover, this information can also be used to track the number of mobile users that are connected to a CCx node at any time. Therefore, if

any alarm happens, the CCx can warn any user connected at this time, if the user has the right permissions. Then, when a user connects to the system, it sends a signal with his/her profile to the DB_User activity in Figure 5-7. Then, this activity registers this user in the CCx node by inserting the user information in the datarepository component. Once an alarm event is raised, a signal to the Search_User activity is sent and then this activity looks up into the datarepository component if there is any user connected at that moment, who should be warned by this alarm. If there is any user, then it sends the alarm through a *toDA* signal to the user or users.

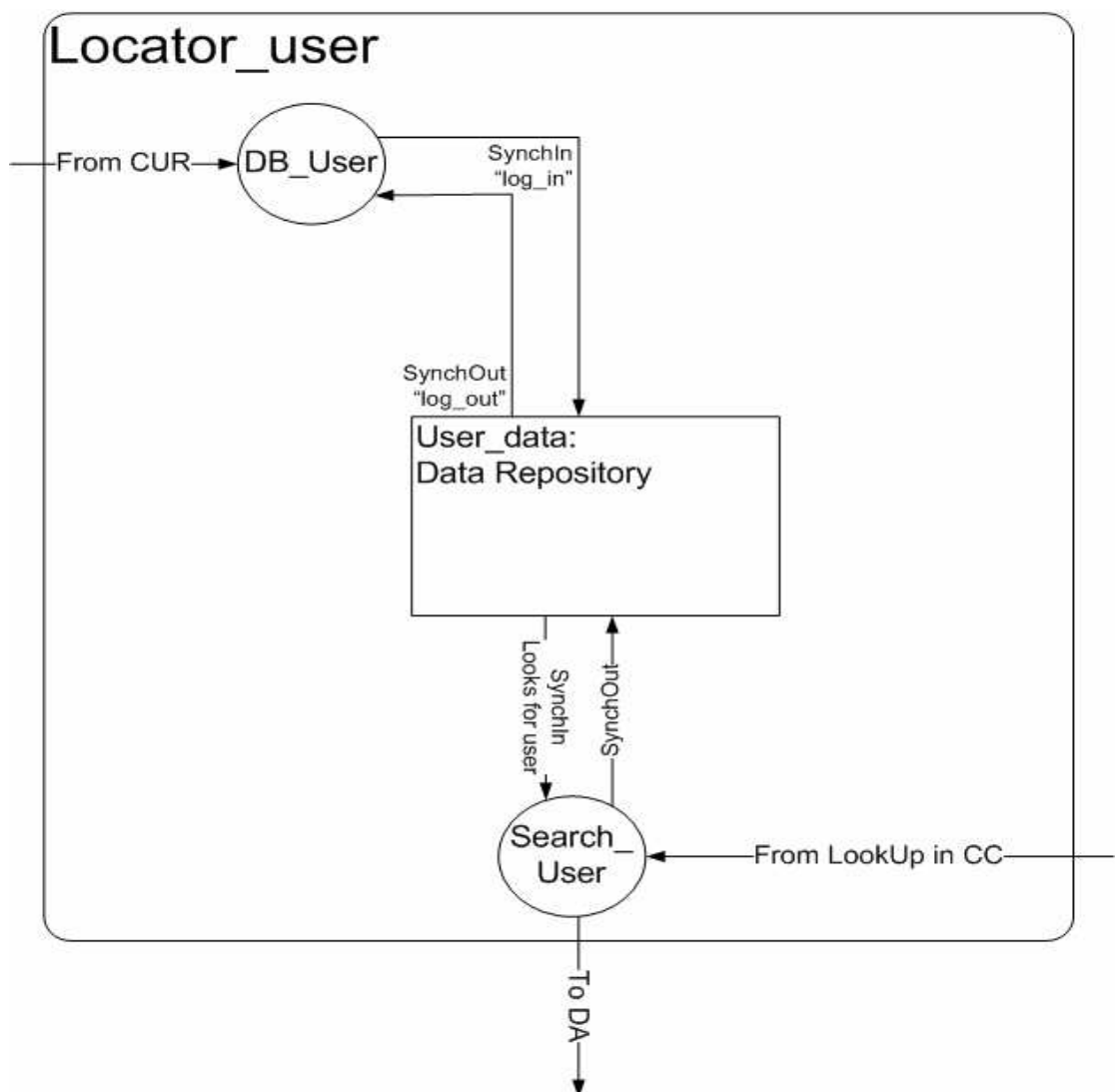


Figure 5-7. Composition of the subsystem called Locator_user.

5.3.1.2 Volatile data

Volatile data is by definition transient and there is no need to store it, as occurs with persistent data. In the design, data that is used between activities, is considered volatile. For example, the data that any of the activities in the CC1_info_module (see Figure 5-6) receives from other subsystems such as TPC through the flash protocol, is considered volatile data, until it is stored in the datarepository component.

5.3.2 Monitoring part of the design

The functionality of the monitoring part of the system consists in visualising: the processed data such as alarm events, archived images or live images. There are three subsystems in this design whose functionality is to monitor signals in the system. This is shown in Figure 5-8, Figure 5-9 and Figure 5-10 and described in some detail below.

As mentioned in section 5.2, besides the data processing functionality represented by DPU nodes, there are other functionalities which are grouped and represented in CCx nodes. One of these functionalities is to monitor the outputs from DPUs. Therefore, each CCx contains at least one subsystem whose functionality consists in allowing a user to visualise constantly the results produced by the system. Figure 5-8 illustrates one of these subsystems, contained in CC0, called the VISUAL subsystem. This subsystem allows the visualisation of real time images from any DPU connected to this CC0, if and only if the user that is connected has the permissions to do it (see the server component called “Kboard to user permissions” in Figure 5-8). The VISUAL subsystem also allows monitoring archive images and the alarms that are raised by the system. There is another server called “Kboard to choose DPU” that allows a user to change the visualisation to a specific DPU. If the user chooses this option, this “Kboard to choose DPU” server receives a signal that will send it to the “chosen DPU” activity. This activity will send this signal to the right DPU so that the VISUAL subsystem can start to receive inputs from this specific DPU.

Moreover, the VISUAL subsystem may receive either real time data or alarm event data and also playback data from other CC0s if the user sends through “Kboard to choose DPU” server a signal (see “send a signal to CC1 to visualise a DPU from another CC0” in Figure 5-8) to the CC1 asking for data of a DPU that is connected to another CC0. Note that this data comes from the CC1 instead of the CC0, where the DPU is connected, because there is no direct connection between CC0s.

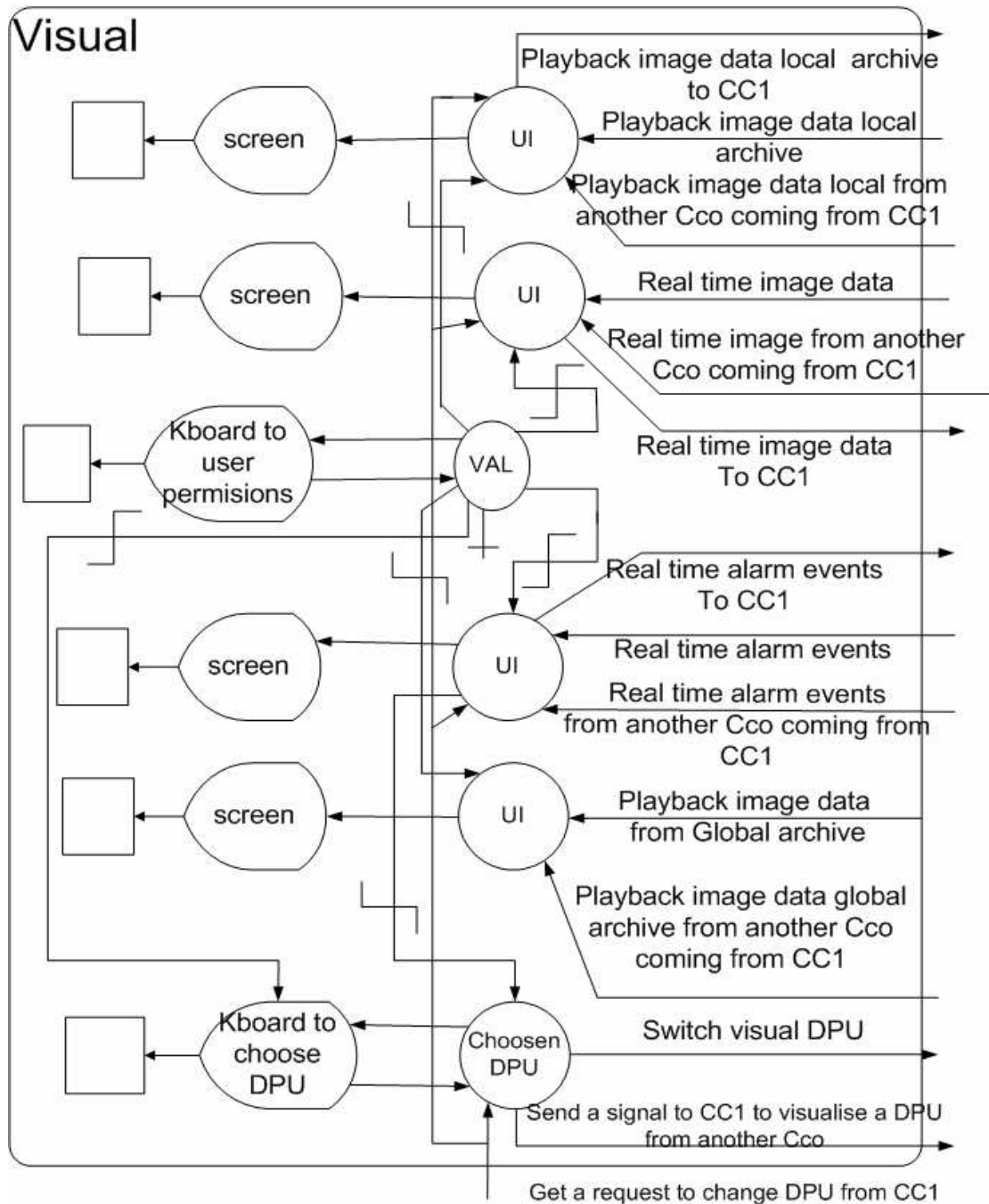


Figure 5-8. The design of the subsystem called VISUAL.

Figure 5-9 represents the visual subsystem called VISUAL_CC used in CC1 nodes. The main difference with the previous VISUAL subsystem is that, in VISUAL_CC it is possible to visualise data from other CC1s (i.e. from other DPUs that are not connected directly to the CC1 node). The DPUs that are connected directly to a CC1 are the ones that are connected to one of the CC0s, which is connected to the CC1 directly. In Figure 5-9, for example, three different “real time image data” signals are shown. One of these three signals represents the data that may come from any of the DPUs that is connected to the CC1 directly. The other signal represents the data that is received from a different CC1. The third signal represents the real-time data that the CC1 sends to another CC1 node that had required it. This real-time data comes from one of the DPUs connected directly to the CC1. This data is sent only if the CC1 receives a request signal illustrated in Figure 5-9 as “Gets request from another CC1 to send data”.

Therefore, in VISUAL_CC subsystem, a user can ask for data coming from a DPU that belongs to another CC1. To do that, a signal is sent to the “Chosen DPU” activity to look for the location and ID of the DPU, from which that user wants to monitor, in the CC1_info_module and the CC0_info_module. If the information is not stored into the CC1_info_module and the CC0_info_module yet, the CC1 communicates directly with each neighbour CC1, until it gets the information. Once it receives the information, it archives the data in both subsystems i.e. CC1_info_module and the CC0_info_module. It establishes then the connexion with the CC1, where the DPU is connected, to get the data.

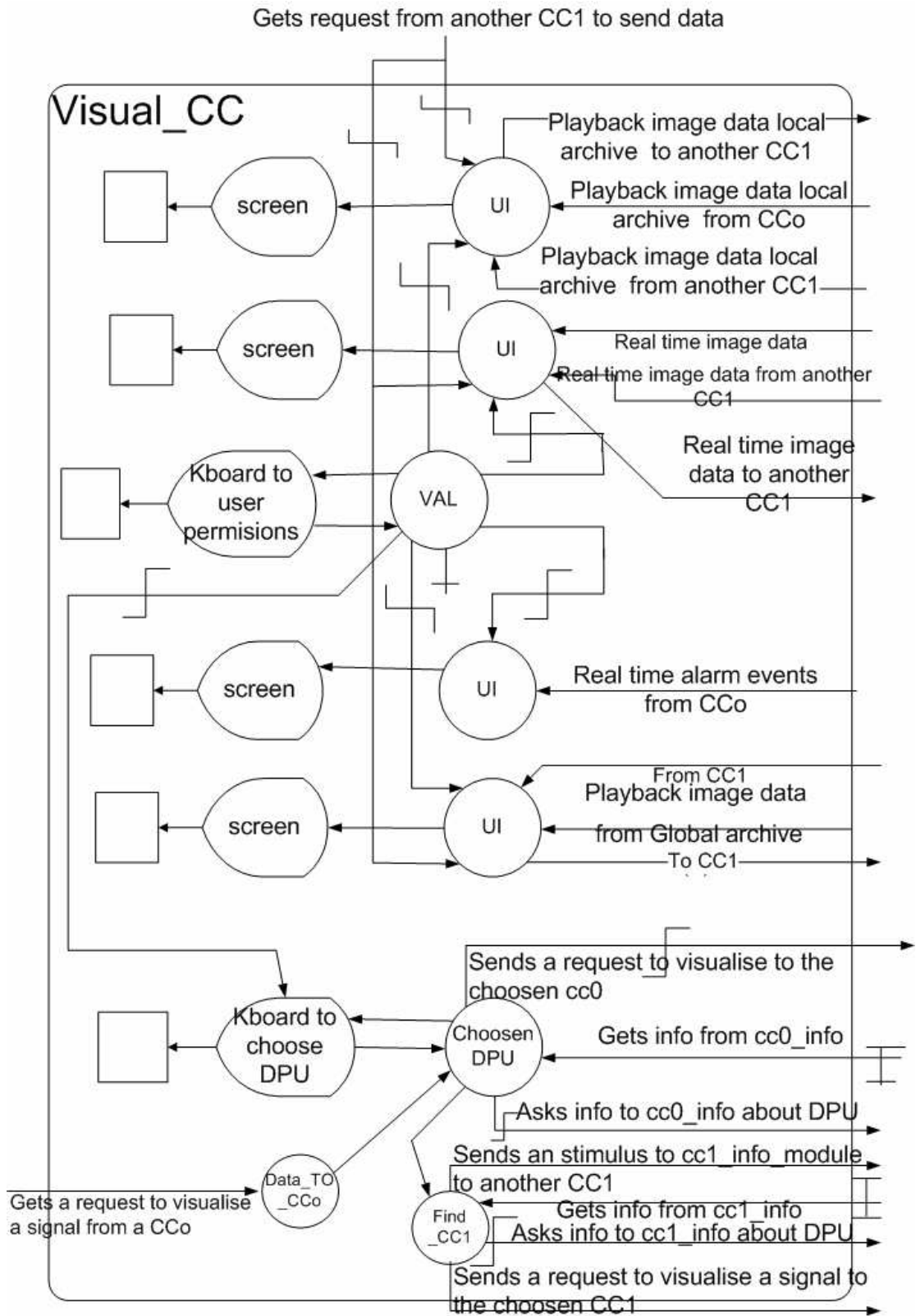


Figure 5-9. The subsystem called Visual_CC. This subsystem is the visual subsystem for any CC1 node.

Figure 5-10 illustrates the subsystem called MOBIL_USER that represents the interaction of the system with a mobile (roaming) user. Any CCx node contains this

subsystem. The subsystems presented above in Figure 5-8 and Figure 5-9, represent the interaction of the system with fixed users e.g. through desktop computers. Besides these subsystems, a user may be monitoring areas outside these fixed monitoring points. Therefore, it should be necessary to allow the system to interact with these mobile users, who may be anywhere in the coverage area of the CCx node. A user connects to the system and then, sends his/her user ID and profile user to Val activity, see Figure 5-10. This activity checks the received information with the information stored in the constant protocol; i.e. the constant protocol has the information of all possible user profiles in the system. Therefore, depending on the type of profile, the user will have one or more access rights. After comparing the inserted data with the persistent data, the Val activity sends a signal to the DA and CIU activities. The DA activity will receive a signal that indicates that the user can or cannot receive alarms of one type or another (e.g. the user may be interested to receive alarms concerning crowd situations but not concerning security situations such as people being in forbidden areas). In the same way, the CIU activity will receive a signal indicating which data the user can visualise.

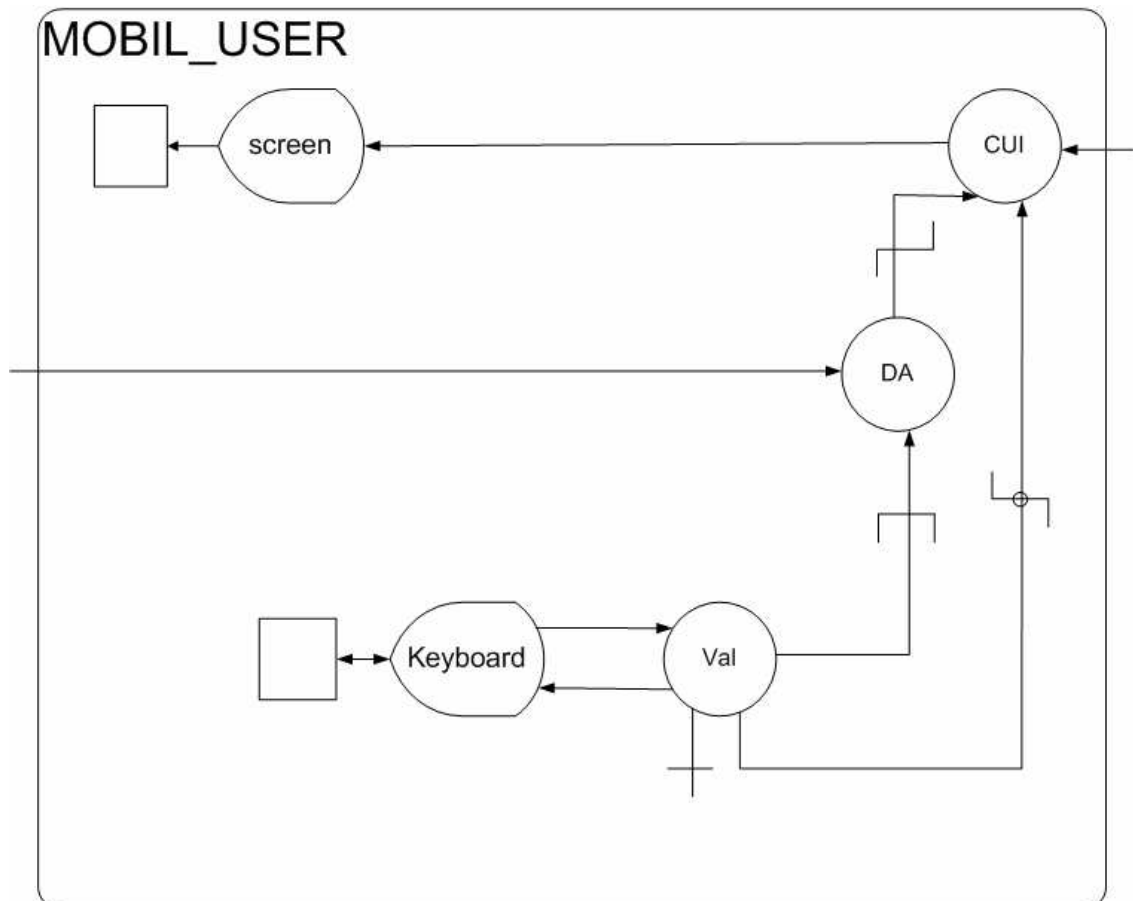


Figure 5-10. The design of a subsystem that represents a visual subsystem for any mobile user.

5.3.3 Data processing part of the design

Although this part of the system deals with all data processing coming from the sensors that are particularised to cameras in this chapter, it does not reduce the generality of the system design as the sensors still can be of different types such as audio or fire detectors. The cameras are represented by servers that the CA subsystem contains, see Figure 5-11. Some of the image processing algorithms that have been taken into account in the design of the system are the ones that are represented in the image data processing flow introduced in chapter 2 (see Figure 2-1). As presented in Figure 5-11, object detection and object recognition algorithms are represented by the OD&OR activity. The tracking algorithm is represented by the TR activity. The behaviour and activities analysis algorithms are represented by the scenario recognition activity (SR). Moreover, the background actualisation algorithm is represented by BU activity.

The rest of the activities that appear in Figure 5-11 do not correspond to image processing algorithms but to the control and also to the analysis of the results coming from these image processing algorithm activities; e.g. if the RA activity gets a signal from the SR activity it means that the SR activity had recognised an alarm event. When RA receives this signal, it checks if it has any control data coming from the CC0 node through the PC_AL activity such as: feedback alarm data, indicating that this kind of alarm event is not an alarm, or new configuration alarm parameters. After checking this data, if the RA activity still considers that this is an alarm event, it sends a control signal to the TI activity allowing it to send (to the VISUAL subsystem in CC0 node) the images that the sensor is capturing while the alarm is occurring. Moreover, the RA activity also sends this data alarm event to the archive component to store it. If TI does not receive a signal from RA, it will not send any data to CC0 node. TI receives the images coming from a SA activity, which only gets the capturing images at the same time that OD&OR activity. The SA activity sends the images to TR and TI; it does not do any processing action in the received images. Therefore, this activity only maintains coherence between the inputs (i.e. the images received from the CA subsystem), from the OD&OR, TR and TI activities. The TR activity, for example, requires the outputs from the OD&OR

activity and also requires the same input image that the OD&OR used to obtain the outputs. On the other hand, if there is any recognised event, the TI activity should get the same input image that the OD&OR and TR activity have used to detect the event.

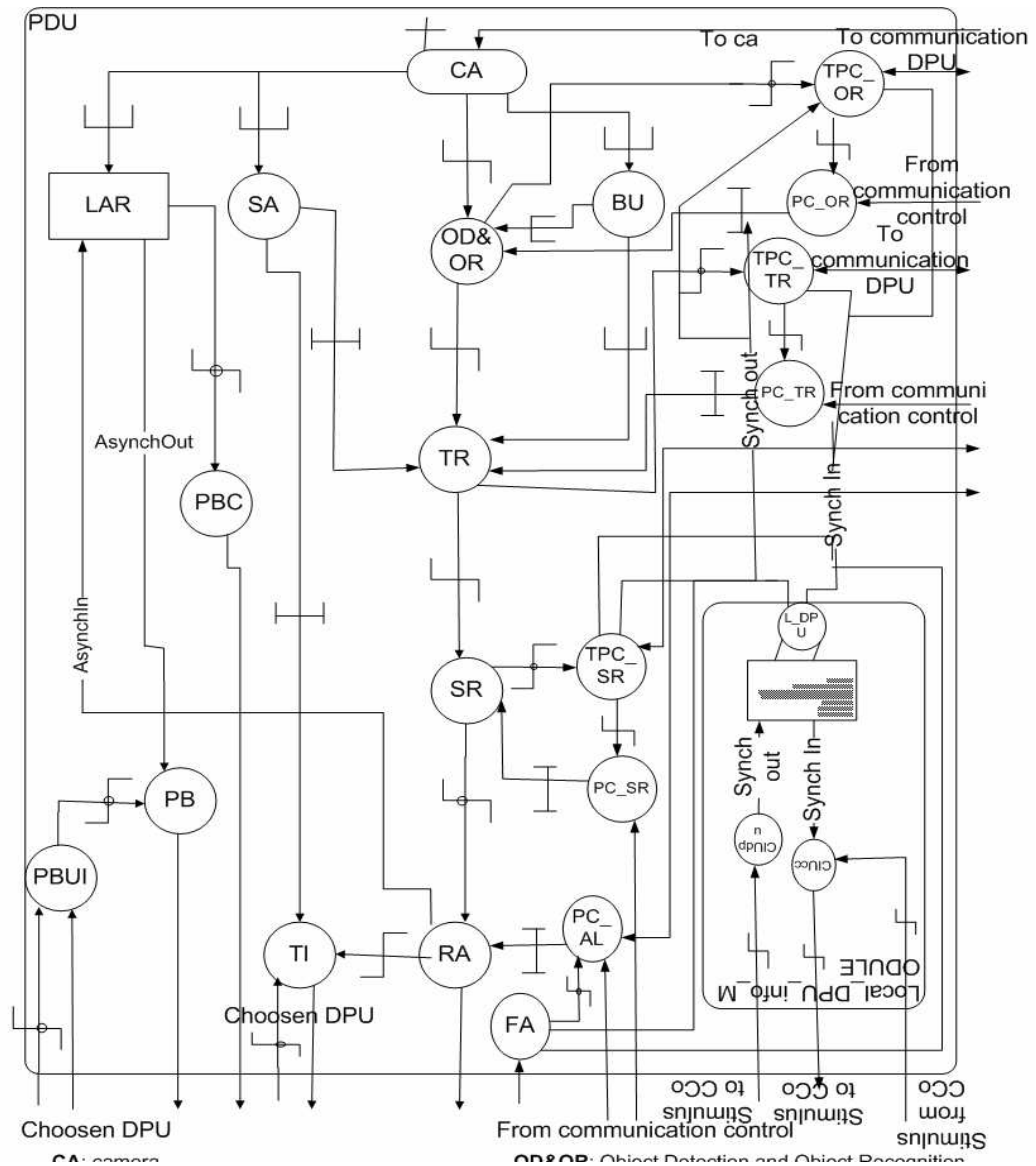


Figure 5-11. DPU subsystem. The sensors are attached to this subsystem through the CA subsystem.

The PB, PBUI and PBC activities in Figure 5-11, manage the archive component as explained above. The PBUI activity receives sporadic control signals coming from a CC0 node. Once the PBUI activity receives one of these control signals, it sends a signal to PB to send archive data to the monitoring subsystem in CC0. The PBC activity controls if the archive component is working properly; e.g. if the archive

component is full, it triggers a signal to PBC indicating that the archive cannot store more data.

The FA, PC_AL, TPC_SR, PC_SR, PC_TR, TPC_TR, TPC_OR and PC_OR activities communicate the image processing algorithm activities such as SR (Scene Recognition), with the control part of the CC0 node or with other DPUs. Therefore, from the CC0 node it is possible to change control parameters related to the algorithms such as thresholds through the PC_OR, PC_TR, PC_SR and FA activities. Moreover, a DPU may also send to the corresponding activity that belongs to another DPU, data that may be important to run the corresponding algorithm more accurately. If two DPUs overlap areas, or their cover area is defined as dependant, i.e. one DPU covers the corridor that connects with a platform and the other DPU covers the platform and they are tracking an object, then it is necessary to provide communication between these two DPUs. This is a very important feature of a distributed surveillance system. Therefore, the TPC_TR, TPC_OR, TPC_SR and PC_AL allow sending information from one DPU to the other (see the bidirectional arrow in these activities in Figure 5-11).

On the other hand, TPC_TR, TPC_OR and TPC_SR store the information received from either the image processing algorithm activities or other DPUs, in the local archive i.e. LOCAL_DPU_info_MODULE (see Figure 5-11). In the case of FA, the information to be stored may come from either the CC0 node or other DPUs.

5.3.4 Feedback part of the design

As mentioned above, the FA activity receives a signal from a CC0 node. This signal is sent by the Alarm Feedback Control subsystem that is inside the CC0 subsystem. **Figure 5-12** presents the Alarm Feedback Control subsystem. The signal called “feedback alarm to DPU” indicates if the alarm, that has been detected by the system and sent back to the user, is considered to be a real alarm or not; i.e. if a user considers the alarm a true or false positive. Therefore, this signal is important because it allows the system to learn through this feedback. If a user considers that this alarm event received is a false positive it sends a feedback signal to the DPU, so that next time when the same conditions happen, the RA activity in a DPU

subsystem (see Figure 5-11) might decide not to send any alarm event. It is also possible that this feedback signal comes from the CC1 node instead of coming from the user in CC0 node (see “feedback alarms from CC1” from **Figure 5-12**).

The Alarm Feedback Control subsystem also sends a signal to the PBUI activity in a DPU (see Figure 5-11). Therefore, if a user wants to monitor data stored in the archive component in a DPU, it sends a signal through the “Kboard Alarms” server component and consequently through the CUI activity to PBUI. As mentioned, if PBUI receives a signal, it then sends another signal forcing the PB activity to start sending the images back to VISUAL subsystem in CC0 (see Figure 5-8). The design of this system takes into account the existence of a global archive data of true positive alarm events per node; i.e. each CC0 subsystem has an archive component that stores the alarms that have been checked out previously by the user as a true positive alarm event. The reason behind this is that, with the same size as the rest of archives, the global archive (called GAR data) is able to store more interesting data, because it only stores the alarm events and not the constant recording data as the local archives do. Therefore, a global archive of an alarms event per zone is designed.

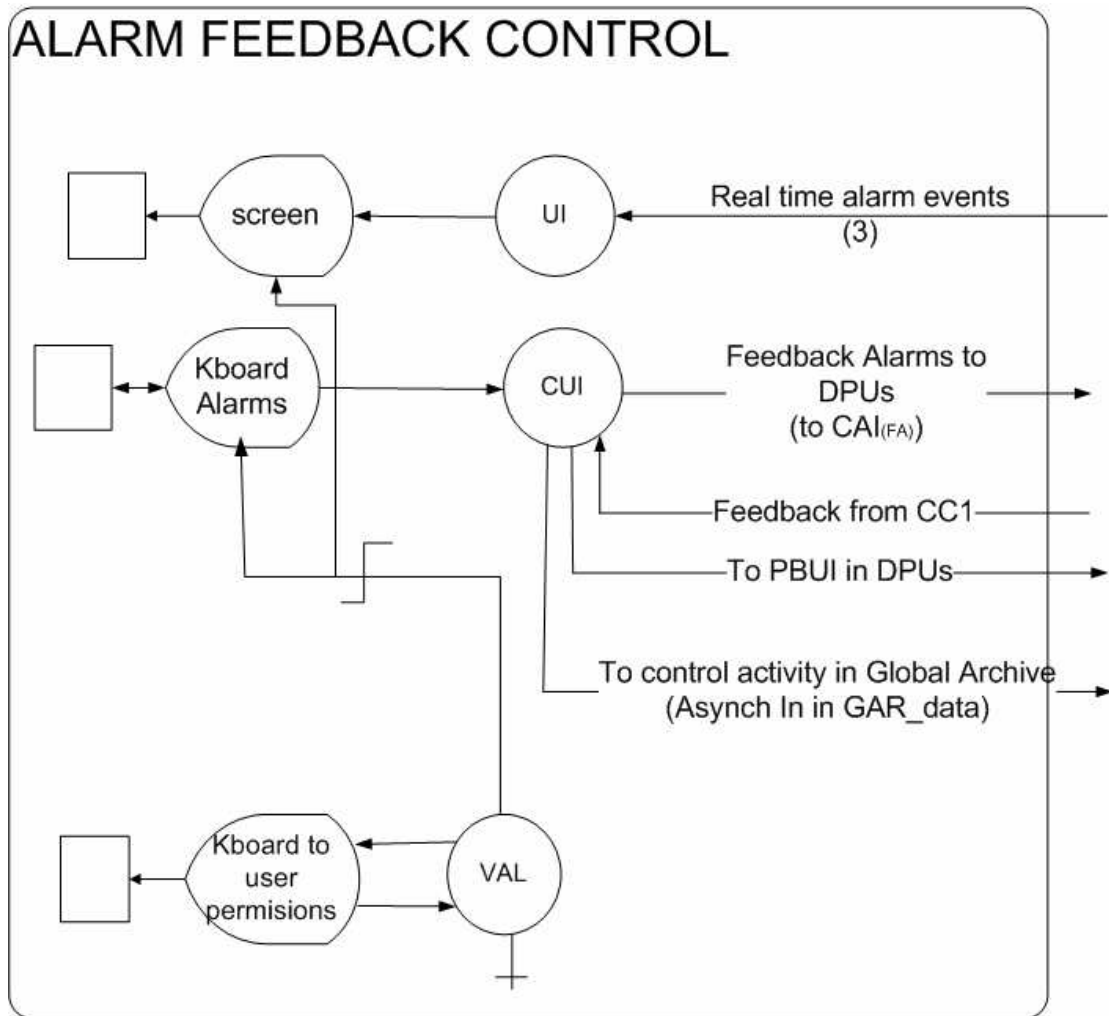


Figure 5-12. The ALARM FEEDBACK CONTROL subsystem that represents a feedback part of the system.

Figure 5-13 presents the ALARM FEEDBACK CONTROL CC1 subsystem, which is contained by a CC1 node. As is the case for the CC0 node, the CC1 node may be alerted by any alarm event that occurs in any of the DPU's under its control. For that reason, a CC1 node is also able to give a feedback on these alarm events, see "feedback alarms from CC1" signal in Figure 5-13. The ALARM FEEDBACK CONTROL CC1 subsystem, through its "Kboard Config_parameters" server component, may also send a configuration parameters control signal to any DPU that is under its control. This signal, called "From CC1 to AddInfoDPU" (see Figure 5-13), sends the information to the archive module explained before, called DPU_info_MODULE (see Figure 5-4), which all CC0 nodes have. These new parameters are archived in the DPU_info_MODULE, and at the same time, sent

back to the corresponding DPU. So, through this subsystem the CC1 node may also change parameters of configuration of any DPU that is under its control.

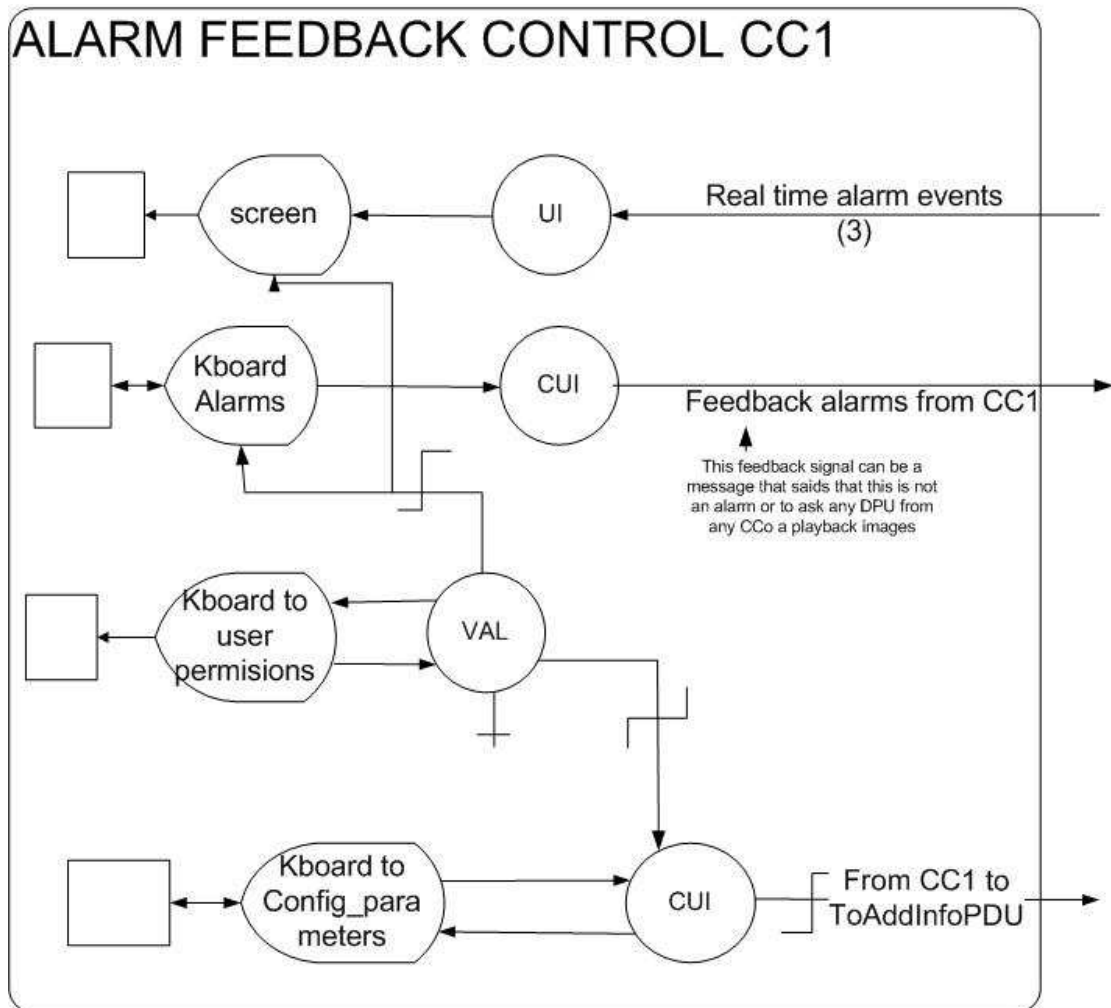


Figure 5-13. The ALARMS FEEDBACK CONTROL CC1 subsystem that represents a feedback part of the system.

5.3.5 Control part of the design

Another important design decision is related to the design of the control parts of the system. By this, we mean the control signals that enable the system to add or change parameters, monitor different outputs of the system and to get information about the whole structure of the system. The control nodes CCx raise most of these control signals. These control signals are different from data signals, in that they are created to activate an action (event) such as to visualise outputs from the system, change or add parameters.

Figure 5-14 presents the CONFIG_MODULE subsystem that deals with the processing of all the signals that are coming from the outside of the system. These signals are generally related to adding or changing parameters of any of the components in the system. Therefore, through this subsystem a user is able to reconfigure some parameters of different parts of the system. A user may change the parameters of the cameras through the “Kboard to control camera” server in Figure 5-14. Moreover, a user may also, for example, want to change the thresholds that some algorithms use through “Kboard to Config_parameters” server. If one of the scene recognition tasks is to detect crowded situations, the SR activity (see Figure 5-11 or the crowd detection algorithm design diagram in Appendix B) applies appropriate thresholds to determine if the scene (i.e. crowded situations) is recognised. Therefore, if the user changes these parameters the SR activity has to change the thresholds to the new ones to recognise the scene according to the new parameters.

Another parameter that a user may want to change or add is the location of a DPU through the “Kboard to info location” server in Figure 5-14. Note that, all these changes alter persistent data, so that the Config_module subsystem sends these parameters to the corresponding DPU and camera, and at the same time it stores the data in the DPU_info _MODULE (see Figure 5-4 and Figure 5-14). The ToCC1 activity that appears in Figure 5-14, gets the data sent by the DPU_info_ MODULE when it gets a control signal from a CC1 node asking for a DPU’s information, as explained before.

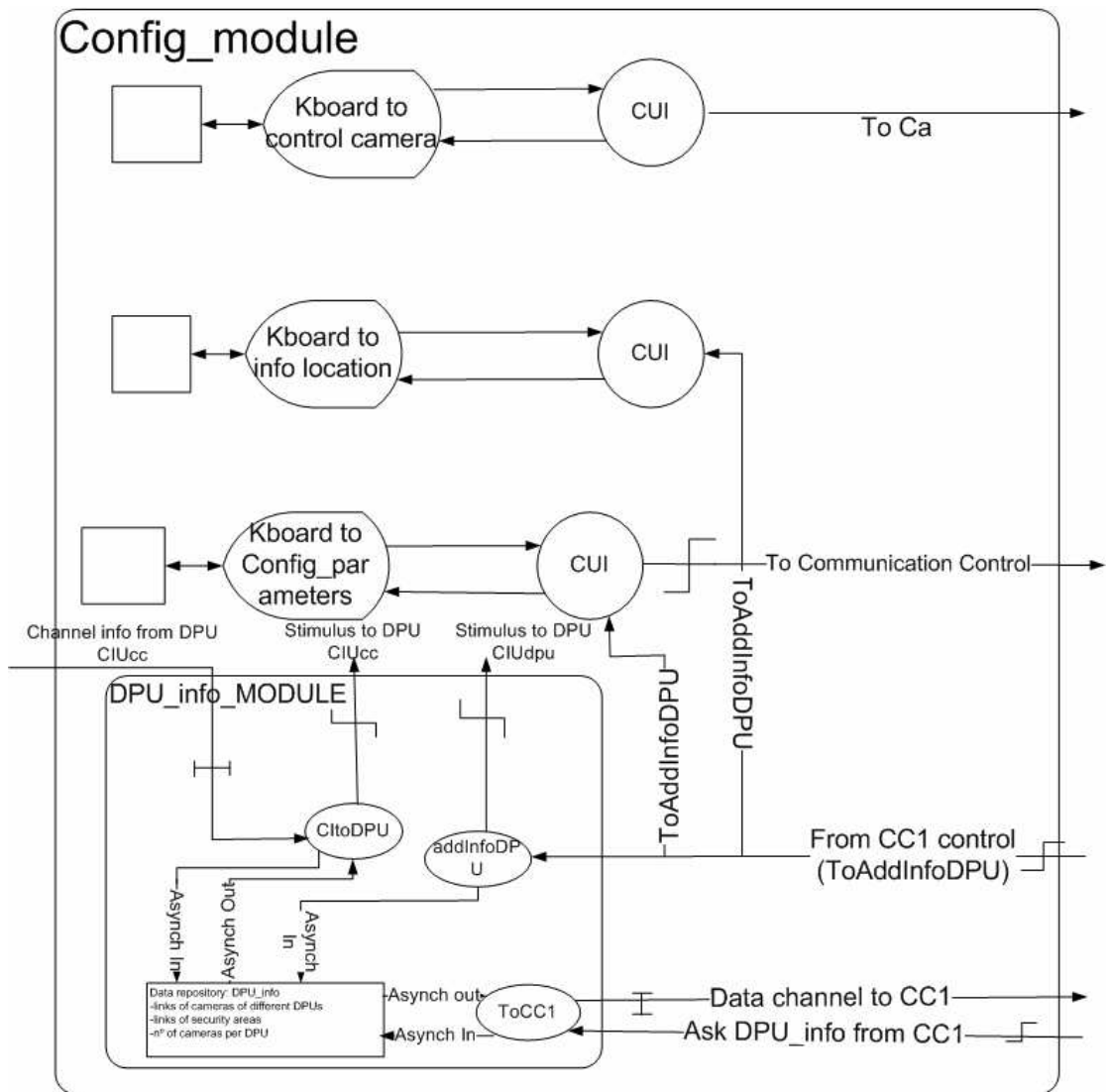


Figure 5-14. Design of the config_module used by the CCo to change configuration parameters.

Figure 5-15 presents the CC0 subsystem that consists of a global archive component, three subsystems, a server and two activities. The activity called CAR connects the global archive component called GAR data with a “screen” server component, so that if the archive triggers a signal (e.g. “the archive is full”) then the user may be warned through the server. The three subsystems as seen in Figure 5-15 are: the VISUAL subsystem and the Locator_user and MU subsystems. The MU subsystem groups two subsystems as explained in section 5.3: the Config_module and the ALARM FEEDBACK CONTROL. Therefore, the MU subsystem does not have any functionality other than to group these two subsystems that have a related control functionality. When the lookup activity, (see Figure 5-15), receives a sporadic signal from the RA activity in a DPU (see Figure 5-11), consisting of an alarm event data, the lookup activity sends the alarm to the Locator_user subsystem.

When Locator_user receives this type of signal, it searches if there is any mobile user connected to the system that may be interested in being warned with this alarm event. If there is any such user, then the Locator_user sends them the alarm event data.

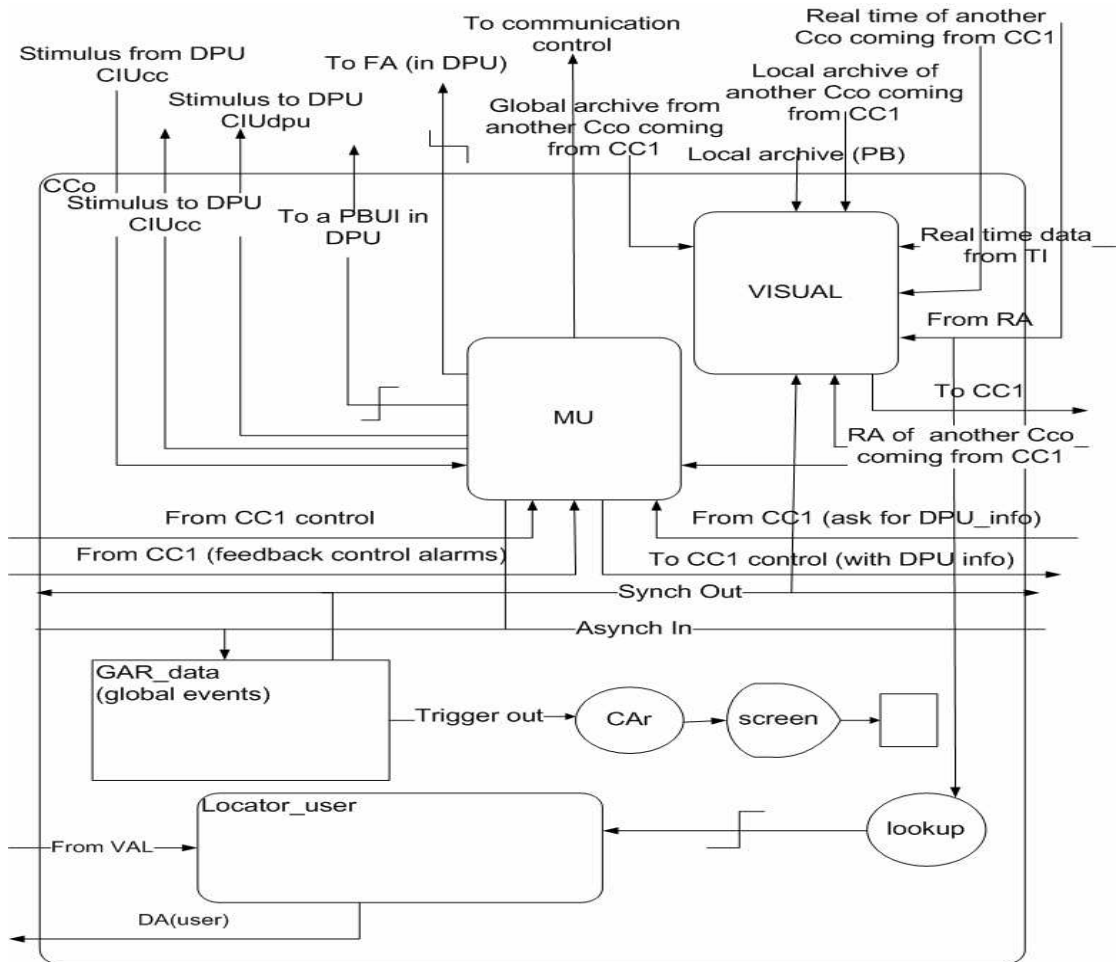


Figure 5-15. The CC0 subsystem, which represents the first level of hierarchical control structure of the system.

Figure 5-16 illustrates the design of CC1 subsystem, which corresponds to the outmost level of the hierarchical network structure. Nevertheless, as mentioned earlier, if the system needs to be scaled up to another level of hierarchical control, subsequent CCx subsystems will have the same architecture design, as the CC1 subsystem. However, the subsystems will need to be called differently because they will interact with different level of nodes (see Figure 5-17). For example, the subsystem called “CC1_info_module” will be changed to “CC2_info_module” subsystem. These changes occur because the CC2 node will communicate with CC2

and CC1 nodes directly, instead of communicating with CC1 and CC0 directly as CC1 does. Moreover, if another level like CC2 is introduced, then the CC1 node will undergo some minor changes. Changes that should be introduced in CC1 in such a case are illustrated in red in Figure 5-18.

The CC1 subsystem consists of four other subsystems and two activities. The subsystems, which have been introduced in previous sections, are the following: CCo_info_MODULE, VISUAL_CC, CC1_info_MODULE, Locator_User and ALARM_FEEDBACK_CONTROL_CC1. The look up activity in Figure 5-16 has exactly the same functionality as the look up activity in Figure 5-15, which has been also introduced. The activity called “DATA_TO_CCo” sends (to another CC0) the data that is required by one of these CC0s. When “DATA_TO_CCo” receives a request from a CC0, it sends a signal to Chosen_DPU activity to look for the DPU. Once the DPU is found and the CC1_module subsystem starts to receive the data, the CC1_module subsystem sends the required data back to the CC0 that asked for it.

The functionality of a CC1 subsystem is to get the information about the number of connected CC0 subsystems, the number of DPUs connected to each CC0, and also the number of cameras connected to each DPU. This information is used by the system to get the whole structure of the system. Therefore, a user connected to e.g. a CC1 subsystem may monitor data from any CC0 and consequently, from any DPU node. The user may also change information of any DPU or give feedback about an alarm event.

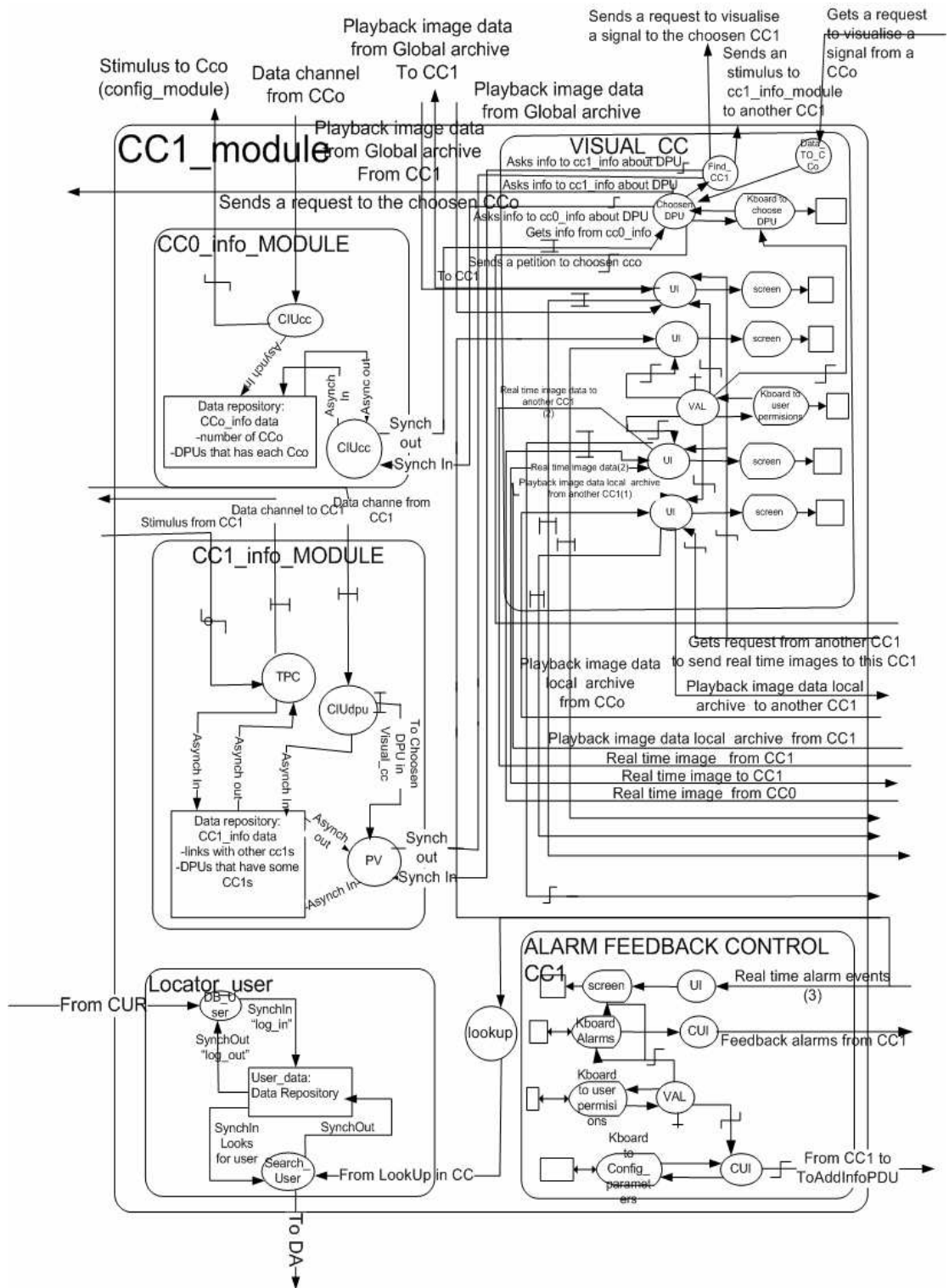


Figure 5-16. The CC1 subsystem, which represents the second level of the hierarchical structure of the system.

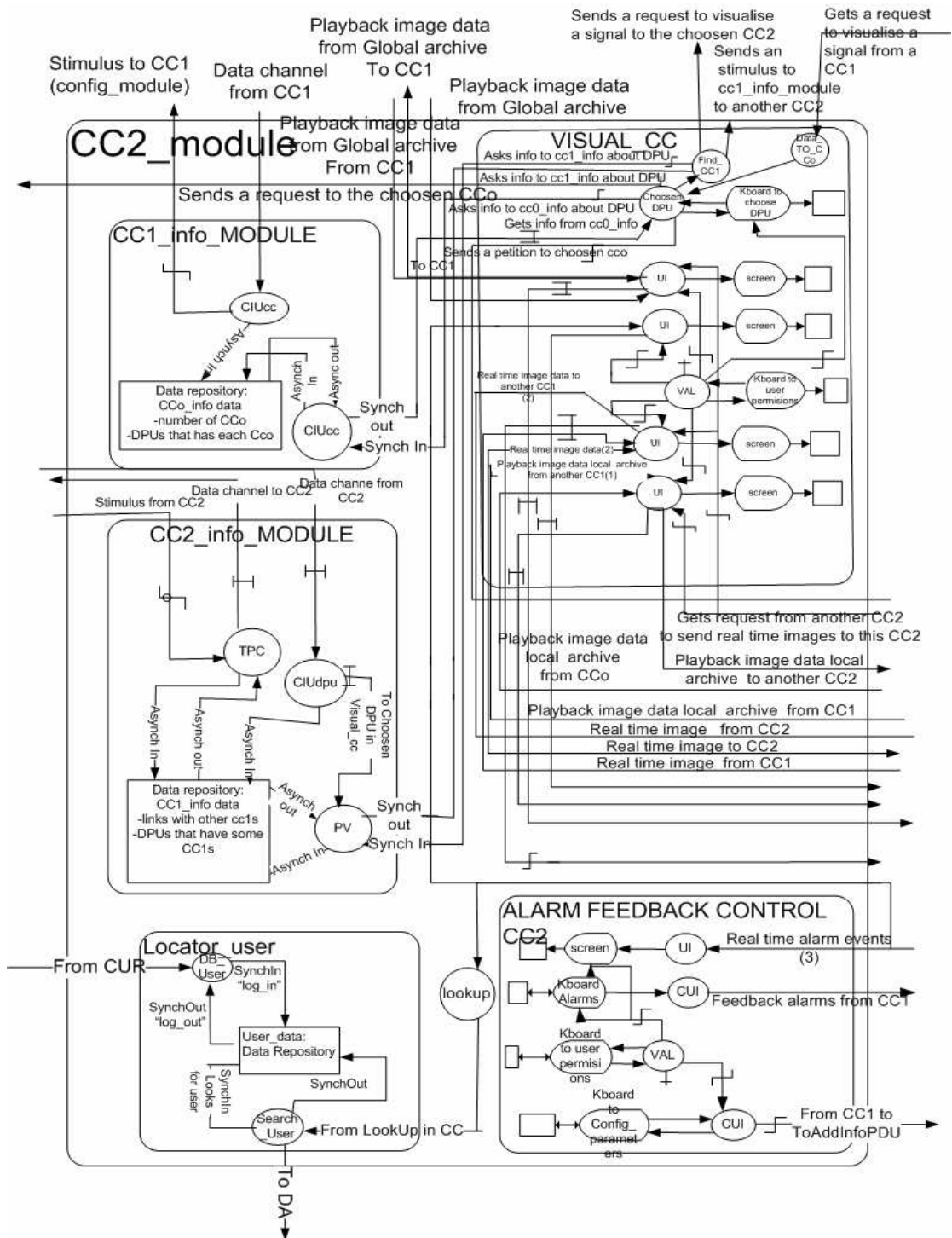


Figure 5-17. The CC2 subsystem design if the system was scaled one more level.

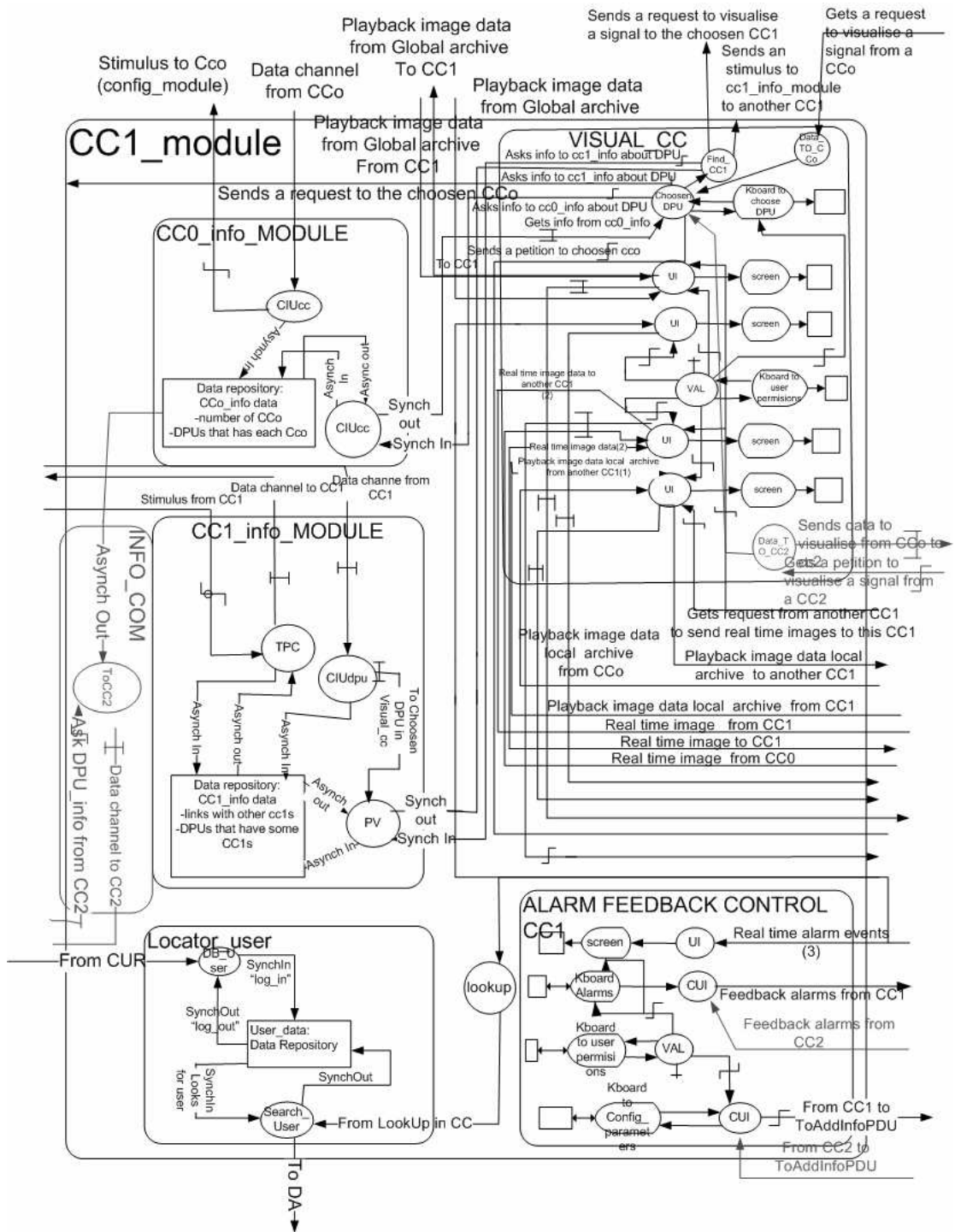


Figure 5-18. The changes that should be applied to CC1 (shown in red) if a CC2 subsystem is introduced in the design of the system.

5.4 Design of the system

In section 5.3, the different functional parts of the system have been described. Appendix D presents the design of the different components using the MADGE graphical representation tool. Note that, although we tried to preserve in the system

design the names of the components presented in previous sections, some of the names of the components have been changed. To make the presentation clearer, the design illustrated in Appendix D, corresponds to only one leaf of the whole network design structure illustrated in Figure 5-1 (see Figure 5-19). Therefore, the design presented in this section consists of: one CC1 subsystem, one CC0 subsystem, one communication subsystem between CC1 and CC0, one communication subsystem between CC1s, one DPUs, one communication subsystem between DPU and CC0, and one communication subsystem between DPUs and a mobile user subsystem.

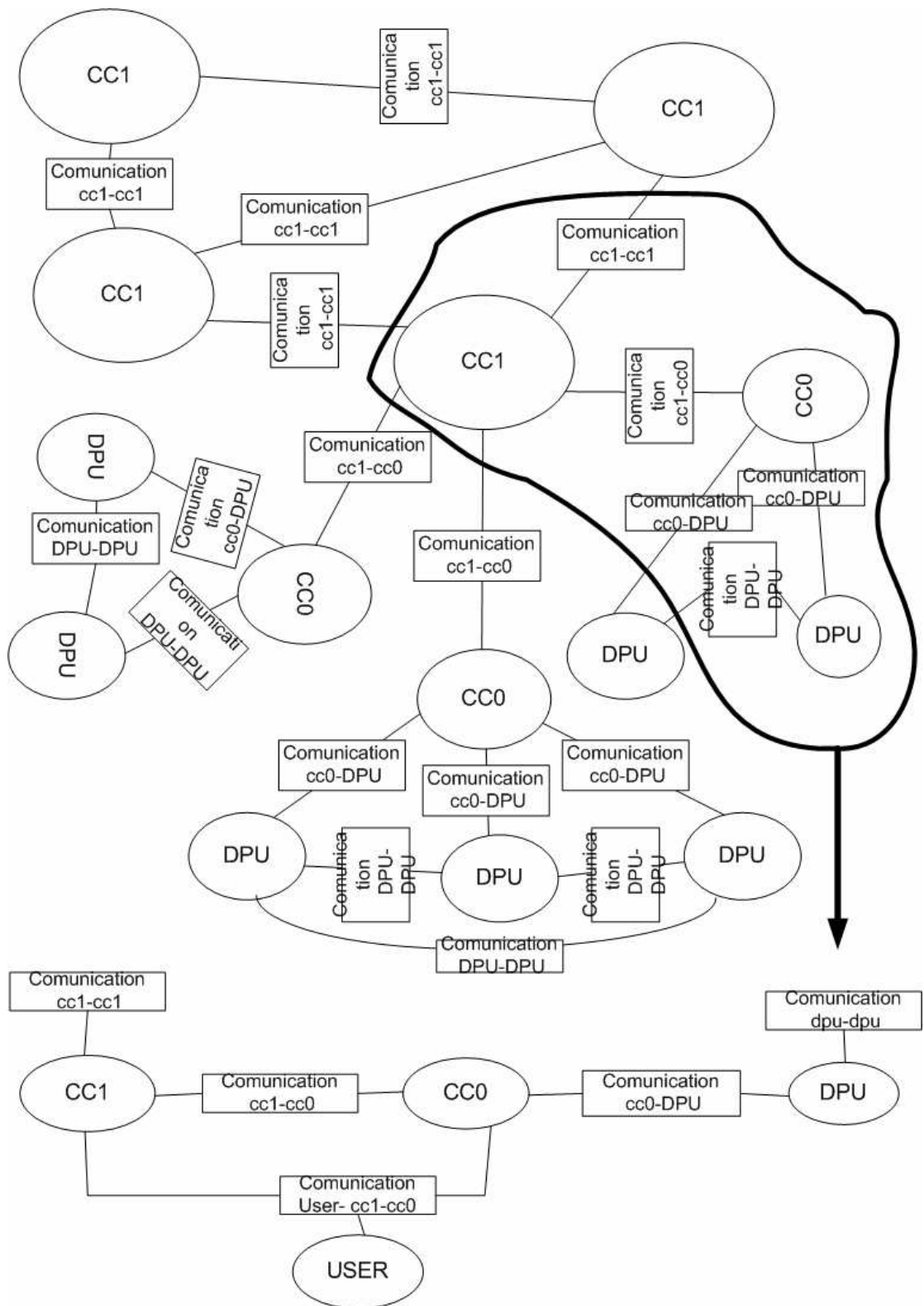


Figure 5-19. Design represented using MADGE tools in Appendix D.

Note that one design decision is to communicate DPU subsystems, CC1 subsystems but not CC0 subsystems. The network structure that is created from this design decision is discussed in section 5.5. The reason for that design decision comes from a compromise between scalability and performance. With this design, depending on

where a user is connected, performance may slow down. If the user is connected to a CC0 node and asks for data corresponding to a DPU that is connected directly to this CC0, the user will get the data faster. However, if the user connects to a CC0 node and asks for data from a DPU that is not connected to the CC0 node directly, the user will not get the data as fast; because the data is received through a CC1 node rather than through the specific CC0 node. Therefore, by introducing this restriction on the communication between CC0 nodes, the monitoring of the zone assigned to the user has priority over that of other zones.

On the other hand, with this design, it should be quite straightforward to scale the system up by increasing the number of CC0 without considerably decreasing the performance of the system from a network point of view. The bandwidth does not decrease dramatically and the routing policies are easier, because communication only occurs between CC1 and CC0, and not between CC0s. This does not occur when another DPU is introduced. Bandwidth allocation decreases significantly by overloading the network with packets through different paths. Moreover, the routing policies may get more complex.

5.4.1 Partitioning

In sections 5.3 and 5.4, the functional definition and the system design have been presented and discussed. Note that the system is partitioned into multiple subsystems that consist of other subsystems, at the same time. Through each subsection, parts of the system have been presented and decomposed into a network of activities communicating through protocols. As mentioned, the selection of each protocol has been done according to the needed interaction between the activities. Once the system is decomposed as a network of activities, the design phase of the system may be considered finished but not over, because the design in DORIS is an iterative process.

Therefore, the design can undergo some changes at any time in the system creation process. Note that the phases introduced in the next paragraphs do not belong to the design part of the system, but they are introduced to give a logical continuity to the creation of the system.

Once the design phase finishes, the implementation phase of the system starts. For example, this would include the implementation of the image algorithms into the corresponding activities, the implementation of the control communication algorithms, the implementation of algorithms to archive and to access to the storage data, and the implementation of the communication mechanisms of the IDAs. Moreover, the choice of the scheduling strategies is also included in this phase.

Once all the activities and communication areas are implemented, the mapping phase process starts. By knowing the number of resources such as CPUs and memory available, the physical mapping of the network of activities to the physical resources starts and a prototype can be built. By using the DORIS methodology, the system may be partitioned into several independent modules that communicate between them through designated areas. Therefore, it is possible to prototype and test each module on its own.

In the next section a discussion of the design of the system is presented, but this time from a network topology design point of view, instead of a software network design point of view.

5.5 *Network Design of the system*

This section discusses the proposed system design from the logical network design topology point of view. At the time of writing, there are three main types of architectural design network models: flat networks, hierarchical networks and mesh networks. Network topologies such as bus, star, tree, star-wired ring or Fiber Distributed Data Interface (FDDI) can be classified in one of these architectural design models; e.g. the FDDI can be classified as a flat network.

Each of these architectural design models has its pros and cons. Flat networks are adequate for small networks; each node has the same functionality and the network is not divided into layers. Flat networks are easy to design, implement, and maintain as long as they remain small. However, changes in this type of network tend to have a high-impact in the network itself. Mesh designs are recommended to meet

availability requirements. There are two types of mesh topologies: full-mesh, where all nodes are connected to all nodes, and partial-mesh, which has fewer connections between nodes. Full-mesh gives complete redundancy to the design and also gives good performance, because there is only one single-link delay between nodes. The same does not occur with partial-mesh architectures, because a node may traverse more than one link to reach the other node. Even though mesh topologies give good reliability, they may be expensive to design and maintain. Moreover, this type of network may be difficult to troubleshoot, scale and optimise; it is possible to overload the network with packets through different paths. As pointed out in [Macmillan Technical Publishing and Cisco Systems 1998], with mesh topologies it may be also difficult to contain network problems, because of the lack of modularity. On the other hand, hierarchical architectures divide the network into layers or modules. Therefore, hierarchical designs impose a modular design helping with control management. Moreover, this type of architecture is more scalable and flexible, because it allows the creation of design structures that can be replicated as the networks grows; each instance of the module is consistent, and the expansion is easy to plan and implement.

There is a rule of thumb to design network systems presented in different CISCO books such as [Paquet and Teare 2001] and [Macmillan Technical Publishing and Cisco Systems 1998]. The network should be designed following a hierarchical architecture. CISCO design methodology based on simplicity, suggests that the design does not require more than three layers that following CISCO terminology, are called: access layer, distribution layer and core layer. Once each layer is designed using modular and hierarchical techniques, the following step is to design the intercommunication layers based on the analysis of traffic load, flow, and behaviour. After completing the logical topology, the logical design phase continues by designing network addressing and naming models, selecting routers and bridging protocols, and developing network management and security areas.

5.5.1 Logical design topology

As mentioned in section 5.2.1, the final logical topology of the system design presented, is a hybrid of these three architecture design models. Please see Figure

5-1 and Figure 5-21. Before arriving at the final topology, others topologies were proposed as illustrated by Figure 5-20. Once the functional definition of the system was defined into data processing units and monitoring units, a first proposed network topology was to connect the monitoring units using a FDDI ring topology and then, to attach the data processing units to the monitoring units as shown in the top design topology in Figure 5-20. The main problems with this topology is the lack of scalability and the lack of distribution, because only one node (CC in Figure 5-20) communicates with the rest of CC0, therefore a single point-of-failure was created. After disregarding this topology, a second topology was proposed based on a hierarchical design model as illustrated by the bottom topology in Figure 5-20; the reasons for the election of a hierarchical architecture model are explained in section 5.5. Nevertheless, to scale this proposed topology, new upper level nodes should be designed and created; adding more complexity into the structure and into the modules that constitute the system.

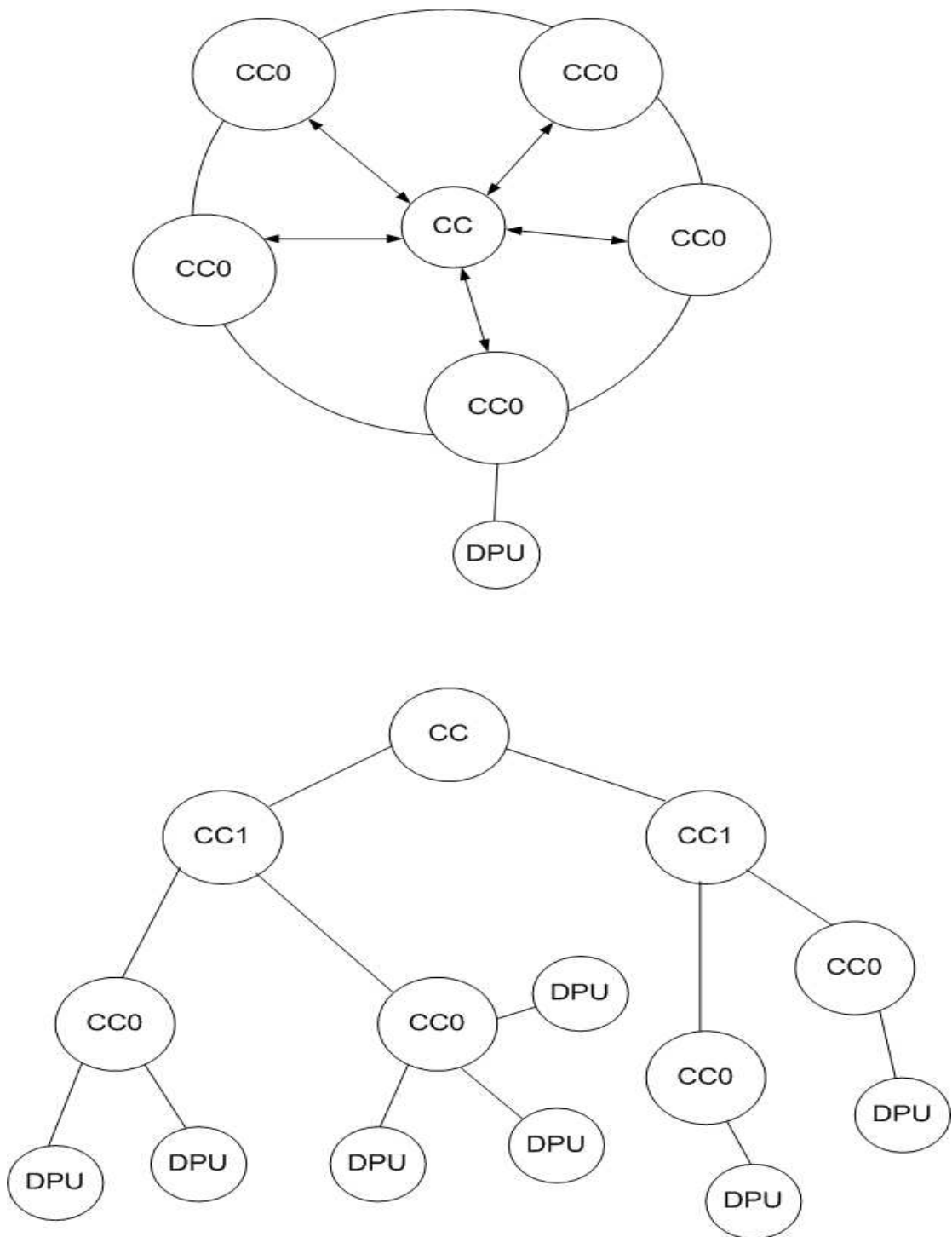


Figure 5-20. Two previous candidate topologies of the system design, before the final topology.

The final structure of the system, following the second proposed topology, is also based on a hierarchical structure, but divided into three layers. The upper layer corresponds to the CC1 nodes, the second layer is represented by CC0 and the lower layer is represented by the DPUs. In reference to the CISCO design methodology, DPUs may correspond to the local access layer, where services like

multicasting are established. A partial-mesh architecture is used in this layer, because the traffic between nodes is higher and therefore, it is better to have a redundant topology; DPU nodes need to communicate between them to get more accurate results. CC0 nodes may represent the distribution layer; each CC0 groups a number of DPUs. Therefore, a user is able to monitor, from each CC0, this group of DPUs. The CC0 nodes use a flat network architecture i.e. star-wired ring architecture because, as has been discussed, there is no communication between CC0 nodes. Finally, CC1 nodes may correspond to the core layer. The architecture used is also partial-mesh, because it needs to be redundant. These nodes allow the communication of any CC1 with any DPU in the system.

The traffic that begins in any DPU is only allowed to be forwarded to the upper levels (i.e. CC0) if and only if it meets the following criteria: if an alarm event occurs in the lower level or if data is required from upper levels such CC0 or CC1. This design decision comes from the suggestion made in [Paquet and Teare 2001] that says that traffic that begins in a lower layer of the hierarchy should be only allowed to be forwarded through the upper levels if it meets defined criteria.

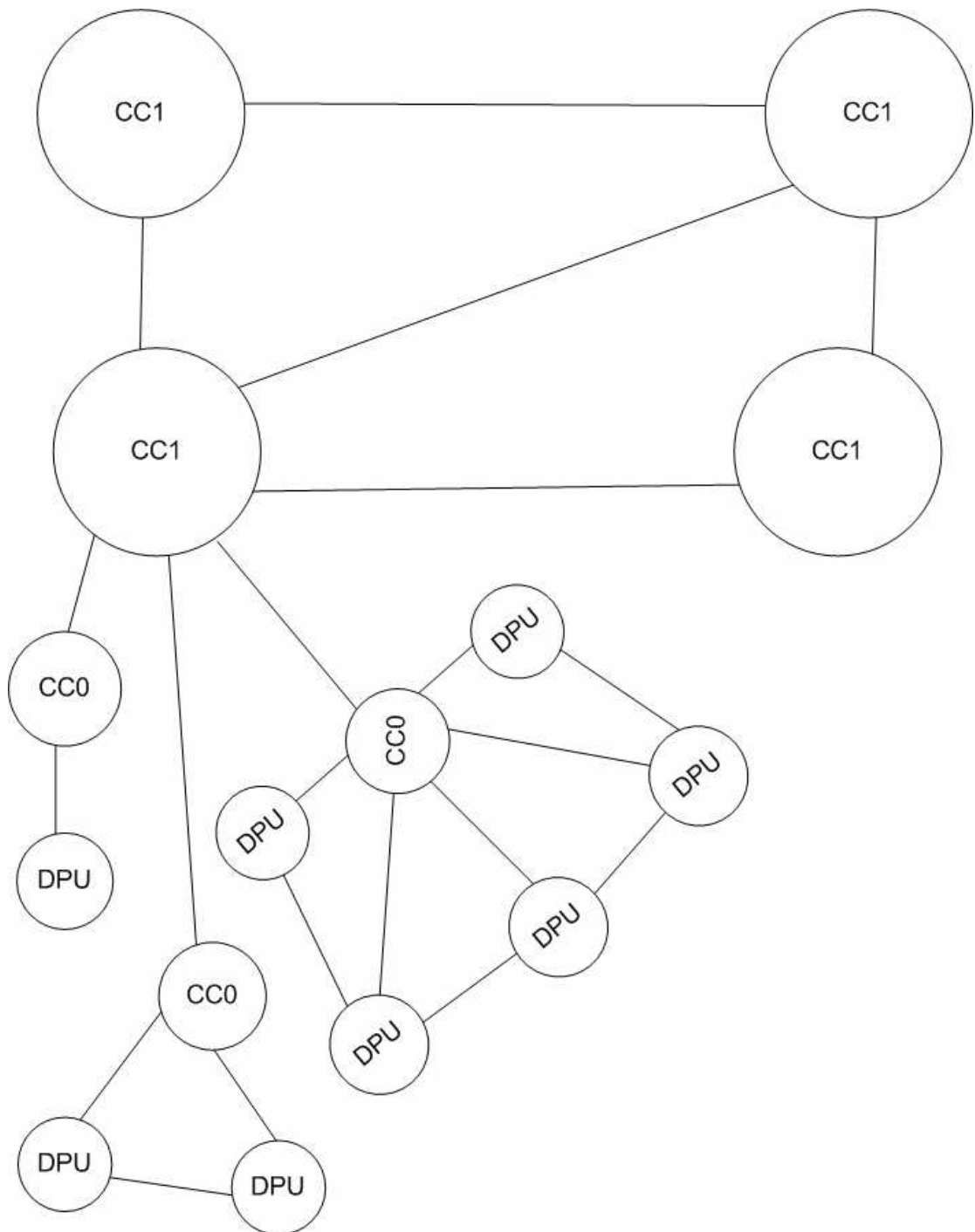


Figure 5-21. The final logical design network topology of the system.

5.5.2 Traffic behaviour- Multicasting

In terms of network design, it is important to characterise the behaviour of the network to plan the network level and its expansion, to quantify the network performance and to be able to verify network services. Therefore, to analyse the behaviour of the network, the traffic flow, the traffic load and also the traffic behaviour should be determined. To characterise the traffic flow is to specify the

type of traffic flow that the system supports such as client/server traffic flow, peer-to-peer, server/server or distributed traffic flow. To characterise the traffic load means to study the number of stations that constitutes the network, the average time that a station is idle between frames, and the time required to transmit a message once the medium access is gained. Finally, to characterise the traffic behaviour includes the analysis of broadcast traffic, i.e. a message that is sent to everybody, or multicast traffic in the network, and the analysis of the network efficiency with the study of frame size, protocol interactions, window and flow control and error-recovery mechanisms.

Here, the behaviour of the network has been analysed in terms of traffic flow and traffic behaviour. The traffic flow of the system design presented has been characterised as distributed traffic because the communication between nodes is modelled as a non-symmetric communication between independent nodes with the same importance role.

From a network perspective, a multicast behaviour dramatically reduces the overall bandwidth consumption as pointed out in [Macmillan Technical Publishing and Cisco Systems 1998] and allows more scalable network topologies solutions, because it allows organising the traffic per groups. In surveillance systems, multicast traffic is constantly used. For example, inputs captured by the sensors are sent simultaneously to many modules and then the corresponding processed data retransmitted, also simultaneously, to other parts of the system.

In the system design presented here, multicast communication is represented by the IDAs components that have a window input where the data is put and then the data is distributed to different components through different windows-outputs. See Appendix D. IDAs can have the mechanisms to be able to replicate and distribute the data to more than one component at the same time.

5.6 Quality of Service (QoS)

As seen in this chapter, and also reported in many articles mentioned in chapter 2, surveillance systems usually require the monitoring of data coming from remote

units such as sensors. Therefore, transmission becomes an issue; e.g. transmission media selection, security in the transmission and Quality of Service (QoS). Even though in this chapter transmission media and security in the transmission are not discussed, in [Mähönen and Saaranen 2000] the authors list, as possible transmission media for surveillance systems, the following technologies: microwave, ISDN, ATM, optical fiber, broadband networks and wireless networks.

Quality of service in surveillance systems can be defined as the ability of the network management to distinguish various actions and to assign different levels of quality and transmission guarantees to these actions. For example, the QoS when a CC0 node requires to visualise data from a DPU that is connected directly to it, should be better than the QoS when the required data comes from a DPU that is not connected directly to the CC0. The parameters that specify the QoS are usually: bandwidth, latency and loss rate. Latency is defined as the accumulated delay between the start of a transmission of a message from one process and the beginning of its reception of this message by another process [Coulouris et al. 2001]. When the network delivers with a variable latency, this is called jitter. Surveillance systems should be designed to minimise jitter.

Quality of Service management is based on organising the allocation and scheduling of resources to meet the requirements. Therefore, the allocation of processing capacity, network bandwidth, and also the allocation of memory for buffering data are important to obtain the required QoS. As mentioned, by using an RTN methodology, the final network system design allows controlling the allocation of the processing capacities, i.e. the activities needed, and also allows controlling the memory space required even in the communication between the activities, through the communication protocols. Therefore, by deploying RTN to the system design, it is possible to provide more guarantees to obtain the QoS required.

5.6.1 Bandwidth

As mentioned in the previous section, bandwidth is one of the critical parameters that QoS depends on. In [Mähönen and Saaranen 2000] the authors illustrate the

effects on bandwidth in a Local Area Network (LAN) and in a Wide Area Network (WAN) depending on the multimedia application; e.g. the bandwidth required to transmit standard TV video uncompressed is around 120Mbps. Therefore, it is important for the network design of the system, to apply policies that take into account bandwidth requirements such as: bandwidth reservation, quality of service negotiation, compressed algorithms to reduce data transmission or the use of protocols like Resource Reservation Protocol (RSVP) or Real-Time Transport Protocol (RTP). In terms of network design topology, switch elements can be used to guarantee high-bandwidth requirements. Hubs, which shared media access, may be used to guarantee an inexpensive access but they do not guarantee high-bandwidth requirements. Finally, Router elements may be used to isolate broadcast traffic, and therefore they may be good to control bandwidth requirements.

Although the use of compressed algorithms works to reduce the bandwidth requirements, it increases the load on the processing resources because these algorithms usually are quite demanding on computing resources. As mentioned, the tendency in surveillance systems is to use specialised hardware such as DSPs embedded on or near the camera to compute these algorithms, or to use software such as codecs/decoders allocated on the CPU where all the processing is done. The system design in terms of software architecture, presented in this chapter, allocates these software algorithms in the activity attached to the server that is connected to a camera device in the subsystem called CA. See Appendix D. As said, the actual physical allocation of each activity, in DORIS, is left to the next phase of the design of the system; i.e. the physical mapping of the RTN network design.

5.6.2 Resource management- Scheduling

In surveillance systems (as in any multimedia application), each process must be allocated adequate CPU time, memory capacity and network bandwidth to perform its designed task and must be scheduled to use the resources frequently enough to enable it to deliver the data to the corresponding process on time. Surveillance systems have to handle both discrete (e.g. alarm events) and continuous data (e.g. images captured in real-time). It becomes a challenge to provide sufficient service to time-dependent data streams without causing starvation of discrete-media.

Therefore, scheduling policies need to be applied to all resources that may affect the performance of the system.

As mentioned in chapter 4, in RTN the choice of the scheduling policies is not imposed, even though it requires documenting the reason of the selected strategy. In [IECCA and MUF 1987c], the selection of the scheduling strategy is based on the idea of optimising the response to external events for a given amount of processing power. Several real-time scheduling algorithms have been developed to meet CPU scheduling needs for the applications. The priority-based pre-emptive algorithm is the most common scheduler used in commercial real-time operating systems. As mentioned in chapter 4, a MASCOT kernel machine usually uses a co-operative scheduler. Each process has assigned a CPU time slot with a co-operative scheduler, so as to ensure that each process will complete the task on time. However, if an interrupt occurs, the scheduler ensures that there is no re-schedule once the interrupt is handled; thus control returns to the process that had it before the interruption. Co-operative scheduling may then limit the ability of the scheduler to optimise the response. This issue is avoided with a priority-based pre-emptive policy, because if an interrupt occurs, the scheduler has the option to be re-scheduled. The scheduler then has the ability to optimise the response of the system to the external event. On the other hand, with priority-based pre-emptive policy a priority inversion may occur and provoke a failure of the system as illustrated in [Kalinsky and Barr 2002]. The priority inversion is a scenario where the high-priority task fails to run when it should. Therefore, the choice of one of the schedule policies implies a compromise between performance and fault tolerance properties.

In [Coulouris et al. 2001], the authors introduced several scheduling policies that are suitable for multimedia applications, where they also state that traditional real-time algorithms are suitable for continuous data stream multimedia applications. The Earliest-Deadline-First (EDF) scheduler uses a deadline associated with the task, to determine which tasks should be processed next. The same authors say that EDF scheduling is proven to be optimal for allocating a single resource based on timing criteria. Nevertheless, EDF requires a scheduling decision for each message; to make the scheduling policy last longer. Alternatively, in the Rate Monotonic

(RM) scheduler, which is a real-time scheduling algorithm for periodic processes, messages are assigned priorities according to their rate.

Therefore, real-time scheduling algorithms should be adjusted to distinguish between time-critical and non-critical tasks to cope with bursty real-time traffic, which is characteristic in surveillance systems. By designing the system as a network of activities, and then, analysing which type of task each one of them is (critical or non-critical), it is possible to select the schedule strategy that suits the best.

In terms of network design, there are some protocols such as RSVP, that prioritise traffic by applying “fair” scheduling policies to the network such as: priority queuing, custom queuing, weighted fair queuing, custom queuing or low-latency queuing (LLQ). These methods may be applied in the core layer, presented in section 5.5.1. They are used to give critical data priority over less critical data transmission during peak traffic conditions.

5.7 Summary

In the first part of this chapter, a proposed architectural model for a large-scale distributed real-time surveillance system has been presented. This chapter tried to focus on the idea that the architectural model of a distributed system is concerned with the placement of its parts and the relationships between them, which have been discussed through sections 5.2 and 5.3. The architectural model determines not only the appearance of the system but its structure, providing a consistent frame of reference for the design, which has been presented in sections 5.4. Because the proposed design solution is for a large-scale system, the solution tries to be as modular and generic as possible to allow easy scalability and management of the system.

Note that, this chapter tried to focus the discussion on the logical design phase, because it is one of the most important phases to the creation of the system. A clear definition of proposal of a system design can ease the transition into the design of the physical implementation of the system. In section 5.4.1 descriptions of the

phases that follow the design process have been mentioned. Apart from the architecture design model of system in terms of software structure, this chapter, through sections 5.5 and 5.6, presented and discussed a possible network topology of the system, coherent with the software design structure proposed. The reason for that has been, to give a hint of how the proposed RTN solution may be mapped to a real solution.

Through the presentation of the proposed architecture design of a generic surveillance system, this chapter tended to concentrate in one of the main ideas for the whole work, namely that although image processing algorithms are crucial in surveillance applications, there are other important parts of the design of these systems that need to be also designed and discussed. Without a design methodology that guides the designers to design each part step by step and understand the whole system, it is not possible to build a system with such characteristics. Even more, it is not possible to control and manage the system without having a global picture of the system and knowing what the system actually does. Therefore, it is crucial to apply a design methodology to the creation of a system, apart from the creation of its vision algorithms. Moreover, by using a design methodology, it is possible to define what specific activities may be of interest in surveillance system and therefore to even improve the required vision algorithms.

The next chapter will now present the conclusions gained through all this work carried out and also will identify possible future lines of research.

6 Conclusions and Future work

6.1 Introduction

This chapter concludes this thesis by giving a summary and conclusions of the work carried out through this project. The main emphasis of this work consisted in investigating how systems engineering could be applied in the conception, design and building of large-scale intelligent distributed real-time surveillance systems (usually called 3GSS in the literature in the field). To summarise the results of using such emphasis, this chapter has been divided in two main parts i.e. summaries of the investigation of existing design methodologies (section 6.2.2) and the application of one of them (RTN) to the design of a generic 3GSS (section 6.2.3). Section 6.3 highlights possible lines of research that may arise from the conclusions, presenting at the same time, the drawbacks found in this work.

6.2 Conclusions

The context of this work, as explained in chapter 1, was that it formed part of an EPSRC-funded project referred to as COHERENT (Computational Heterogeneously Timed Networks). The aim of COHERENT was to sketch and verify an architecture design to construct embedded real-time systems as on-chip systems (SoCs), called real-time networks on a chip (RTNoC), with potential applications in control and data processing [COHERENT 2005]. To design this architecture the DORIS methodology was proposed given its relevance in the design and construction of embedded real-time distributed system used in control applications. Moreover, as mentioned in Chapter 1, because the research was focused on heterogeneous systems, the inherent temporal diversity of these systems naturally led to the study of asynchronous communication mechanisms (ACM) to link different parts of the system, and asynchronous techniques applied into design and verification tools. Another reason for including RTN/DORIS in this project was, as explained in chapter 3 and 4, that RTN provides an asynchronous communication mechanism known as the four slot mechanism [Simpson 1990c] (in a

pool protocol) which allows fully asynchronous communication between the activities that are connected to the protocol.

6.2.1 How this research linked to COHERENT

The contribution of this work to the COHERENT project was to investigate the use of RTN and DORIS as a design methodology, to a specific important application domain namely the design of 3GSS. These systems, as mentioned on various occasions, are naturally heterogeneous, arising from the variety of timing requirements from diverse response times and processing rates of different parts of the system. Moreover, 3GSS are also inherently real-time and concurrent as discussed in chapter 2 and 4. It is also likely that some parts of these systems will even be embedded in DSPs, such as data processing parts, which are integrated into what are called “smart cameras”, as discussed in chapter 5. These systems have data processing parts as well as control parts, as demonstrated in Chapter 5. Therefore, because RTN/DORIS has been successfully used in designing and building embedded real-time distributed systems, the use of RTN for the design of these systems was proposed.

Moreover, as mentioned in chapter 2 and 5, the research field of surveillance systems has tended to be centred on the study of these systems from the vision algorithm point of view. Therefore, there is a lack of research based on the creation of these systems from system engineering point of view. Consequently, as illustrated in Chapter 5, it has proved difficult to successfully build robust large-scale systems without having available a methodology (or at least the practice of using a methodology) that helps the designers to understand and to build the system. Therefore, the main contributions of this work to the surveillance systems research field and to similar fields are: the identification of the need to find a system design framework appropriate to 3GSS, to develop and check the conceptual basis for such a framework, and to assess such a framework against the specific requirements of such systems.

6.2.2 Design methodologies

The research literature in design methodologies presented in chapter 3, was focused on discussing some of the most important object oriented (OO) methodologies. The reason for centring the review on OO methodologies is because OO technology is nowadays the most used technology to build most of the distributed real-time systems such as telecommunications or financial systems. Consistent with this tendency, most surveillance systems aimed at scalability and reported in the literature are also implemented using OO technology. Moreover, recent vision algorithms used in surveillance systems have tended to be implemented, using OO technology.

Nevertheless, different design methodologies for distributed real-time systems used mainly in the 60s and 70s were mentioned briefly. These methodologies were applied mostly for the creation of control systems [Gomaa 1993c]. RTN may be categorised inside these group. It was created on the 70s for the design and building of distributed real-time embedded systems for avionics control applications.

Once the review of some design methodologies is conducted, the study was centred on the justification of the selection of RTN/DORIS in preference to OO, which was based on a consideration of the conceptual basis of such methods. To justify the choice and to illustrate the conceptual differences between methodologies based on OO or RTN, a discussion on theoretical or conceptual ideas between the two technologies was presented. Then, for example, it was seen that these two technologies have conceptual ideas in common, such as the concept of object in OO and the concept of activity in RTN, which is the representation of an active software entity. Nevertheless, these ideas are expressed differently, e.g. an activity is an active task, therefore usually represents more than one object at the same time. The comparison between RTN and OO emphasised the differences in the basic concepts such as the one just explained, and in the communication model, especially the protocols taxonomy and the asynchronous communications that RTN provides. The properties that OO provides like inheritance or polymorphism and the properties that RTN provides like concurrency were also compared.

One of the conclusions extracted from the discussion of conceptual ideas between RTN and OO lies on the fact that, in RTN the system is conceived as a set of (relatively simple) active tasks (activities) that interact with one another through intercommunication data areas (IDAs), which altogether constitute the RTN network. When considered in detail, this is quite different to the object-centric view of synchronised message passing tied up to passive tasks (methods) when objects need to interact with one another.

From the point of view of design methodologies, one of the conclusions extracted from the comparison is that OO is a well known technology widely used and without any doubt, it will carry on being used. There is so much research going on, documentation, technical papers, books which help to improve OO day by day. Moreover, there are many tools that help designers implement easily simple systems. Furthermore, the design methodologies based on OO are suitable to build systems for different application domains. On the other hand, the OO design methodologies are more focused on the design of the system from the implementation point of view rather than from system engineering point of view, which is explained in following paragraphs.

As mentioned, one of the conceptual ideas in OO technology is the object. Therefore, a primary focus in these methodologies, consists in trying to find the objects (and hence classes) required to represent the system. The typical next step, once the nature and number of the objects have been found, is to define the internal structure of these objects, i.e. the values that each object should have and the methods that each object needs to access its values or to communicate with other objects. When each object is characterised, it usually follows a step consisting in defining the relationship between objects, if any relationship exists. Then if possible, the design consists of high cohesion relationships inside the object and low coupling relationships between objects. At the same time, design decisions are taken on grouping some objects depending on their functionality. Therefore possible components are defined. This is one of the most difficult parts of the OO design. Because OO defines several kinds of relationships

between objects, as mentioned in chapter 3, a bad choice may strongly affect the architectural design of the system. Notice that, this largely a bottom-up design process; by the time all objects are defined and their relationships established, the system design as a whole may be very difficult to picture. Moreover, if the language used to carry the design is, for example, UML, it can get even more difficult to readily understand a design, because there are too many diagrams and graphical components to depict elements of the design.

On the other hand, in the design methodologies proposed and used as far back as the 60s and 70s including RTN and further later extensions such as DORIS, the design of a system is focused on a system engineering point of view. Notice that, DORIS, which extended RTN concepts as mentioned in Chapter 3, is a complete framework to design real-time embedded systems. In RTN/DORIS the design *is* the solution to the system. Therefore, it is necessary to understand what the system has to do. The first step is to define the functionality of the system, what it tries to do. Once the functional definition has been done, the next question to answer is how to model what the system intends to do. Then, further refinements from the functional definition of the system are done to get the final solution, i.e. the RTN network. Therefore, in RTN the process through which a solution is arrived, enables the designer to have a better understanding of the whole system design compared to OO. The RTN design process starts firstly as a top-down but after it becomes a typical bottom-up process, because from these refinements it is possible to discover that more functionality is required, so the system functional definition can also be refined.

We can also note that most of the OO design methodologies reviewed here do not have a completely defined process that allows moving smoothly from the requirements and specifications of the system to the design and implementation of the system. However, the current research in Object Management Group (OMG) through a Model Driven Architecture (MDA) technology, tends to the creation of a framework to accomplish this. RTN/DORIS as mentioned in chapter 4, put special emphasis on a strong link between the different stages of design and development so that implementation

decisions can be traced back to the design. This is a very important practical consideration as otherwise; it is difficult to ensure that a particular implementation (perhaps when there is most pressure to deliver) is consistent with an agreed design. This break of the link between design and implementation may be an important source of errors and lack of project control when creating a complex system/software.

6.2.3 RTN/CORBA for designing real-time distributed surveillance systems

In chapter 4, the discussion of DORIS as a proposed design methodology to use in a possible framework to build 3GSS systems was extended. Chapter 4 presented further discussions and comparisons, through a case study, between RTN and OO technologies from an architecture design point of view. In chapter 4, an existing real-time distributed surveillance system called ADVISOR was studied. The first prototype version of ADVISOR used CORBA as middleware architecture solution to integrate the different parts of the system. CORBA as mentioned in chapter 4 is middleware based on OO concepts and it is aimed at easing the integration of heterogeneous platforms. Even though CORBA may be considered an architecture design framework to build distributed event real-time systems, rather than a design methodology, it was studied because one of the aims of this work has been to provide the theoretical basis to create a framework for building 3GSS. Therefore, CORBA could be a candidate to be used in such a framework. In fact, in some research work as in ADVISOR [ADVISOR 2003], the authors proposed the use of CORBA as the key component to create surveillance systems. Therefore, chapter 4 focused on comparing two different architecture designs of ADVISOR system: a middleware architecture design solution provided by CORBA and the architecture design solution provided by RTN, so that an insight into their weakness and strengths could arise.

After illustrating the differences at the architectural design level between the two solutions, the discussion continued based on three issues: concurrency and distribution, run-time facilities and finally development aspects. Because 3GSS are strongly concurrent and distributed, it is clear that it is mandatory to use technologies that

provide architecture solutions which are concurrent and which may be distributed. Therefore, part of the discussion in chapter 4, was centred on how both approaches deal with distribution and concurrency. On the other hand, run-time facilities and development aspects are important issues to consider in technologies which are possible candidates for their use in a framework to design and to create such systems. Moreover, if the system to be created has to be a real-time one, run-time facilities are crucial.

One of the outcomes of this comparison, which is connected with the conclusion explained in previous section 6.2.2, is that even though CORBA provides an easy solution to the integration of heterogeneous parts of the system, the architectural design solution is focused on a solution from an implementation point of view rather than from a system engineering point of view. See Table 6-1. The last statement is based on the fact that, in CORBA, a solution is based on defining the CORBA objects needed and the relationship between them (this is clearly a result of its OO roots). Because CORBA is a middleware technology the way CORBA components actually communicate is completely transparent to the designer. Therefore, although CORBA reduces the difficulty at the implementation level, it inevitably also reduces the full understanding of the solution as a system, as illustrated in chapter 4. The designer does not really know how this transparent communication is done, because it is left to the chosen vendor. It is like the structure of a house, where it is important to know and build each component inside the room like the sink, table so on, but it is also important to know which and how rooms need to be connected in the house. If the architect does not show (explicitly) what the structure of the all house is, i.e. the functional plan of the house, and how each room should be connected, e.g. where each door should be located to connect one room with another, then it is not possible to understand how the house is structured. Furthermore, if there is a problem or change to do, it will be very difficult to solve the problem or to realise the convenient change. The transparent communication that CORBA provides (normally accepted as a good feature because it makes the process of implementation easier and faster), might prevent the designer from conceiving the different interactions between parts of the system in such a way that

better reflects the problem (as a loose interconnection of concurrent and possibly asynchronous communicating processes). Therefore, a CORBA architecture design solution might work against a simpler and hence potentially more robust design solution of a distributed real-time system, as presented in chapter 4.

The other outcome (see Table 6-1) is that, as mentioned in chapter 4, CORBA, at the time of writing, has a lack of development facilities, to help the designer to go through the design and construction of the system. Moreover, the run-time facilities are strongly dependant on the ORB vendor. Therefore, although CORBA technology has been used in many real-time applications even critical applications such as telecommunications systems, these two conclusions are significant enough to question the inclusion of this technology to the framework for designing large surveillance systems.

		Advantages	Disadvantages
Underlying communication details	CORBA	The designer need not worry about these details which are transparent to the designer.	The designer does not have to complete information to gain system knowledge.
	RTN/DORIS	Greater understanding of the system.	Communication design between activities proves difficult at the design stage for the designer.
Run time support	CORBA	It is carried out by the vendor that implements the ORB and is transparent to the designer. Thus, allowing easy integration of different platforms.	Lack of control of system management.
	RTN/DORIS	Conducted through the MASCOT machine. Full control of system management.	
Development aspects	CORBA		It does not guide the designer through the design and implementation process.
	RTN/DORIS	Provides consistency in system design.	

Table 6-1. Summary of advantages and disadvantages using CORBA and RTN/DORIS.

Once the justification of the inclusion of RTN methodology in the framework had been argued, chapter 5 presented a generic design of a large-scale distributed semi-intelligent real-time surveillance system to illustrate the importance of applying system engineering to the design of such systems. The most important outcome of this chapter 5, apart from the presentation of a proposed architectural design, is that even though these systems have data processing tasks based on computer vision algorithms, these algorithms are just one part of the whole system. The rest of the parts that constitute the system are equally important. Therefore, the proposed system design illustrates that surveillance systems are not just a cluster of vision algorithms that are grouped to create a system. The creation of the functional definition of the system, the definition of different types of data and the distinction between the control parts of the system and data processing parts of the system, provides enough elements to understand how a system of such characteristics should work.

Since one of the main goals in chapter 5 was to propose a generic design for a generic 3GSS, after presenting and discussing the proposed software architecture design using RTN, a discussion on the logical network design was conducted. As mentioned several times, the RTN solution provides a naturally concurrent and distributed solution. Therefore, once the network is defined, the designer can assume that the solution may be distributed. To understand the distribution of the real-time network solution, it was convenient to propose a logical network design for the designed system. Therefore, a logical network design solution, that could map the software real-time network design presented previously, was presented and discussed in Chapter 5. Notice that the proposed logical network design was strongly influenced by the suggestions taken from network designs books from CISCO [Paquet and Teare 2001].

Surveillance systems like any multimedia application, have an important requirement in terms of Quality of Service (QoS) that the system should be able to provide at any given time. Three parameters are of primary interest for QoS when it comes to processing and transporting multimedia data: bandwidth, latency and loss rate. One

way to manage the QoS and guarantee a good QoS, is to know the number of processes and memory, which are required for the application for allocating enough resources and for applying the right scheduling policies. Then, another conclusion extracted from chapter 5 is that, by using RTN it is possible to directly quantify at the end of the design phase (i.e. when the network is determined), the number of tasks and an approximation of the memory that will be required to communicate certain tasks. Note that, after the design of the system using MASCOT, the next process, following DORIS methodology (as illustrated in Figure 3-3), corresponds to mapping the activities and the IDAs to hardware. Therefore, the activities may be mapped to different processors in a multi-processor environment. If the activities are in a co-processor environment with shared memory, the IDA that the activities use might be mapped to the shared memory hardware component. If the activities are located in a multi-processor environment without shared memory, then, the IDA might be mapped, through the template substitution, to an external shared memory with an active element, i.e. a thread that moves the data from one side to the other.

6.3 *Future work*

In this section some possible lines of research obtained from the conclusions extracted at the end of this work are presented. These are presented in two different sections. Section 6.3.1 is related to the framework to design and build 3GSS and section 6.3.2 presents the idea of applying formal methods to the framework. Moreover, in these sections the drawbacks found during this work are also presented.

6.3.1 Framework for designing real-time distributed surveillance systems

The framework for designing these systems should consist in a design methodology associated with CASE tools, which may help provide consistence to the process of design, including a library of designed components for surveillance systems defined in chapter 5. Also, the framework may include a semi-automatic translation to a predefined language code of the designed components. Finally, to verify and to formalise the consistency of the final design it may be necessary to add to the

framework some mathematical techniques that allow by formalism to validate the final design solution. Following paragraphs in section 6.3.1 and 6.3.2 expand each mentioned idea for the framework.

Although, as concluded in chapter 3 and 4, this work proposed the use of RTN concepts into the framework, a significant drawback to the use of RTN and DORIS technology lies in the fact that the tools that MATRA BAe Dynamics (MBDA) use, are not part of a standard and thus, only the company is benefiting from the use of RTN concepts. The use of this tool outside the company is done only through a few collaborative research projects with universities like COHERENT, and only one part of the CASE tool (i.e. MADGE) is provided. Therefore, it is not possible to use DORIS directly as a framework to the design of surveillance systems.

In chapter 5, the bases for the creation of this framework were established. The definition and encapsulation of different RTN components into subsystem modules were presented and we showed how these subsystems can be used in the design of a generic surveillance system. Thus, these generic subsystems can be held in libraries and then can be added to the framework making them available for the design of such systems. Nevertheless, an important drawback that has been found using RTN concepts to design such systems is that RTN networks are by nature static, although RTN has the tools to allow dynamic RTN network designs. On the other hand, OO methodologies and technologies based on OO concepts such as CORBA, claim that they provide good scalability to the system, by allowing the creation of dynamic components and their integration into the system “on-the-fly”. It is usually through the use of data repository components, which contain the required information to create components dynamically. Nevertheless, it is strongly recommended by RTN practitioners that the activities should not be created dynamically and the network should remain invariant at run-time. This is because the dynamism in the network as explained in chapter 3 and 4 may provoke unexpected behaviours, which may affect the stability of the system. It may also increase the lack of control management over the system and over the resources raising possible failures within the system.

However, surveillance systems can be by nature very dynamic. It is common to find the need to add new components like sensors or data processing units to the system within its lifecycle. Therefore, research needs to be conducted into the study of the extension of RTN, i.e. DRTN (Dynamics in RTN), to the design of dynamic systems. In this case, a possible solution to combine the dynamism of surveillance systems and the static nature of RTN network designs is obtained by following what RTN practitioners recommend: once the design of the upgraded component is done, it should be integrated to the RTN network by turning on the new component and then, turning off the old component without switching down the whole RTN network. Moreover at the design phase, e.g., if the designer wants to attach more than one DPU to the CC0 subsystem defined in Chapter 5 (see Figure 6-1), then the new component has to be extended easily to allow the attachment of a new DPU. In other words, the creation of extended components from the static components (templates) found in the library of the framework in the same design, should be a straightforward process. Once the new CC0 is extended, it should replace the old component in the network. Therefore, the RTN network may be upgraded under the management control over the new components that have been inserted, avoiding then, the dynamic creation of new upgraded components.

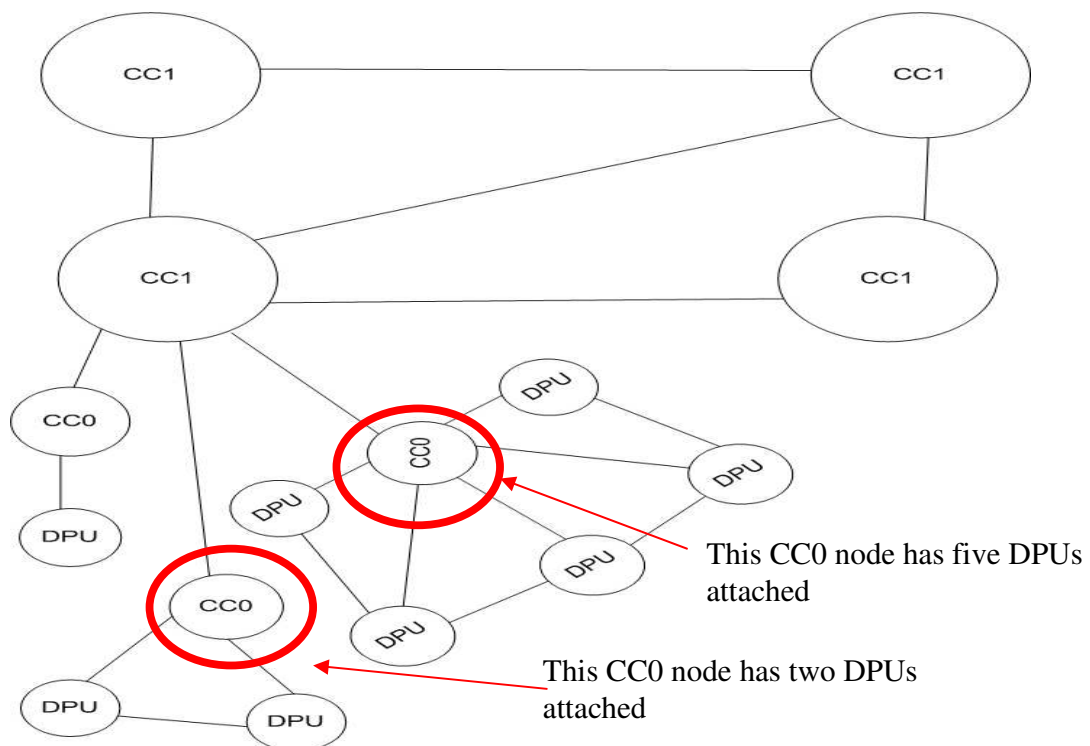


Figure 6-1. Topological network view of the generic 3GSS presented in chapter 5.

An ideal framework helps the designer to go step by step from the capturing requirements process through the design, implementation and building process. Unfortunately, there is no a current methodology that can accomplish this, not even with DORIS methodology. Besides the substantial research in developing modelling tools to execute designs allowing the evaluation of the design decisions, there is research in software engineering to make the process of capturing requirements an automatic process. However, this process requires significant interaction with the users therefore; it is extraordinarily difficult to capture this process automatically.. Even though the work reported in this thesis has focused on the design phase, the capture of the requirements and specifications of a generic surveillance system was necessary, and was done through the research and study of several existing surveillance systems. Therefore, the requirements for a generic surveillance system are captured and expressed through the functional definition of a generic surveillance system, illustrated in chapter 5. Even though requirements were captured, it was a laborious process that required the research and the review of many existing surveillance systems to conclude with the obtained possible generic requirements for such systems. Therefore, like in

DORIS methodology which as explained in chapter 3, it uses the CORE method to capture the requirements of a system. Thus, it should be interesting to research the building of a semi-automatic process (instead of automatic process, because the interaction with the users is still necessary) to help to capture the requirements for specific surveillance application.

As mentioned, the design phase of the framework proposed here follows the RTN concepts and DORIS methodology. Therefore, the CASE tool for this framework should be consistent with the standard notations as DORIS. Moreover, connections between these components should use the same RTN concepts of paths, ports and windows; ports components used by active elements, windows used by passive elements and paths to define the connection between port/windows.

Once the design is realised, as mentioned in chapter 3 and 4, MADGE tool checks the (internal) consistency of the design through the stages defined as part of the development aspects in MASCOT-3: Introduced, Register and Enrolled. In the last stage, the modules are completed and coded (in ADA programming language). As discussed above, nowadays OO programming languages have a strong popularity in the implementation of visual surveillance systems. Many programmers know at least one OO programming language only a few of them know the ADA programming language. In MBDA, as deduced from a private conversation [H. Simpson 2005], some programmers tend to carry on the design and the implementation of a corresponding activity in a subsystem using UML and then an OO programming language, once they are assigned the implementation of a specific task, which has been obtained from the design of the whole system using RTN/DORIS. Furthermore, in Newcastle University, some research has been conducted to map some RTN concepts like the pool protocol using agent programming languages. Thus, a possible research line could be to carry on the study of the generation of a library of RTN components to conventional OO programming language like C/C++ to generate then the code in such a language instead of using ADA language. Although as mentioned in chapter 3, the use of these kinds of languages into the implementation of the design of surveillance systems may not be a

pragmatic approach; it may provide less control management over the system, and unexpected behaviours could arise. There is also for example the possibility of bringing up unwanted effects such as memory leaks.

6.3.2 Inclusion of Formal Methods to the framework

As introduced in chapter 1, there is research work to apply formal methods to the requirements and to high-level designs where most of the details are abstracted, or to apply them only to the most critical components. Formal methods may be defined as the mathematically rigorous techniques and tools for the specification, design and verification of software and hardware systems. In other words, a formal method is a formal proof that verifies that the created system accomplishes its specifications. The value of formal methods is that they provide a means to symbolically examine the entire state space of a digital design (whether hardware or software) and establish a correctness or safety property that is true for all possible inputs. However, this is rarely done in practice today (except for the critical components of safety critical systems) because of the enormous complexity of real systems and the lack of understanding of what formalism can be associated with a particular representation or problem domain (for example, we talk about real-time throughout this work but temporality is never represented explicitly).

The textual notation and the graphical notation in RTN/DORIS are the two forms that help to control the evolving design structure, wherein each stage must reflect precisely the definitions set out in previous stages. As suggested by [Mustafa 2000], RTN/DORIS practitioners use contingency analysis¹³. This analysis technique is time and resource consuming, but does not provide an exhaustive test. In other words, at the time of writing, RTN/DORIS methodology does not apply any formal method directly to the verification of the high-level designs.

On the other hand, RTN/DORIS practitioners claim [IECCA and MUF 1983b] that the method is formal enough to provide the necessary visibility to support management and

¹³Approach which consist in prototyping a component, testing it and simulate over time domain if it works.

control of the design during development and subsequent maintenance. Note that this formalism on the design process is concerned with the idea of applying rigorous techniques to the design process rather than defining the designs using mathematical tools. However, drawing from a private conversation [H. Simpson 2003], RTN/DORIS practitioners also claim that a design using DORIS is formal, because the design is constituted by a network of basic RTN components, which have already been verified using a formal verification technique such as RTL¹⁴, which models the design of these components to evaluate their design. Nevertheless, as stated in [Mustafa 2000], it is not the same to say that “as far as we can tell, there are no errors” than to say “there are no errors in the design”. The last statement implies the proof of the design.

The following formal modelling languages have been used in the past to try to verify some RTN components or some RTN/DORIS designs. In [Simpson 2003f] a formalism using RTL is applied to the definition of the four protocols and the routers of RTN illustrated in chapter 3. In [Clark 2000], [Feixa 2000], [Mustafa 2000] the authors applied Petri nets¹⁵ and Coloured Petri nets to verify the three and four slot communication mechanisms. In [Haveman 1997] the author verified a simple system that receives data through two different sources, using RTL. In [Paynter 2000] the authors research on the integration of new formal notation for the specification of the temporal and functional behaviour of the concurrent processes, to supplement and formalise DORIS method. In [Muñoz 2002] the author modelled RTN components using LOTOS and pointed to possible ways in which RTN designs could also be modelled and verified using this technique. Therefore, it should be interesting to carry on with the research work established in [Muñoz 2002], researching further more into the mapping of RTN/DORIS components to the E-LOTOS language to obtain the formalism required in the designs, and then, to include this into the framework to design surveillance systems.

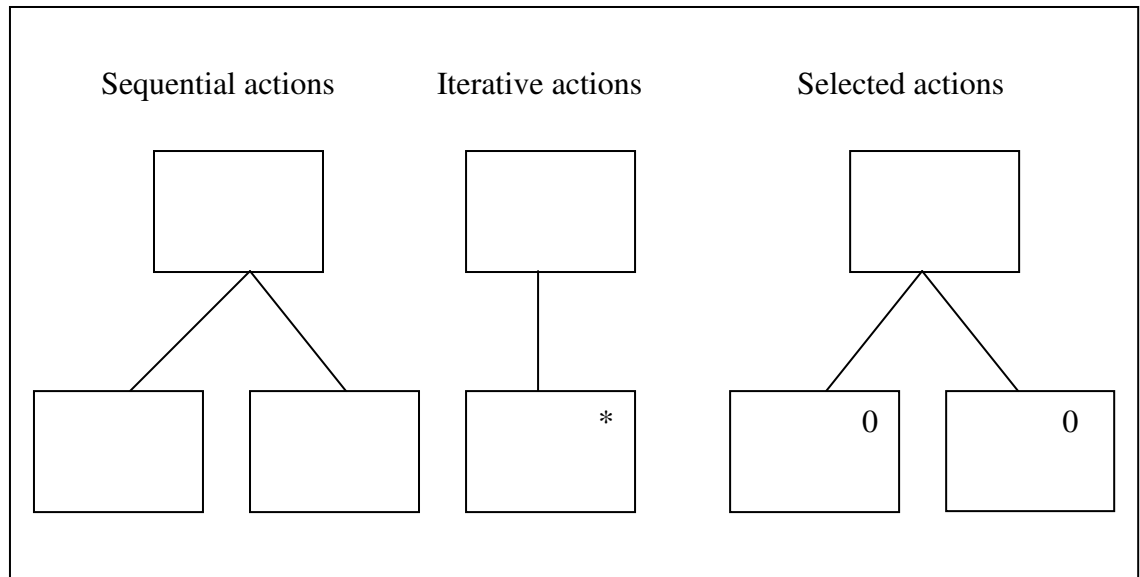
¹⁴ It is the first order logic language for reasoning in system events about time occurrences.

¹⁵ It is a formal, graphical and executable technique for the specification and analysis of concurrent, discrete-events.

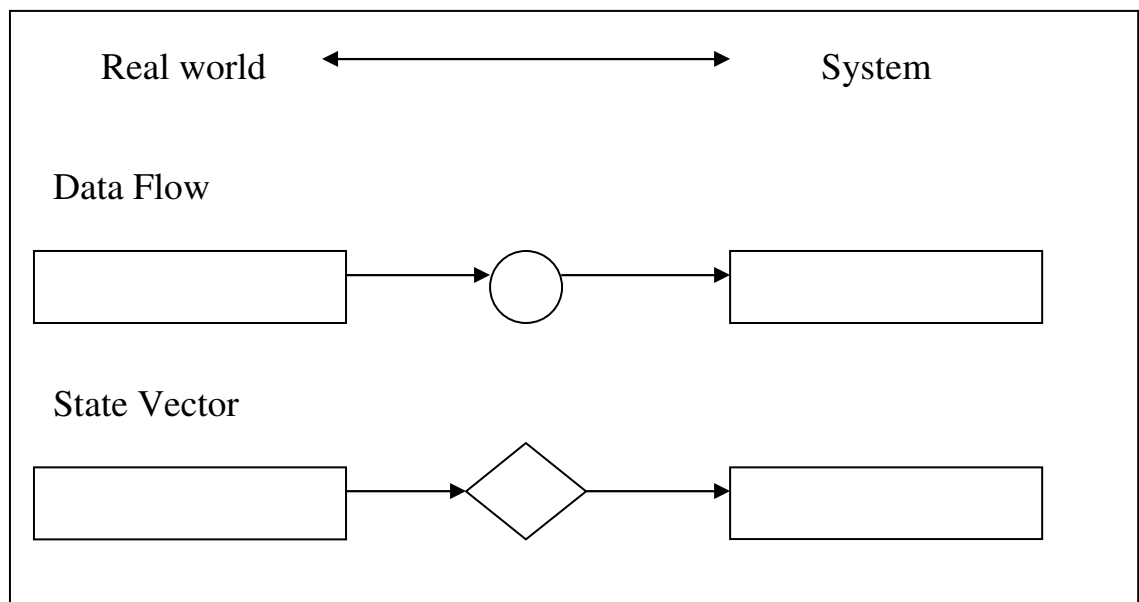
Appendix A

Jackson System Development

Action Structure Diagram- illustrates the structure of the different actions.

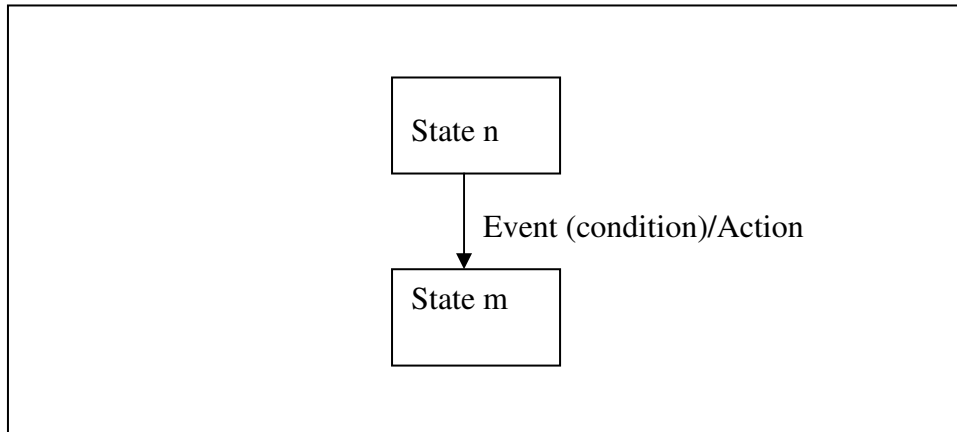


System Specification Diagram- illustrates the specification of the system as a model of the real world.

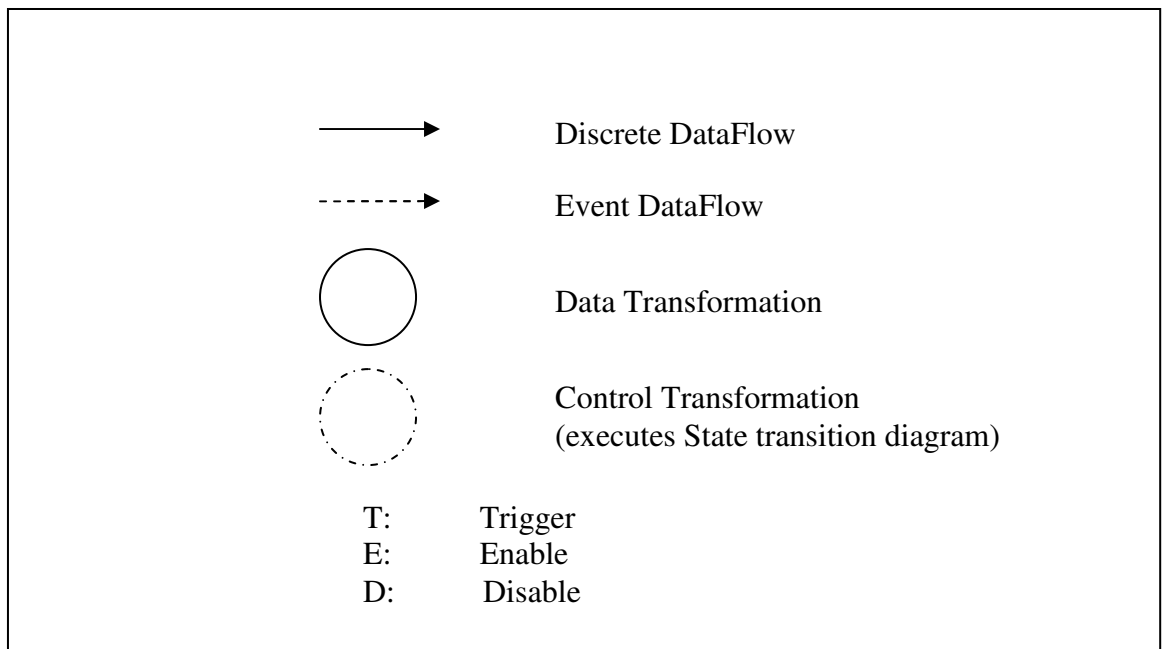


ADARTS

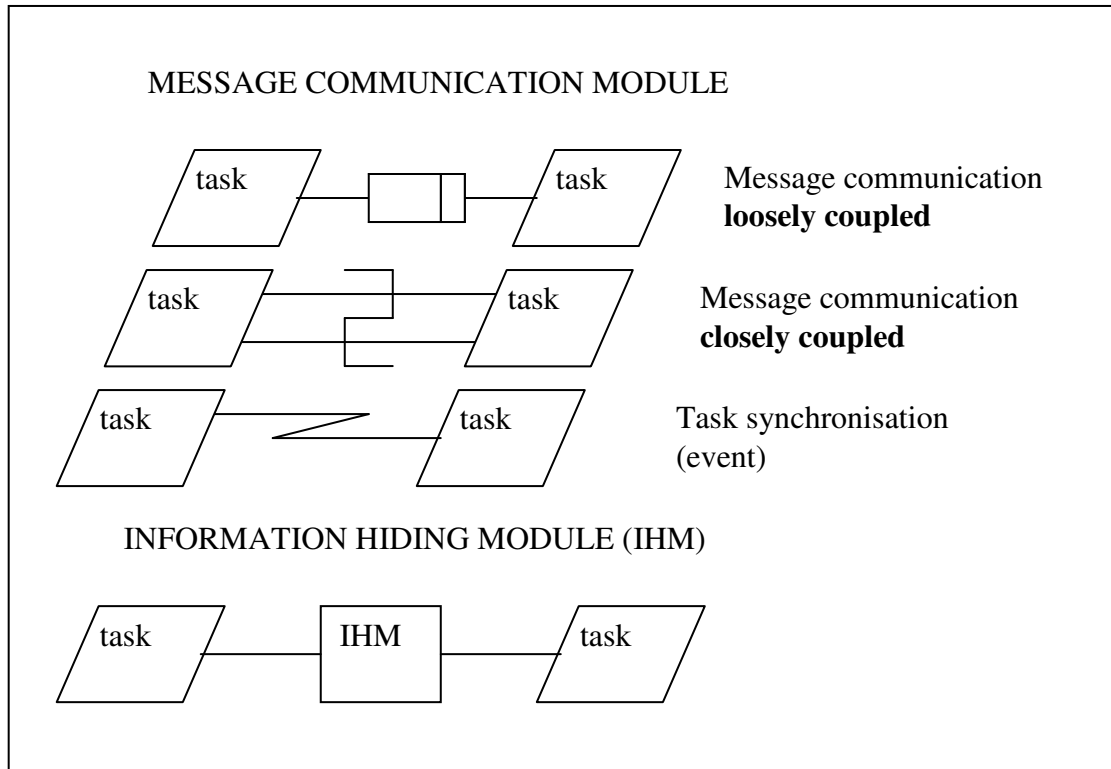
State Transition Diagram- illustrates the states and the transitions that occur in the system.



Data Flow/Control Flow Diagram- shows the relationship between the control and data transformations.

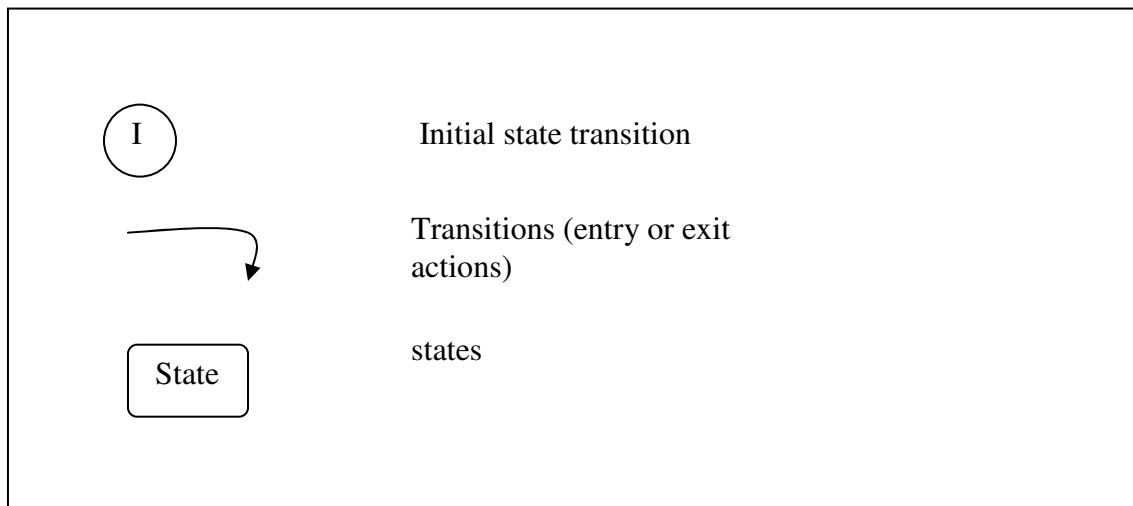


Message Communication Module and Information Hiding module- represents respectively all the possible communication cases among tasks and the data stores or state transition tables.

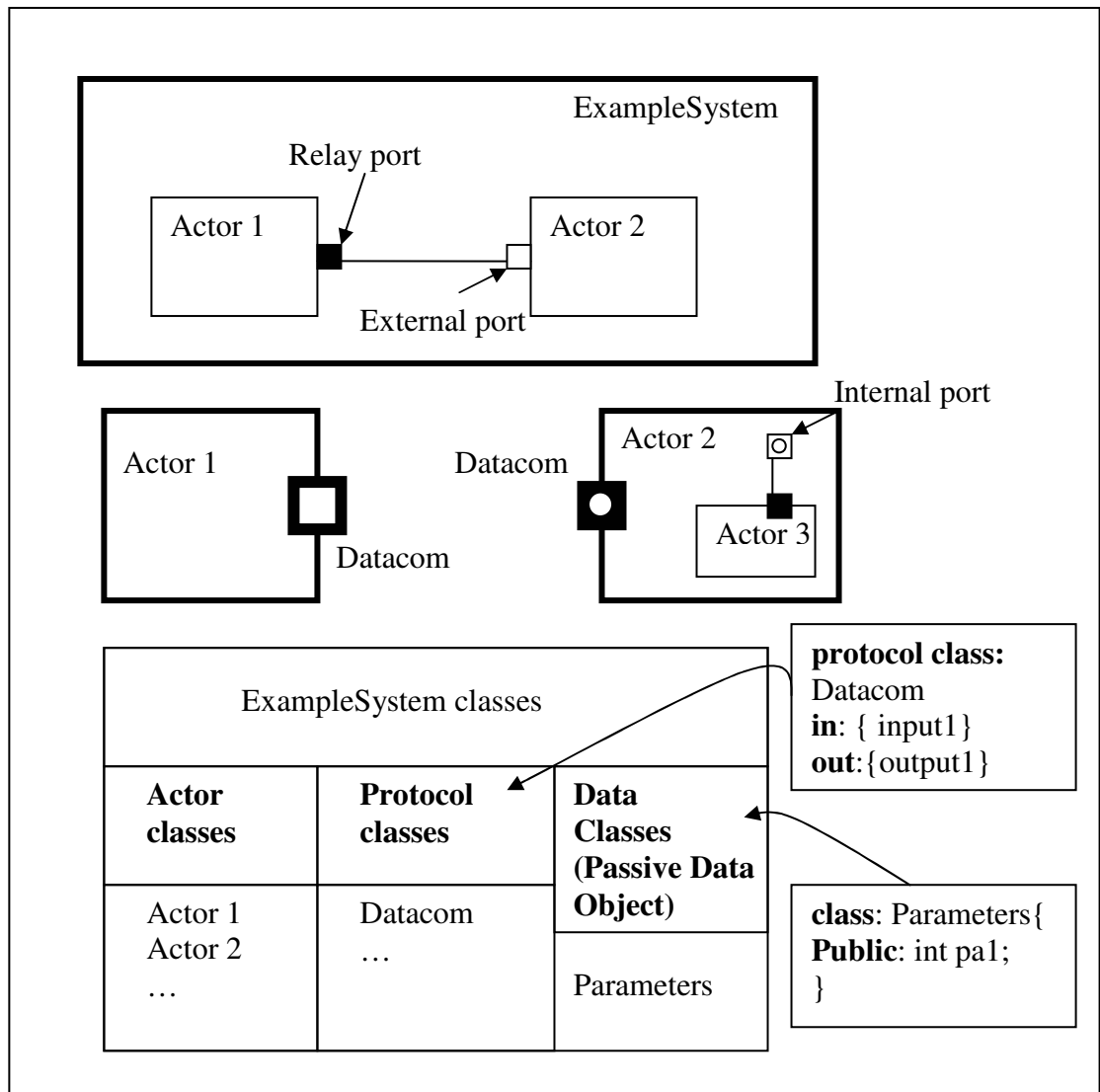


ROOM

ROOMCharts- represents the finite state machine of the system

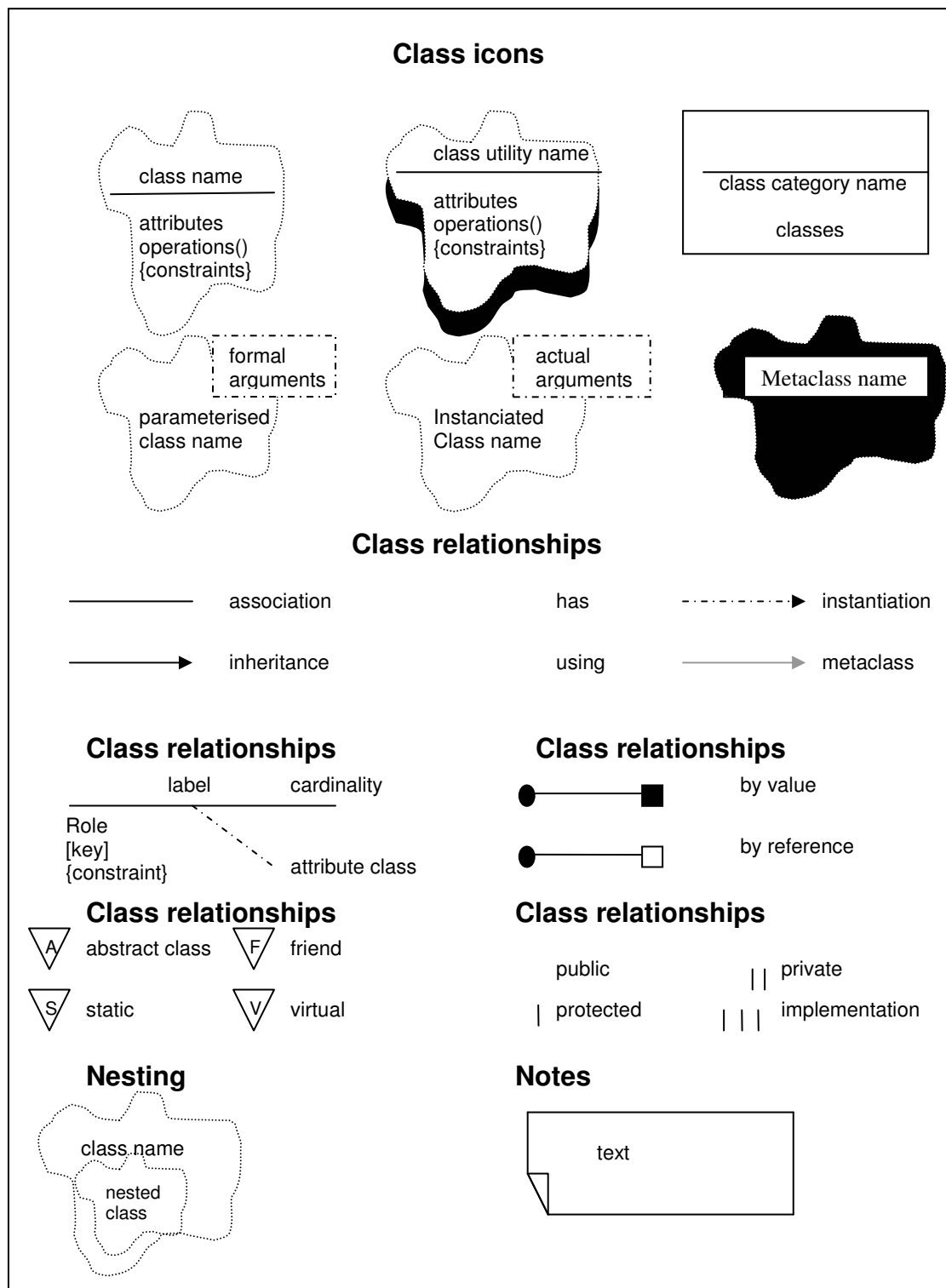


ROOM entities- the graphical notation of the elements used in ROOM.

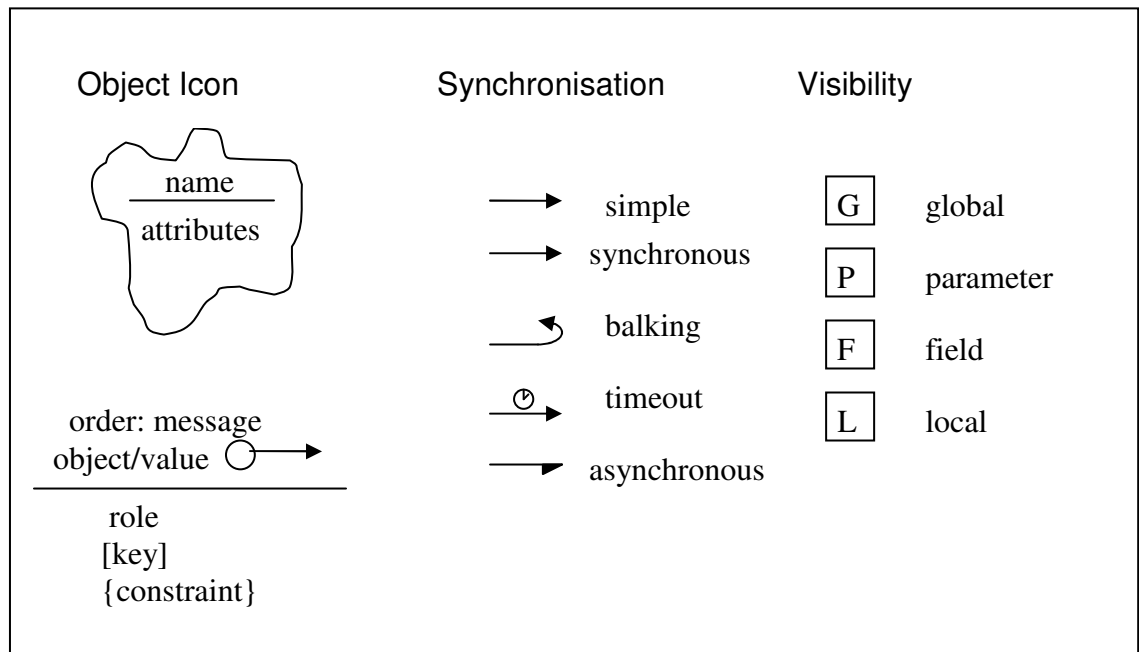


BOOCH

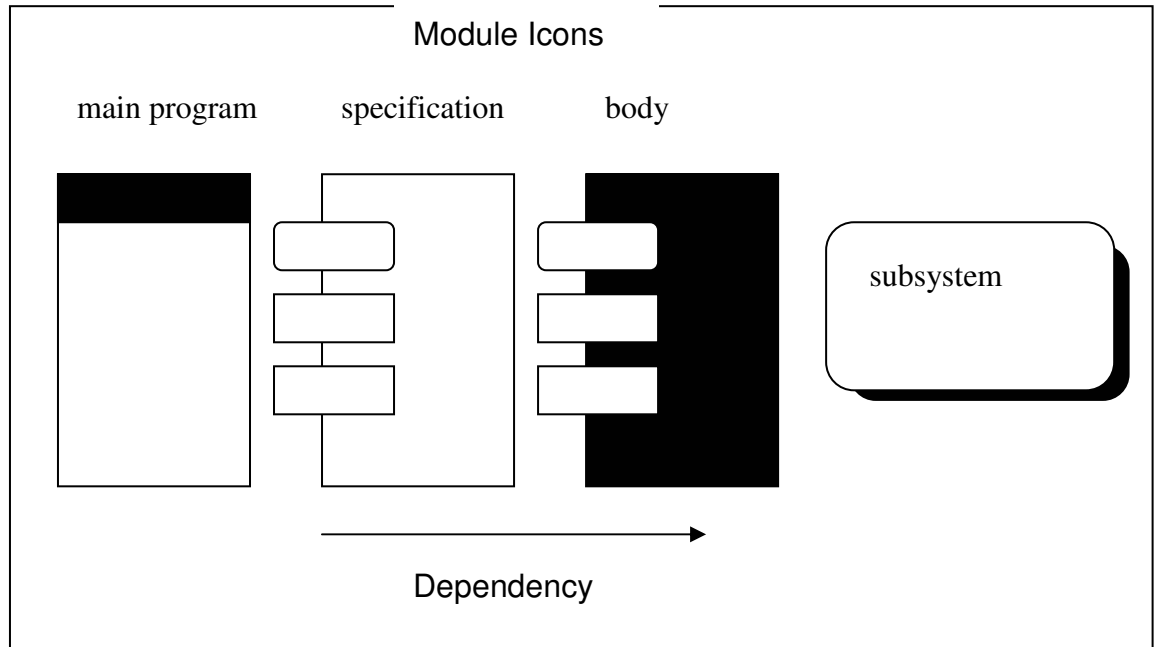
Class Diagram- represents the classes and their relationships.



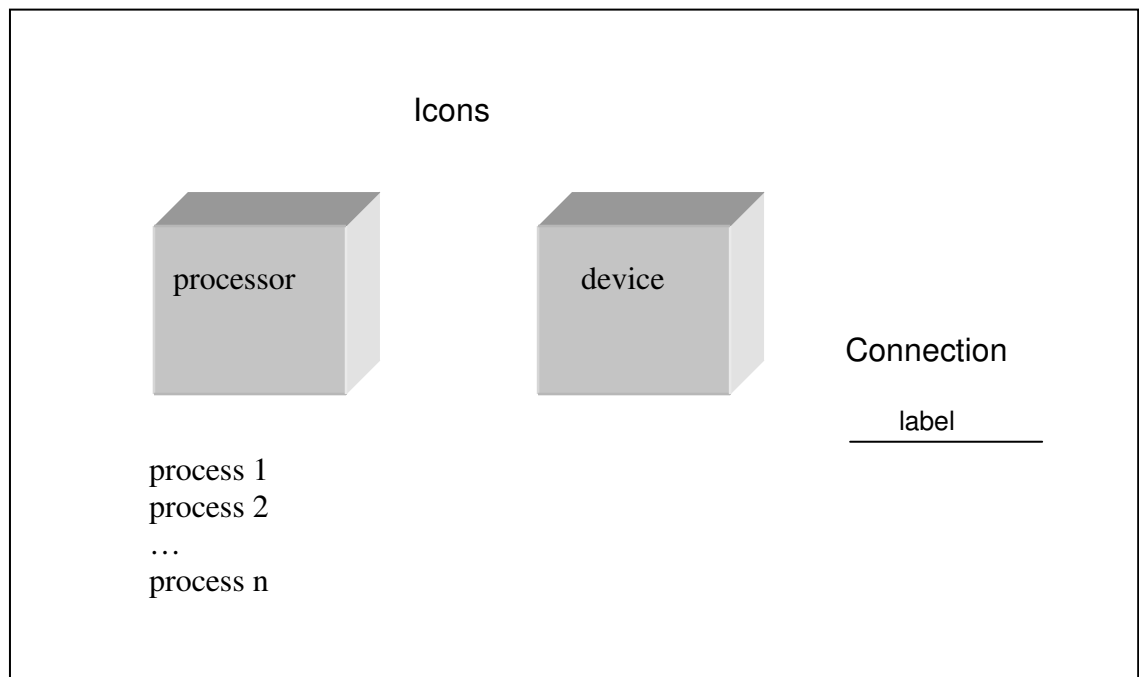
Object Diagram- represents the objects that make up the system and their relationships.



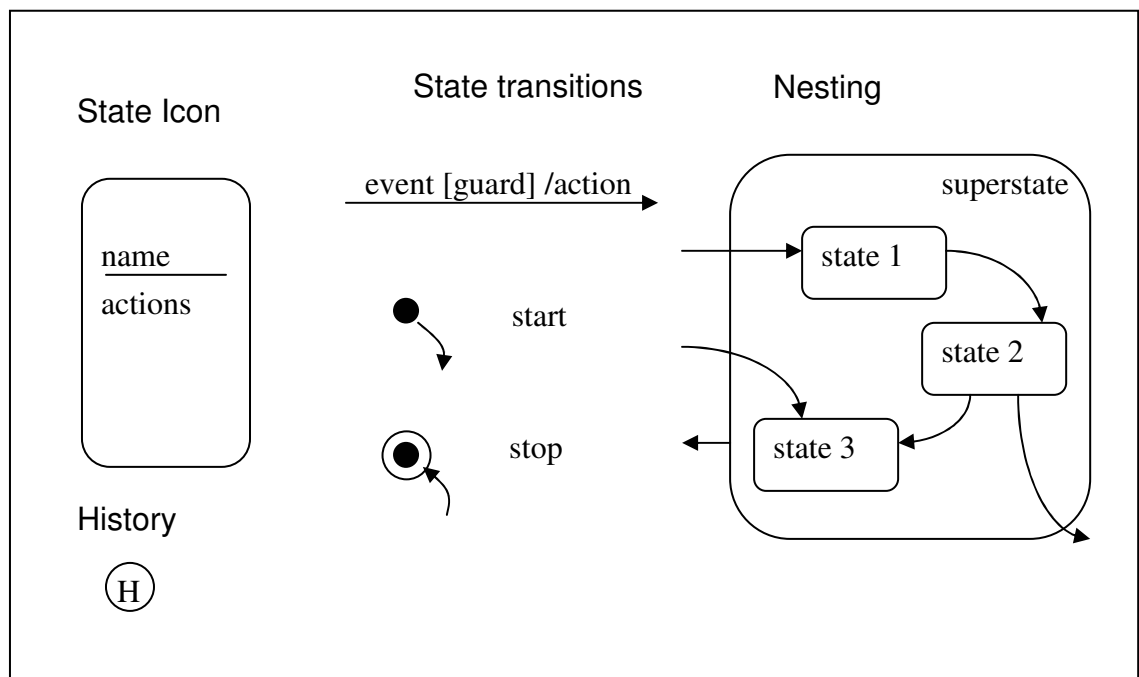
Module Diagram- illustrates the number of classes and objects in the module.



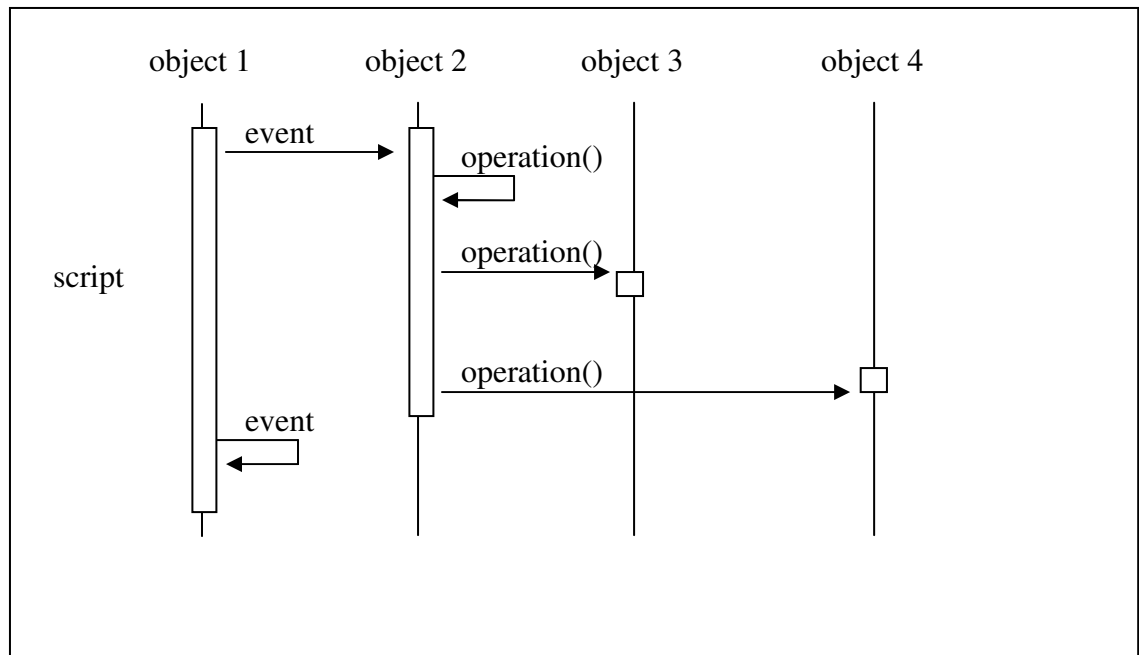
Process Diagram- shows how the processes are going to be mapped to processors.



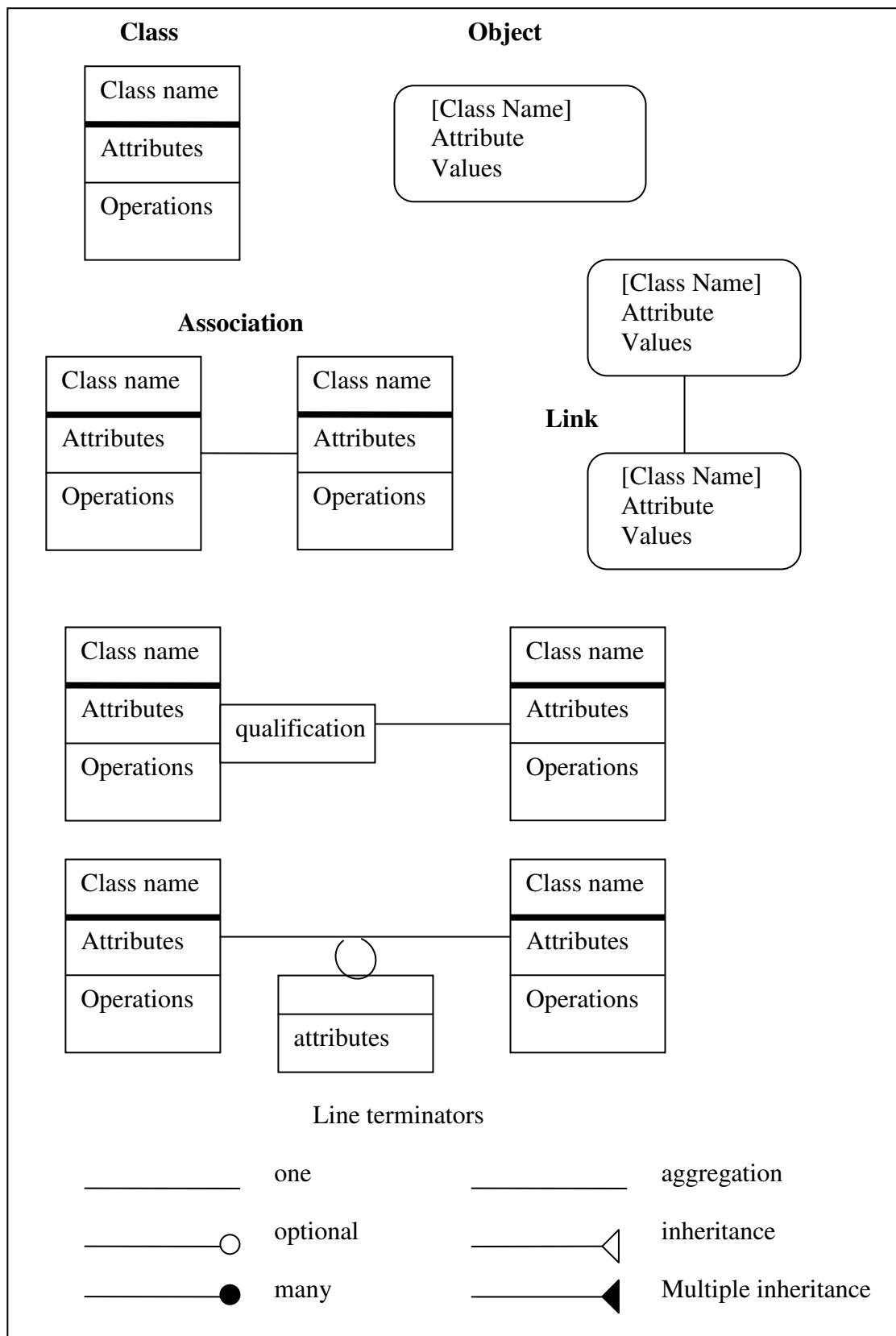
State Transition Diagram- illustrates the events that cause a transition of a state and the actions that result from that state change.



Sequence Diagram – illustrates the interactions between objects occurring at run-time.

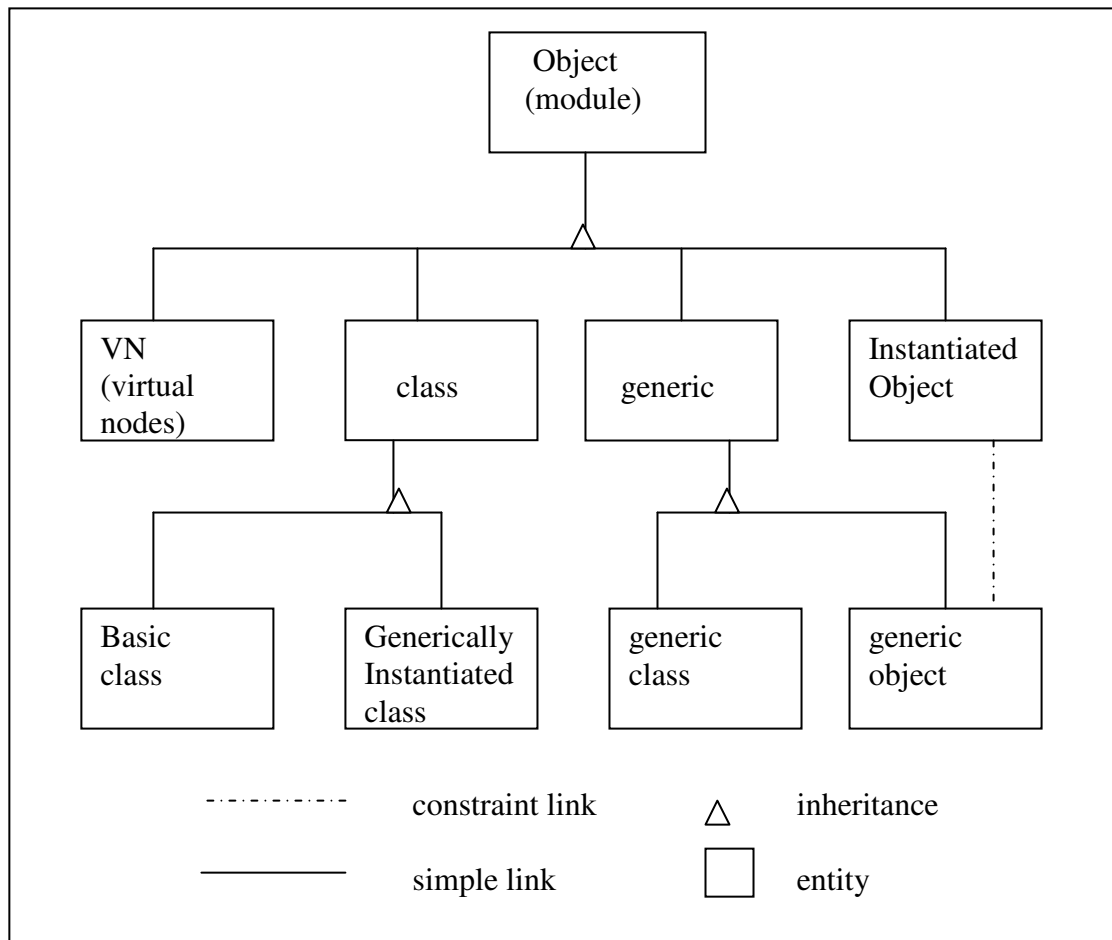


OMT- the graphical notation of OMT entities

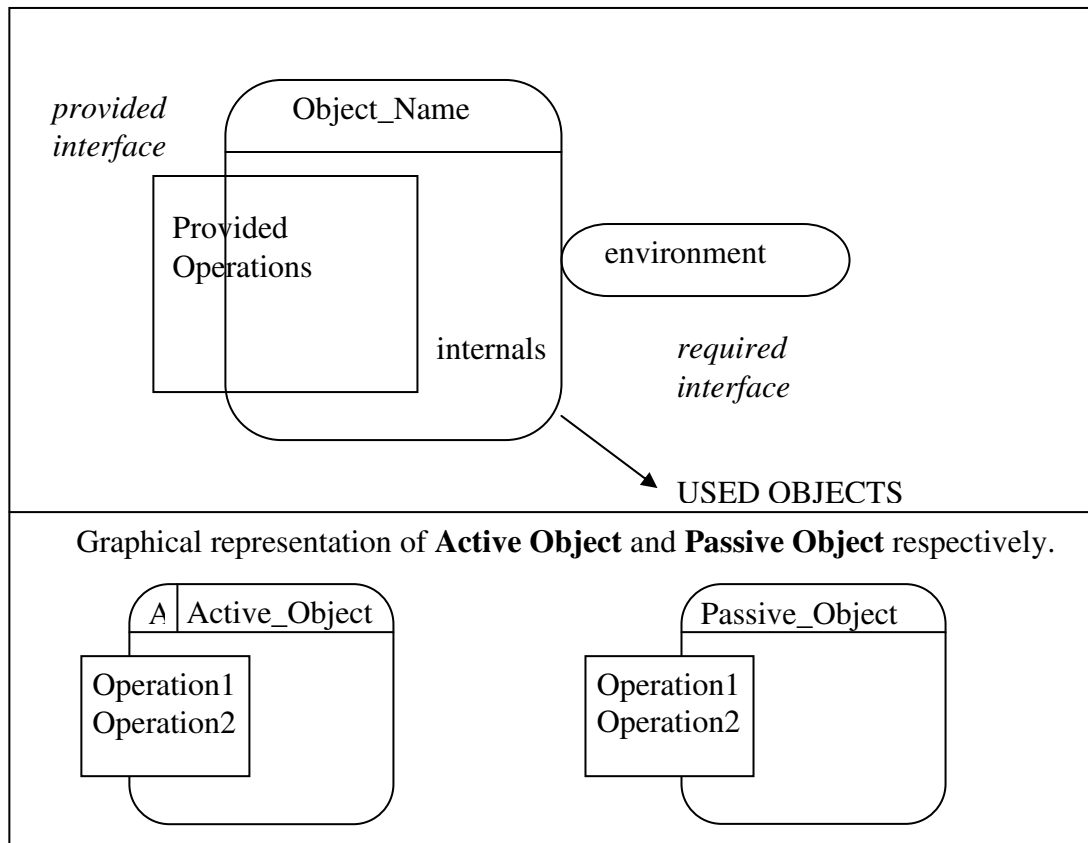


HOOD

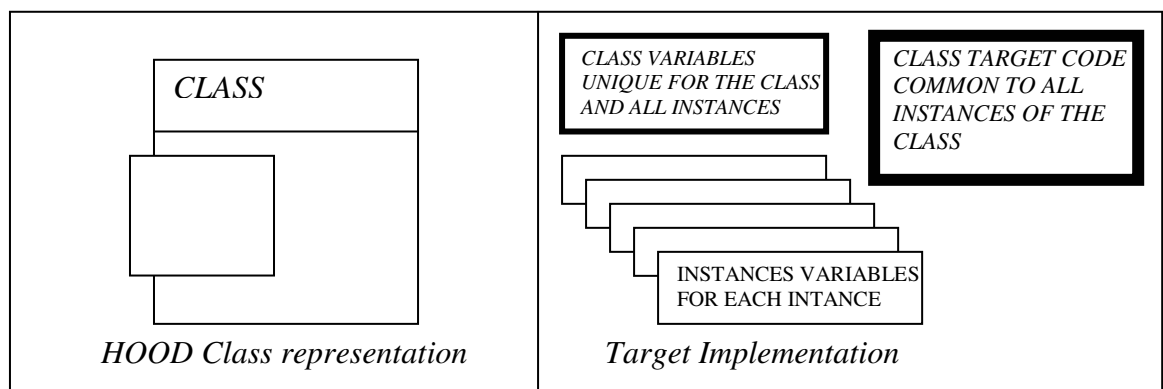
Summary of HOOD entities- using OMT notation [HOOD 1986a]



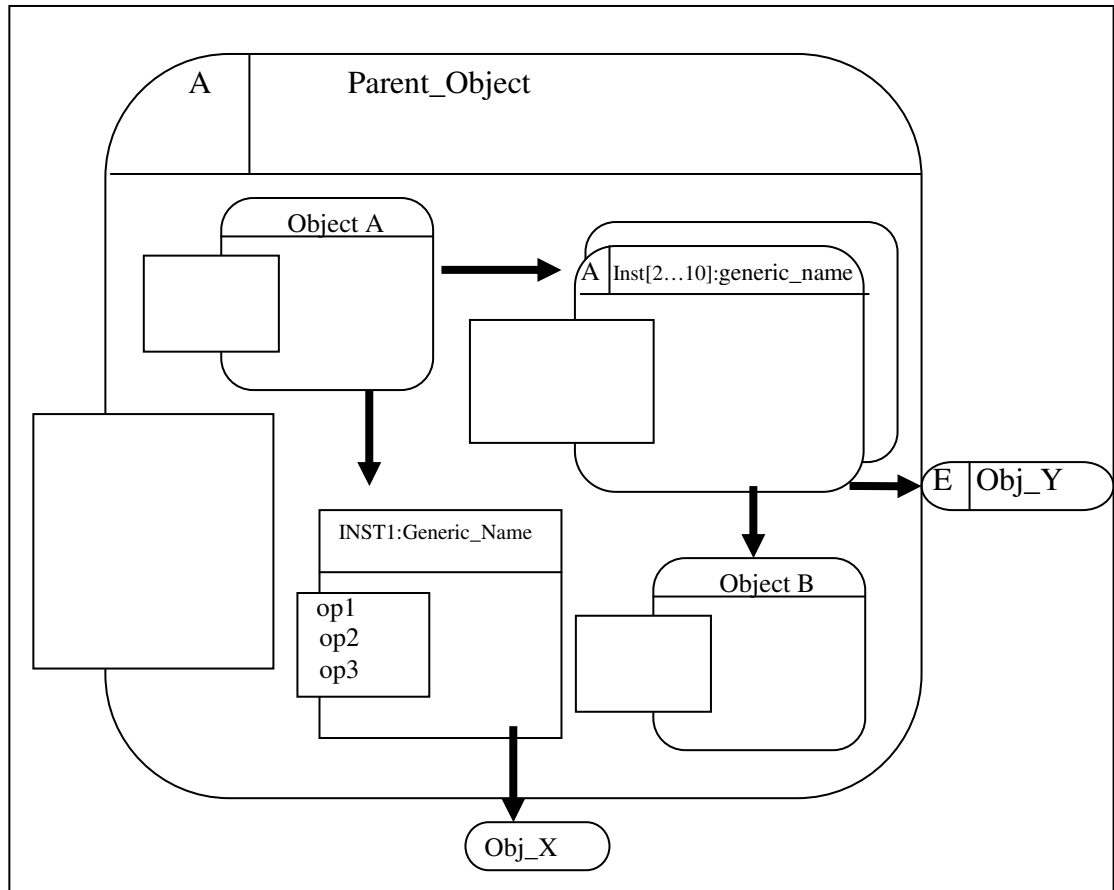
HOOD Object- the graphical notation Object Description Skeleton



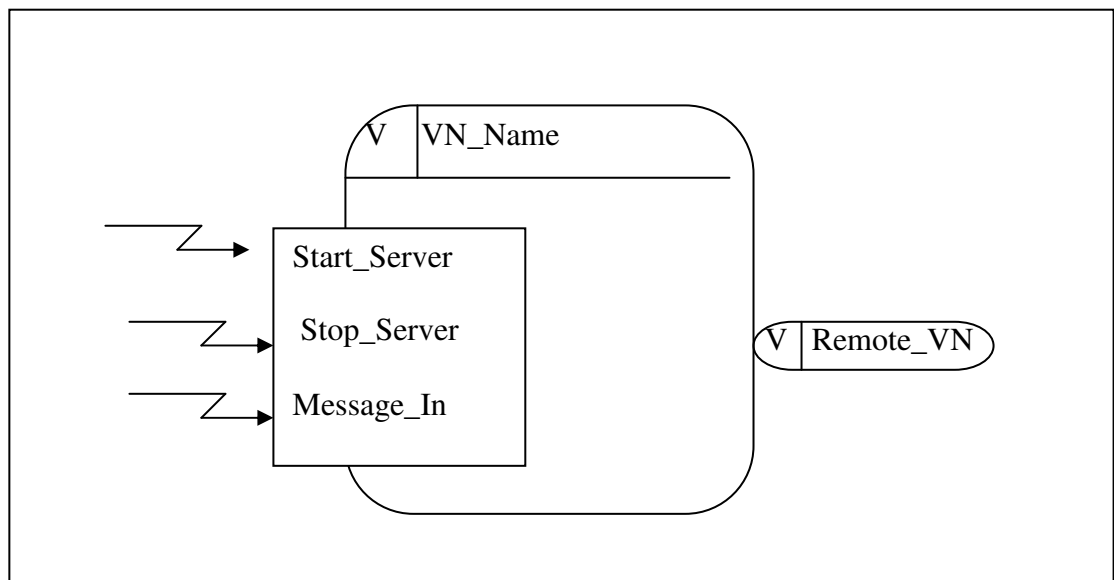
HOOD Class and instances implementation



HOOD generic instance

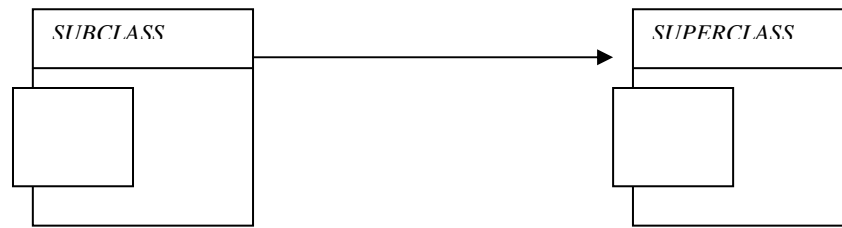


HOOD Virtual Node



HOOD relationships

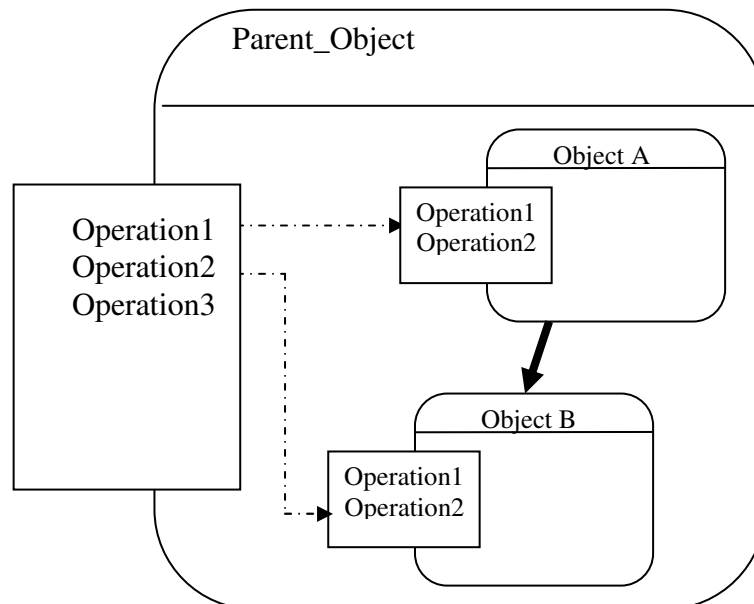
inheritance



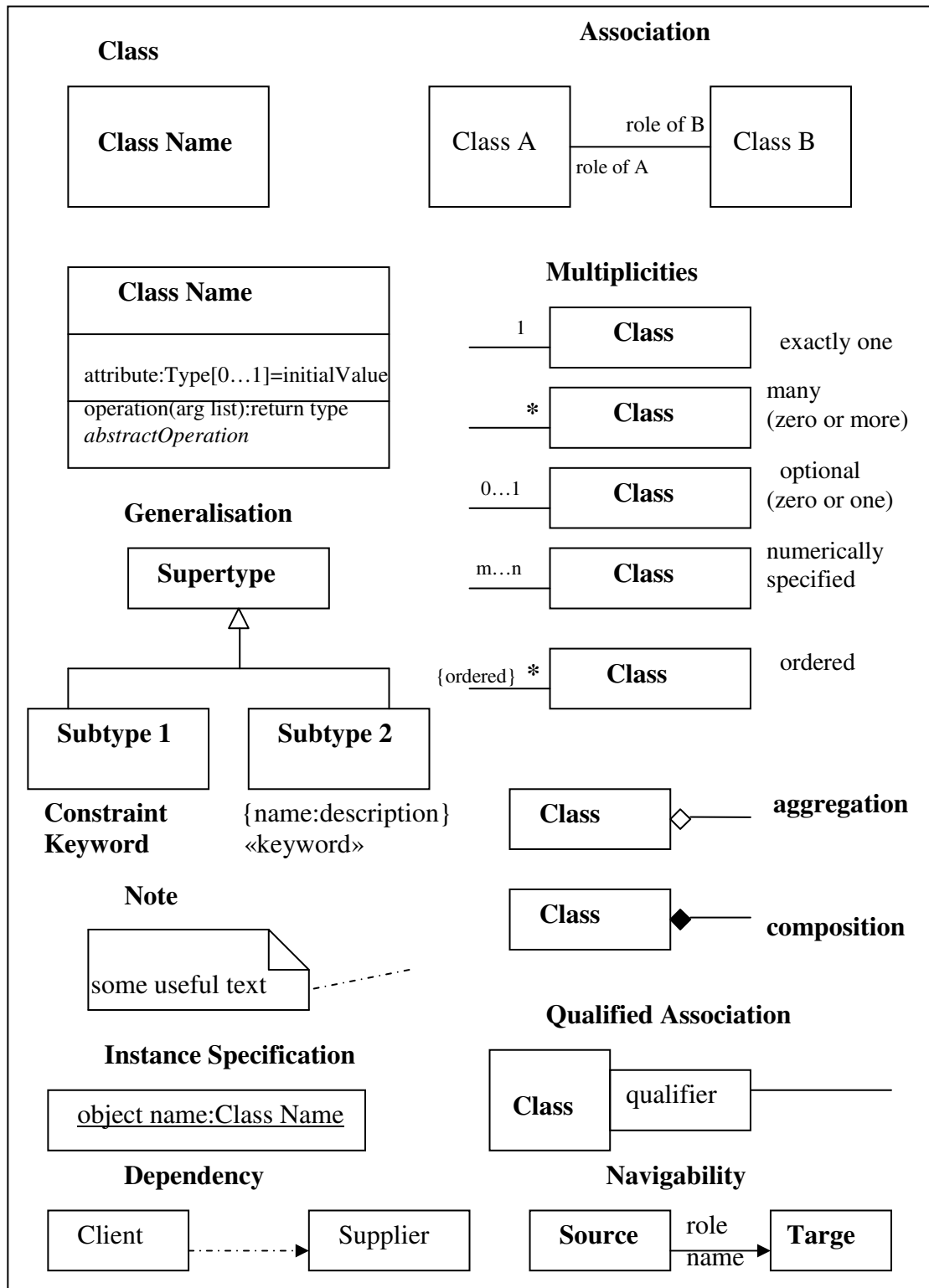
Uses relationship



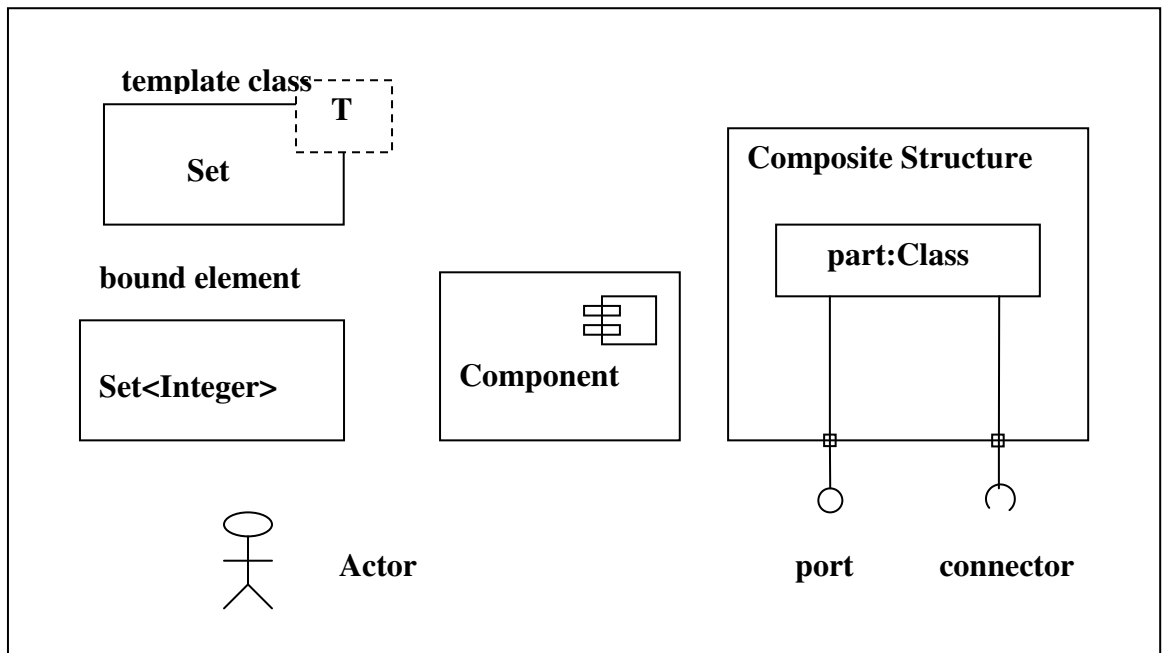
Include relationship



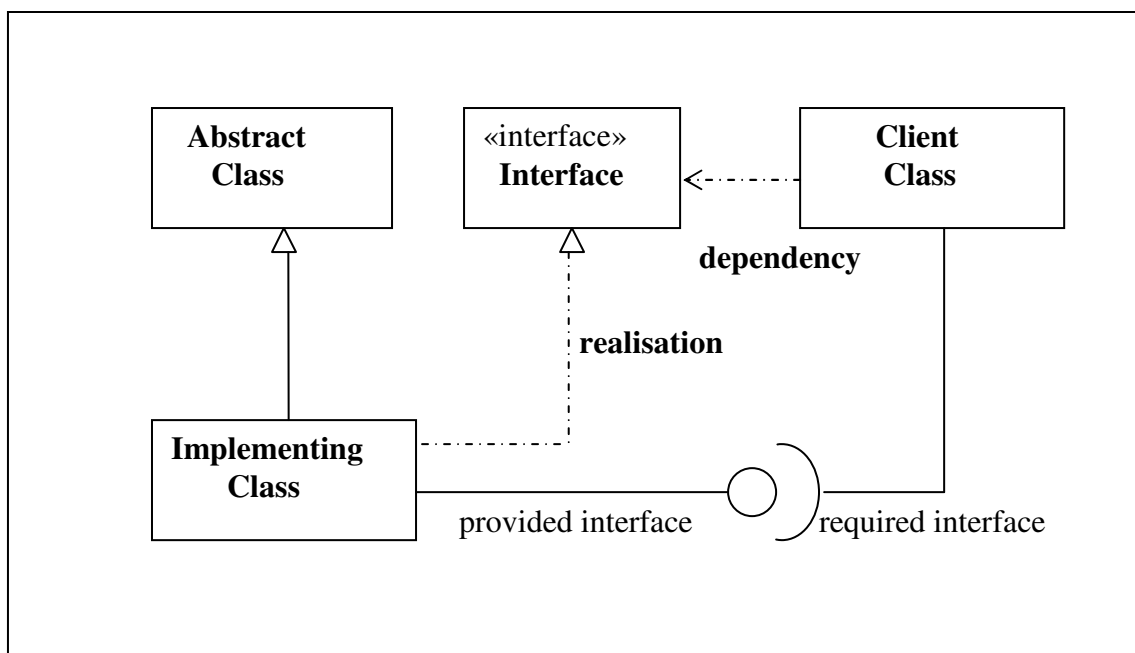
UML entities- the graphical notation of different UML entities from [UML2003]



UML entities- the graphical notation of different UML entities from [UML2003]

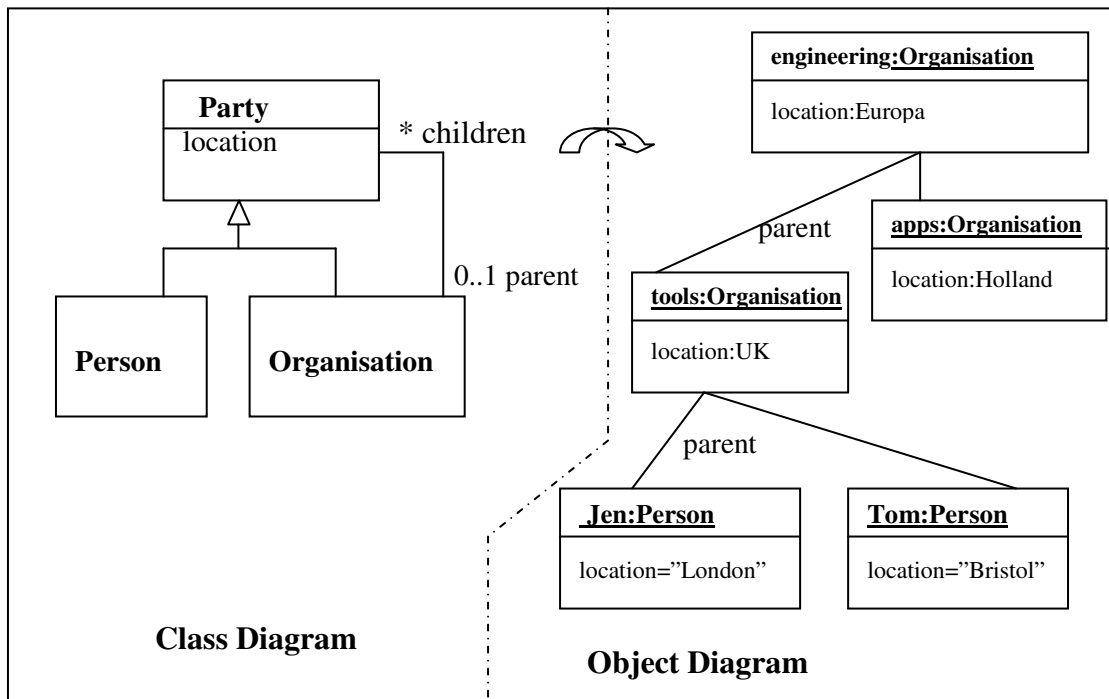


Class diagram- shows the properties and operations of a class and the constraints that apply to the connected objects.

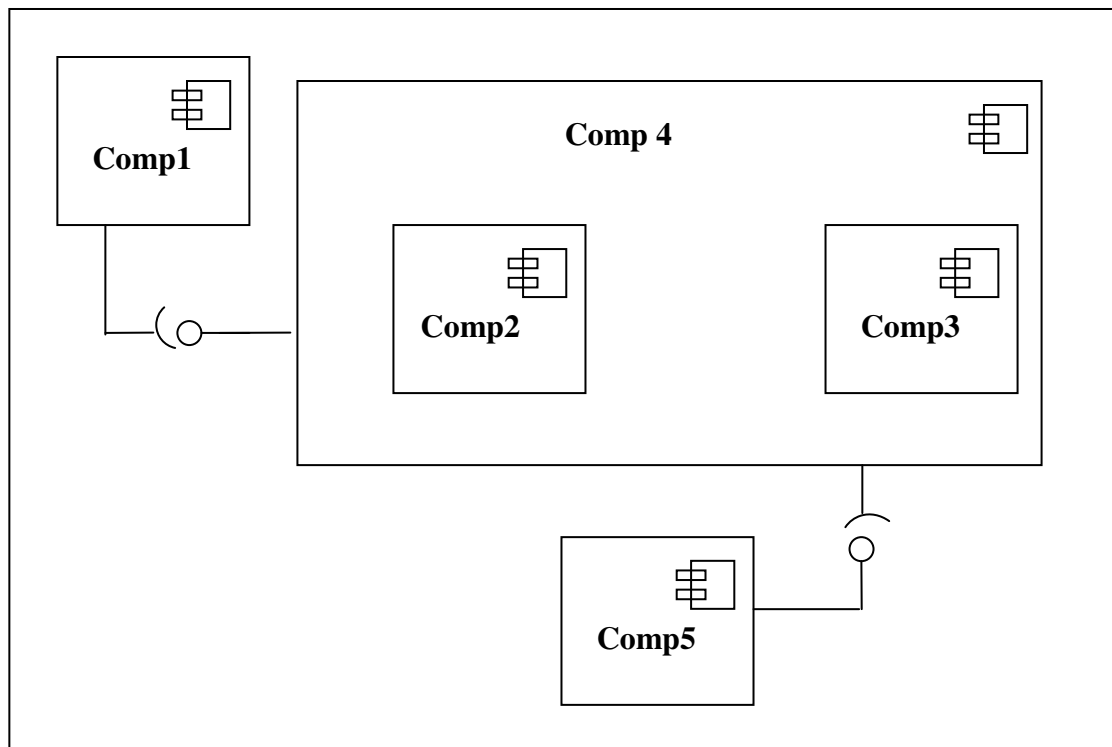


Object Diagram- illustrates a picture of the objects in a system at a point time.

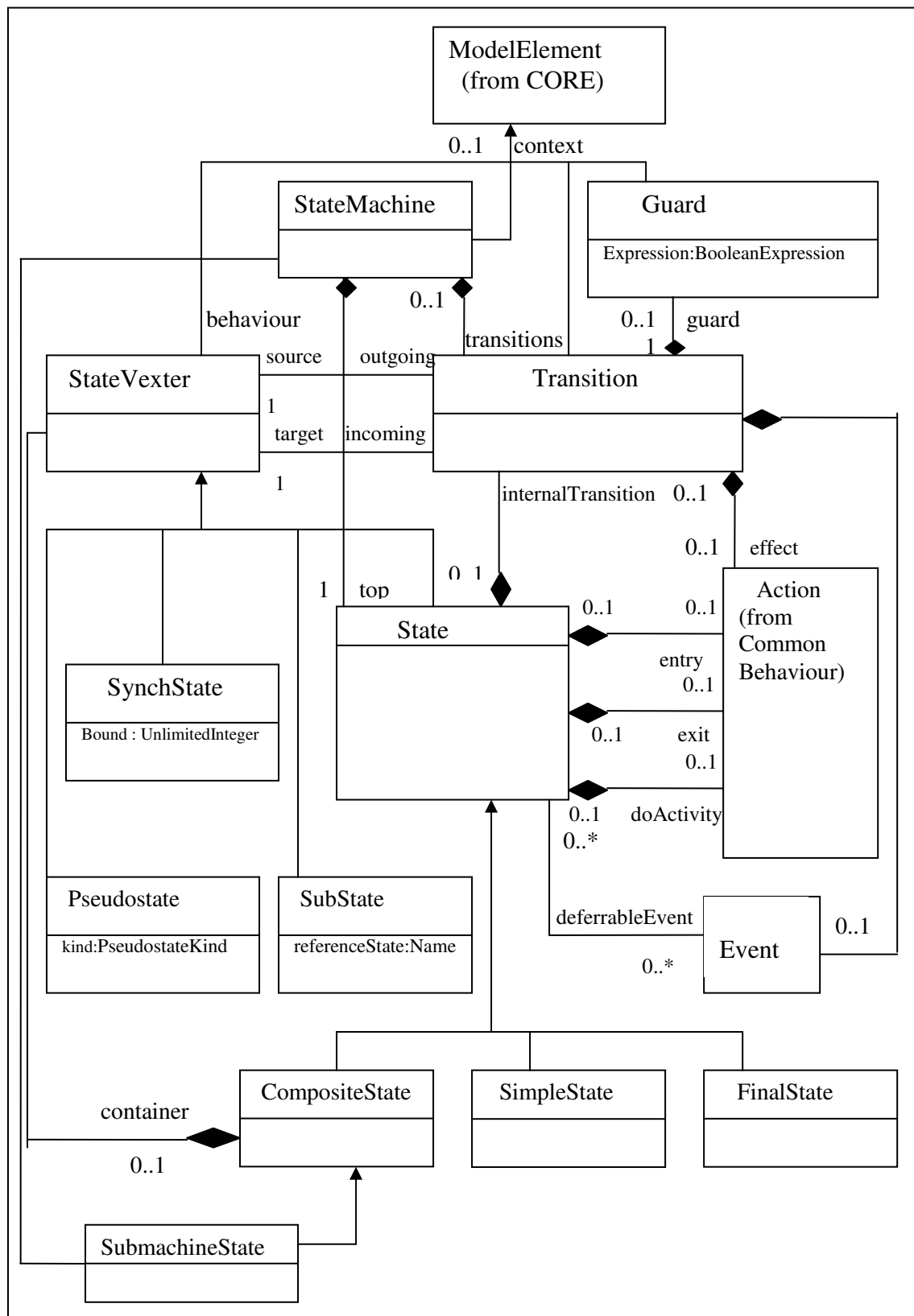
Example taken from [UML 2003,pp.88]



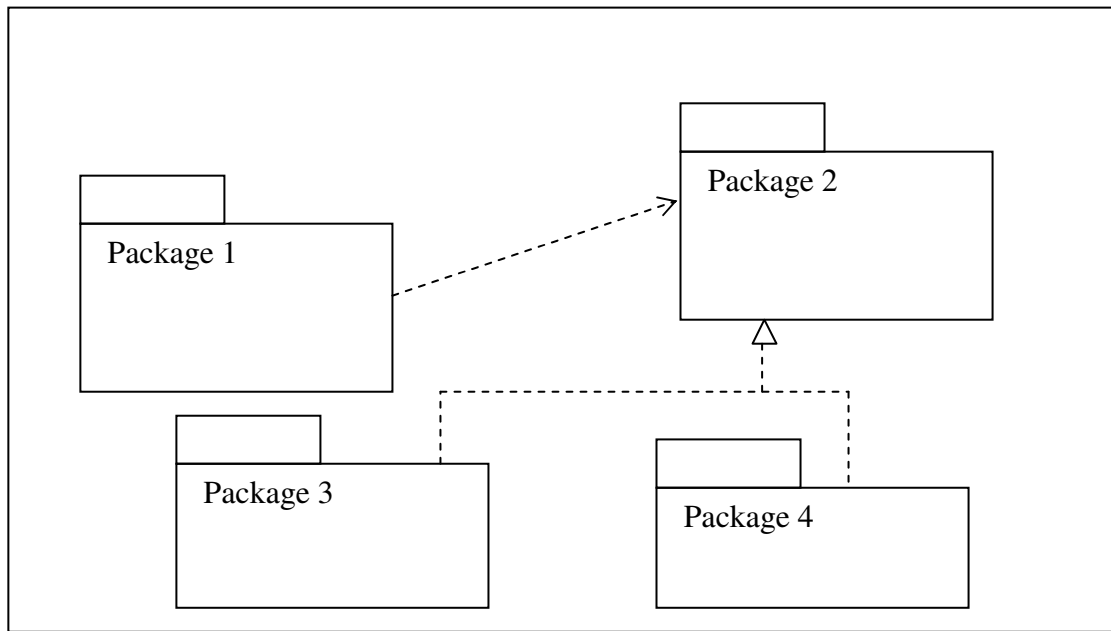
Component Diagram- shows the connection between the components



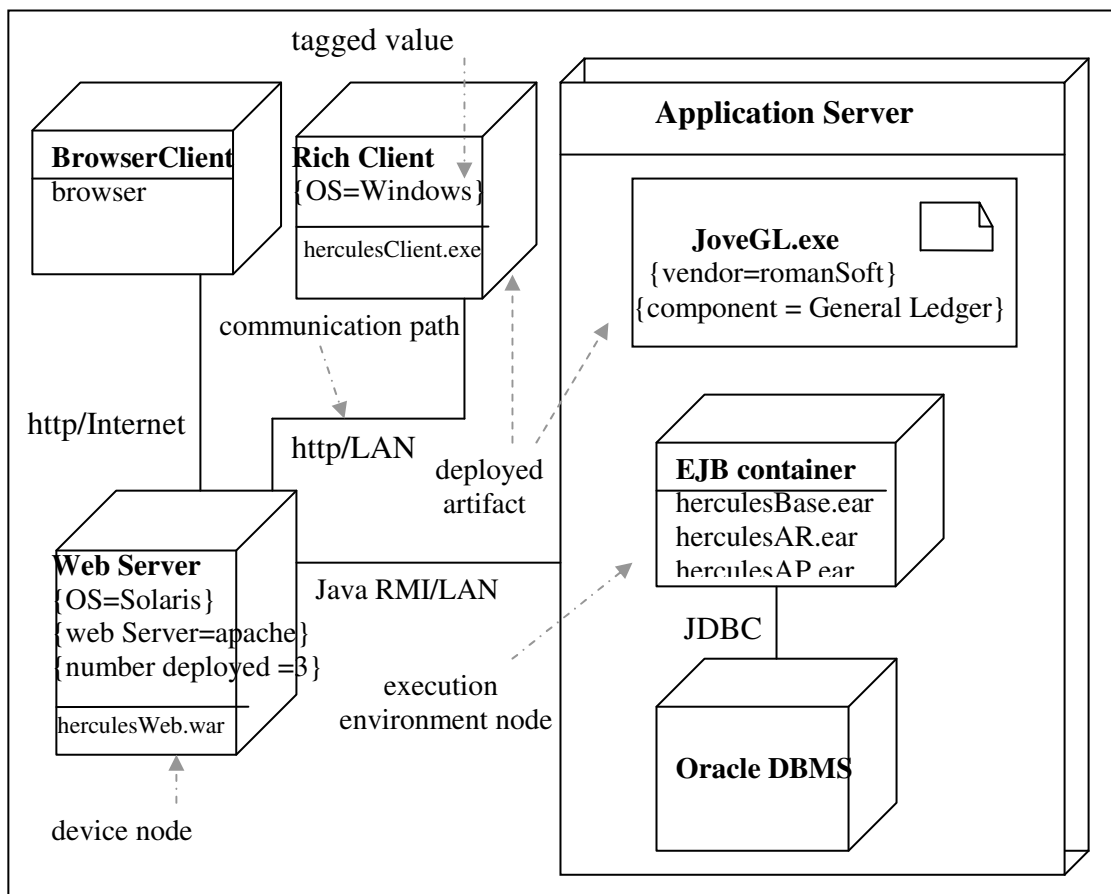
Statechart Diagram- Example taken from [UML 2 Metamodel2005,pp.129]



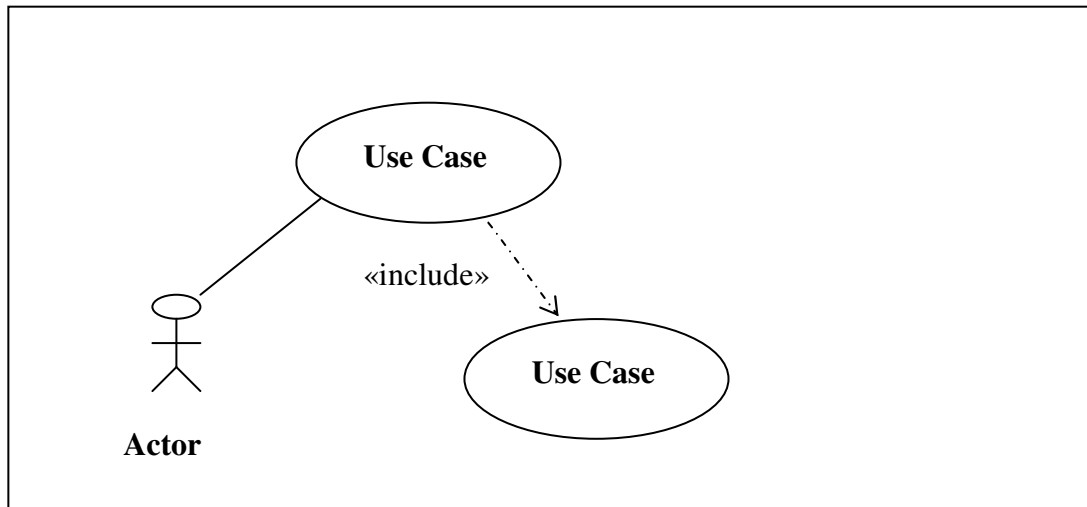
Package Diagram- Illustrates the packages and their dependencies



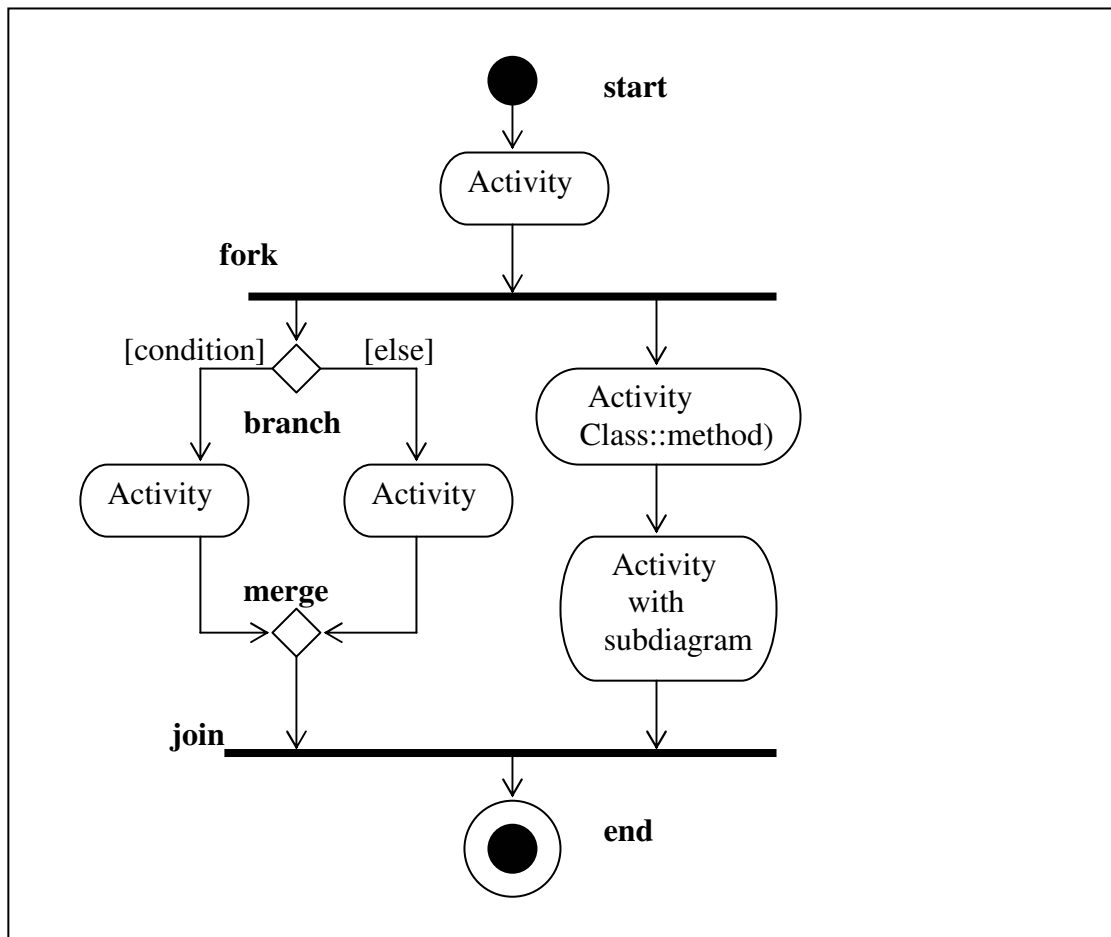
Deployment Diagram- illustrates which software pieces run on which hardware pieces. Example taken from [UML 2003, pp.98]



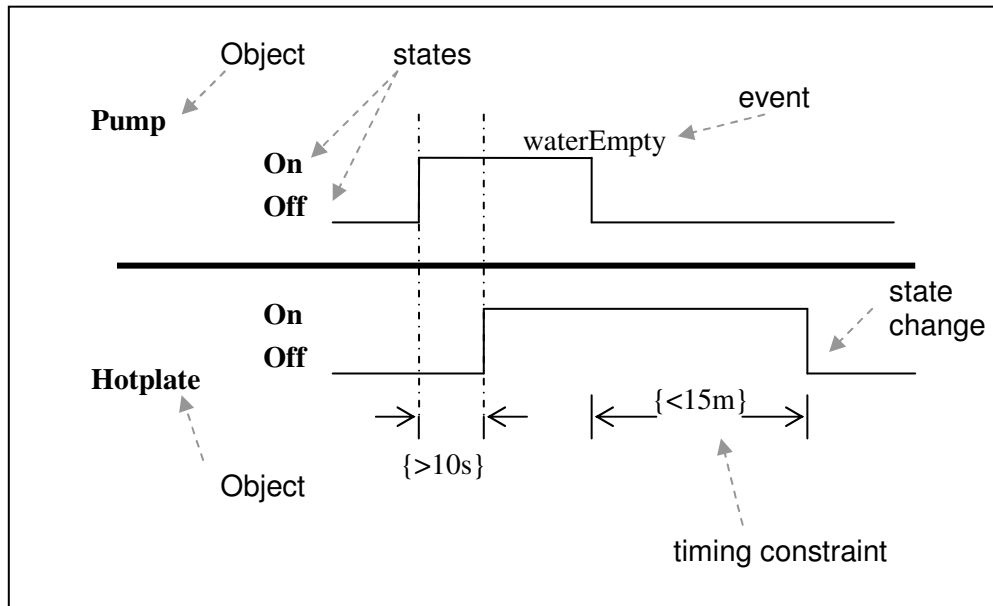
Use Case Diagram- illustrates the actors, the use cases, and the relationships among them.



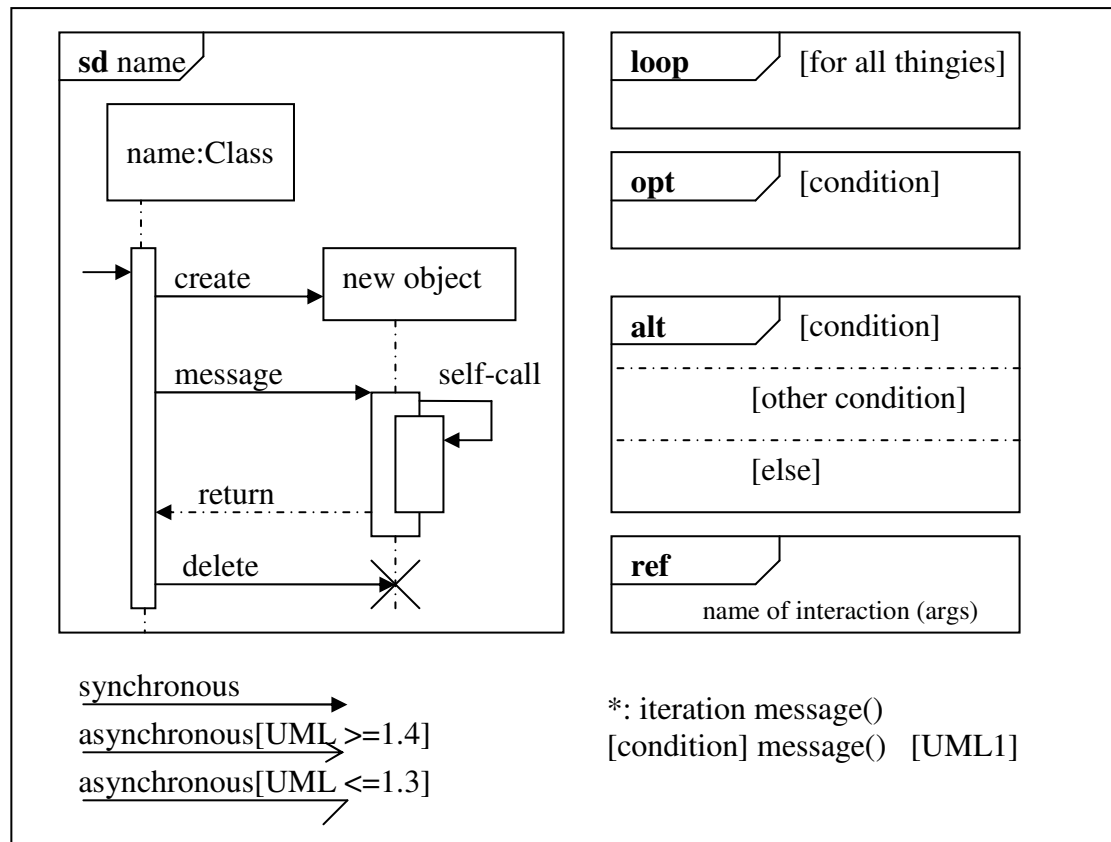
Activity Diagram- states the essential sequencing rules.



Timing diagram- shows the timing constraints between state changes either for a single object or for different objects. Example taken from [UML 2003, pp.150].



Interaction Diagram- illustrates how group of objects collaborate in some behaviour. The most common interaction diagram is called sequence diagram.



Appendix B

Capture module of ADVISOR

The capture module captures and digitises the video input data taken by cameras. It sub-samples and compresses the video information into JPEG format to maximise storage capability and adds time stamping information to the captured images. The capture module outputs, which are raw image sequences, are transmitted to several other components in the ADVISOR system. The capture functionality also includes a mode of operation that allows playback of previously captured video sequences into the system from the hard drive.

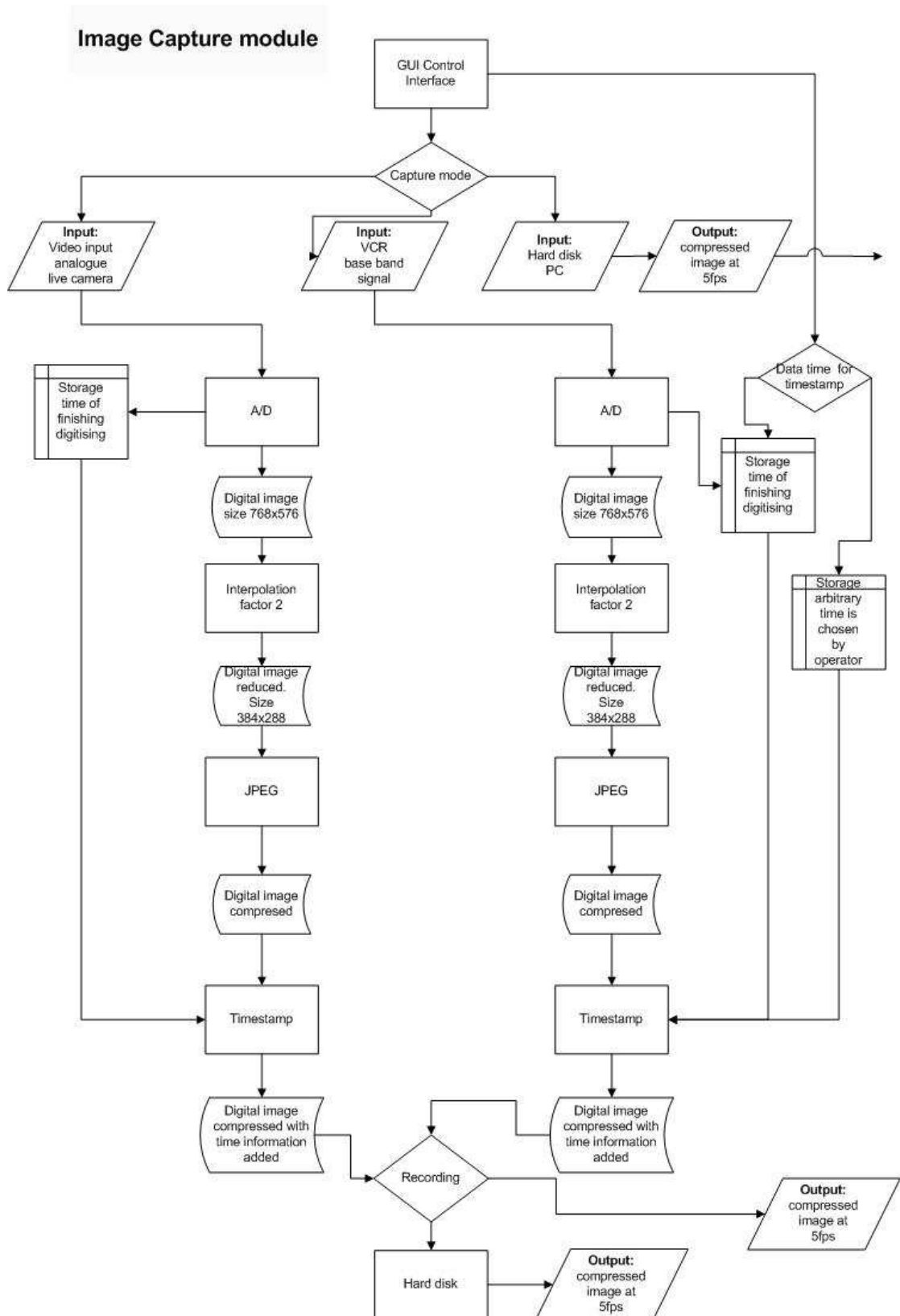


Figure B- 1 Diagrammatic representation of the low level design of the capture module.

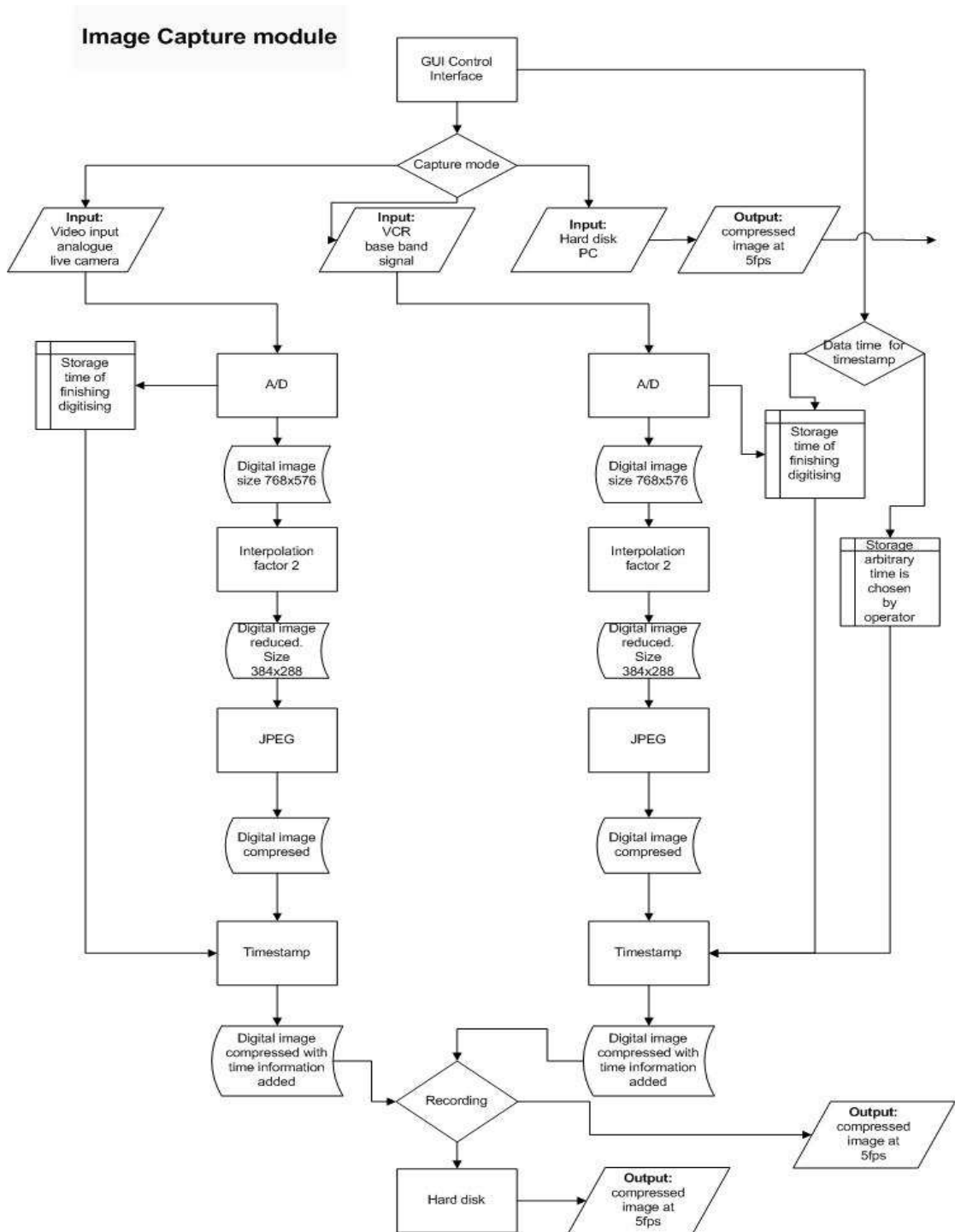


Figure B- 1. Capture module

Motion detection module of ADVISOR

The Motion Detector submodule detects image changes when objects move and generates a description for each moving region. The Motion Detector module also classifies the moving regions into mobile object classes such as *Person*, *Group*, *Metro-train* and *Noise*.

The Motion Detector component can be viewed as a two-stage algorithm. The first stage separates an image sequence into a ‘relatively’ stable *background* and a varying *foreground* overlay. The second stage identifies the moving regions belonging to the *foreground* overlay and classifies them in a predefined mobile object class as described previously. The identified moving objects are framed by fitting ‘blobs’ around the group of pixels belonging to the same moving object. For each input image, the motion detector submodule generates three outputs:

- The background image: this module performs the task of upgrading the background. The upgraded background is transmitted infrequently (the frame rate was one image per minute per camera) because of the relatively low changing nature of the image intensities.
- The foreground image: this output was transmitted at the same rate as the incoming image sequences per camera, i.e. 5fps.
- Blob descriptions: one set per incoming image per camera, which is sent in XML format.

Figure B- 2 shows a diagram for the motion detection module. Figure B- 3, 4, 5 and Figure B- 6 illustrate the diagrams of moving regions detection algorithms, for the update reference image algorithm and for the merging of detected moving regions and reclassification algorithm respectively.

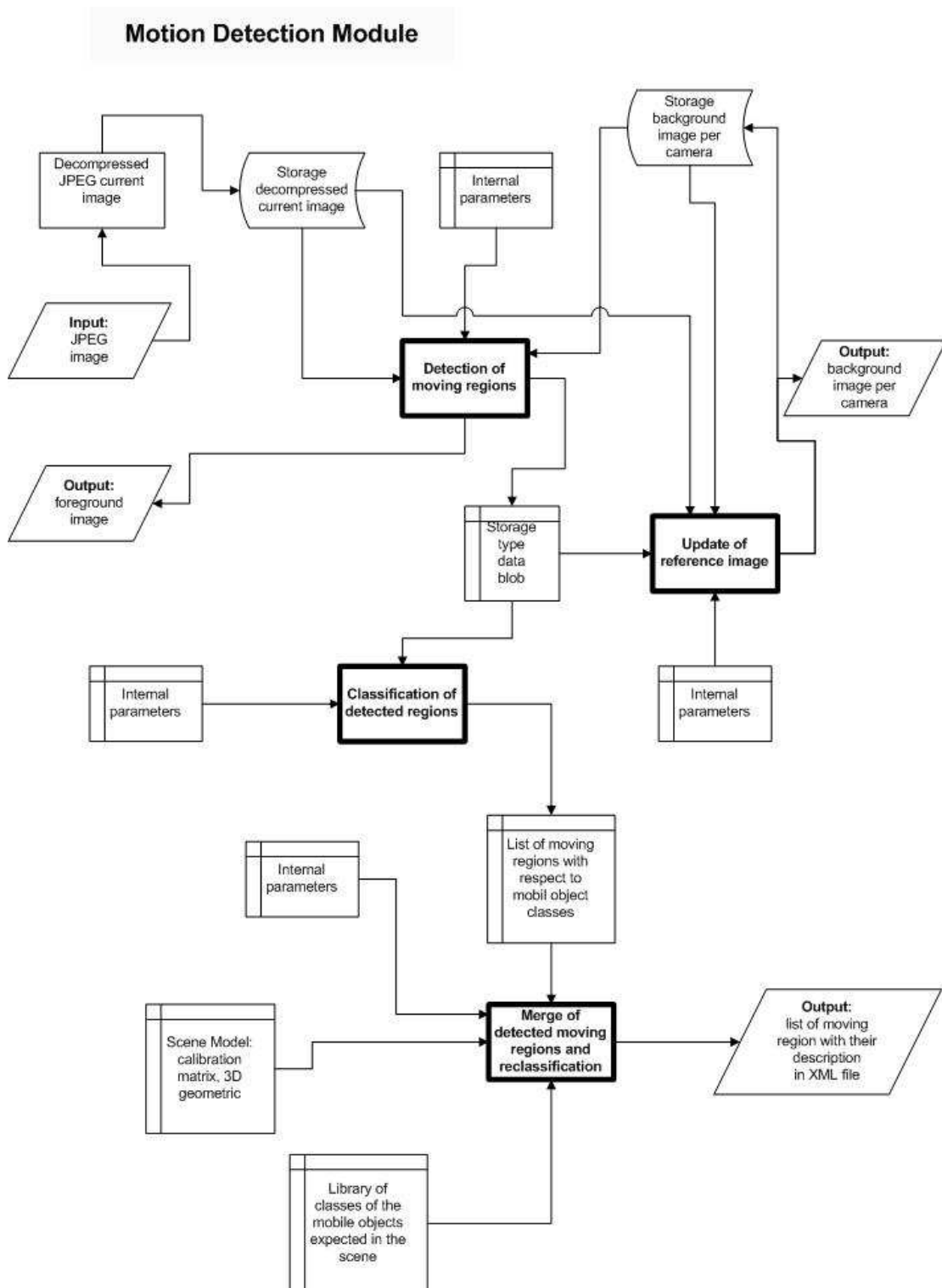


Figure B- 2. Diagram of the motion detection module (extracted from the ADVISOR specification documents [ADVISOR 2003]).

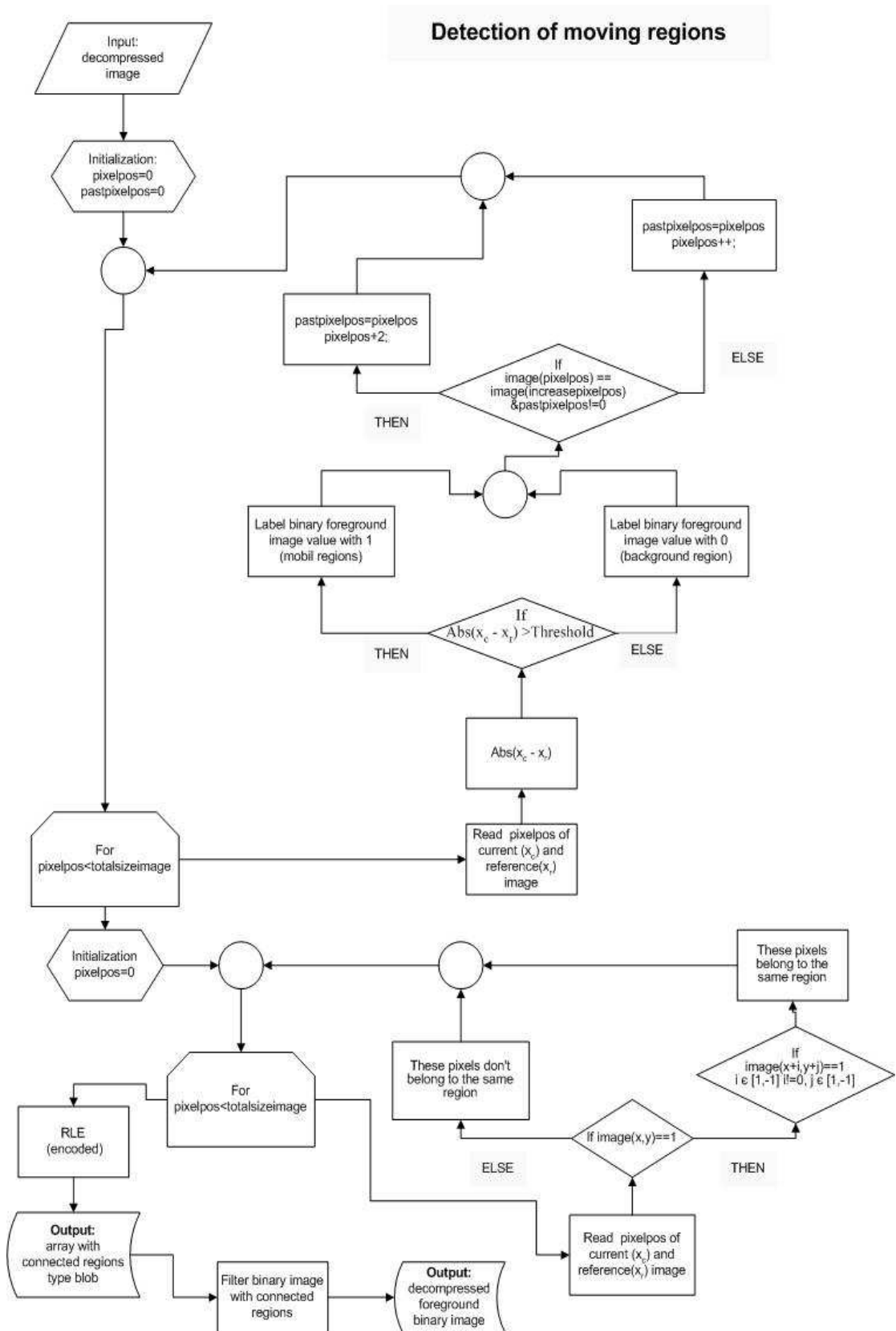


Figure B- 3.Diagram of the submodule for the detection of moving regions.

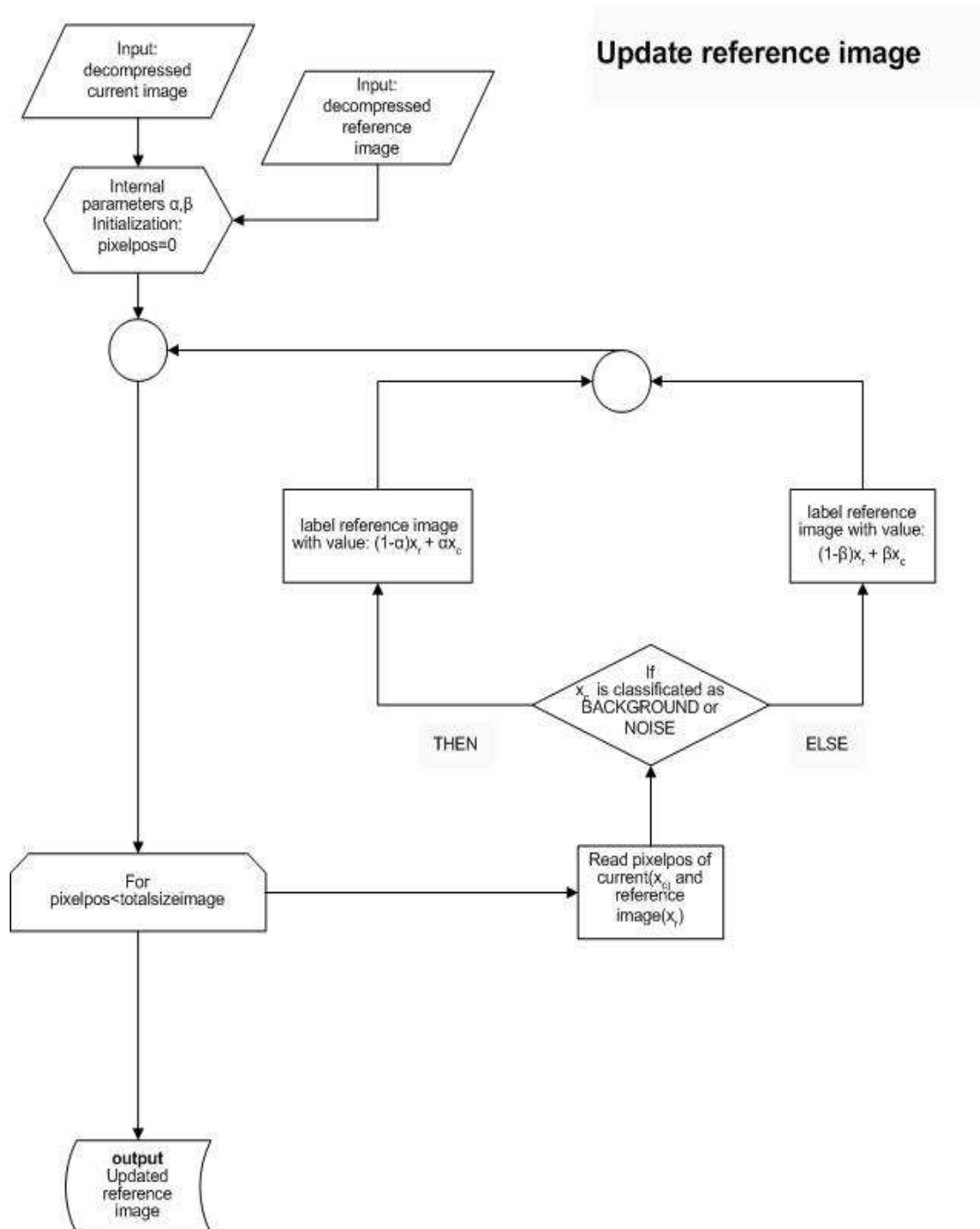


Figure B- 4. Diagram of the submodule to Update the reference image (or commonly called “background” in computer vision).

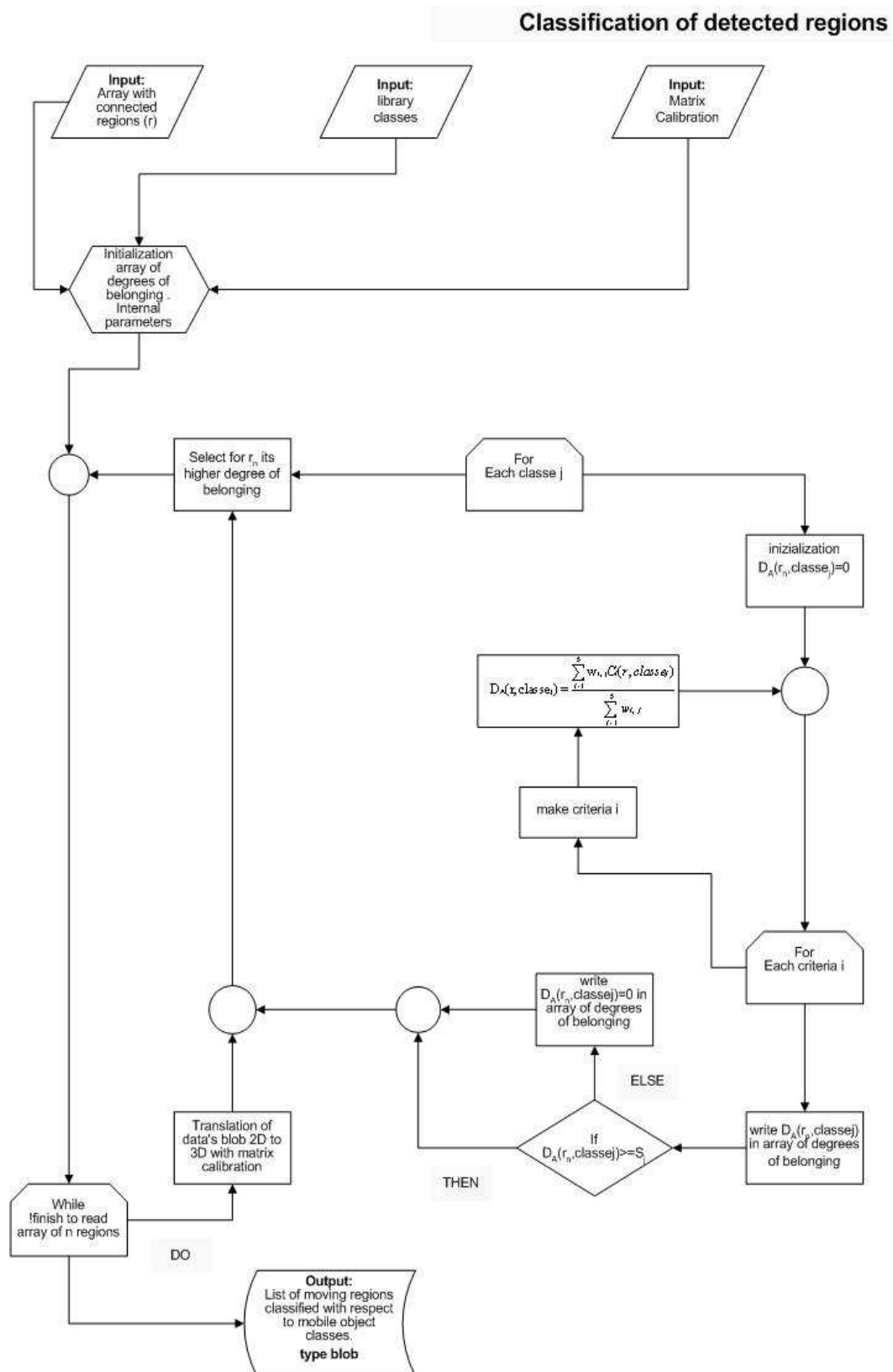


Figure B- 5. Diagram of the submodule for the classification of detected regions.

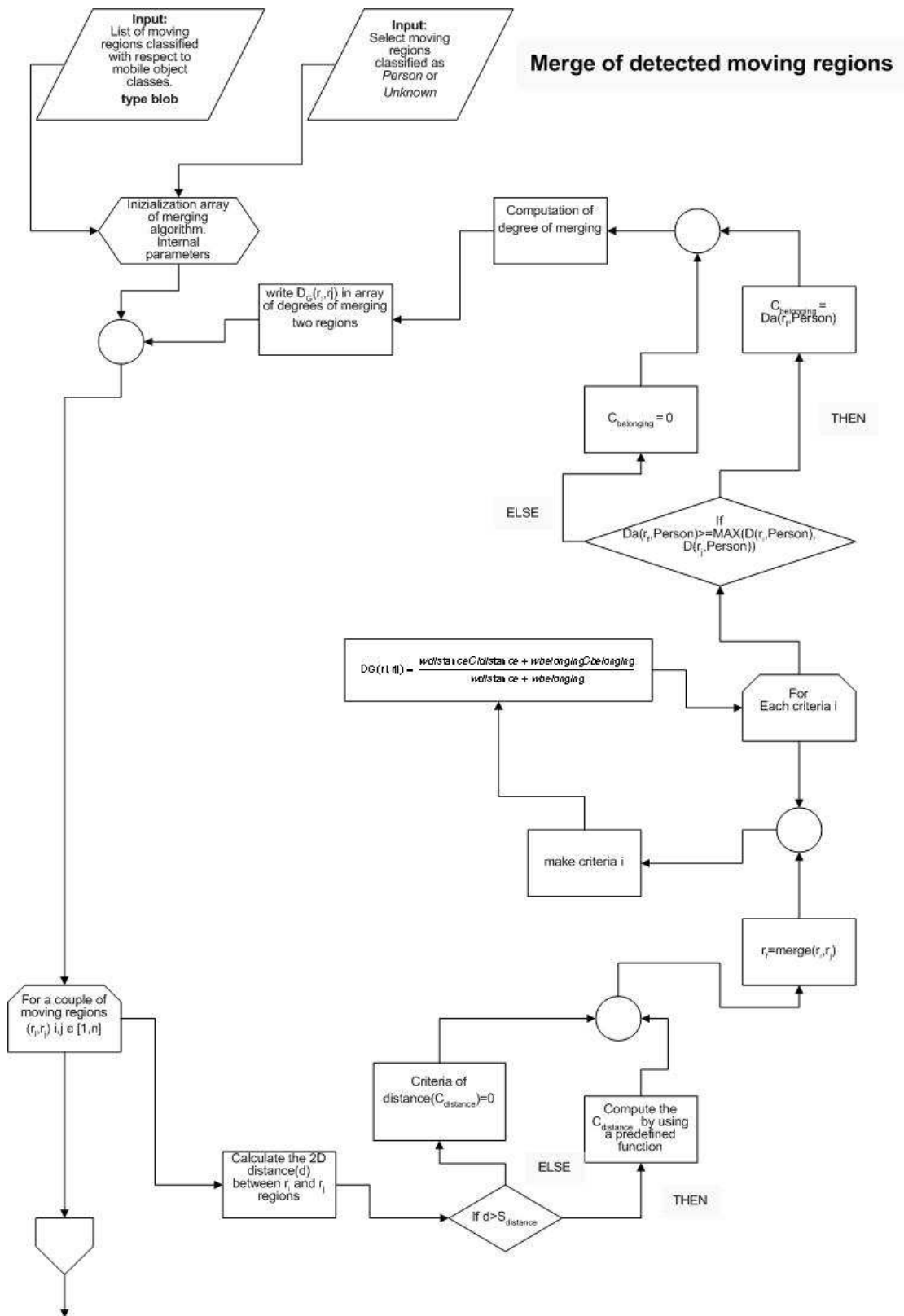


Figure B- 6. Diagram of the submodule to merge detected moving regions.

Crowd Monitor module of ADVISOR

The ADVISOR crowd monitoring module measures crowd related properties such as direction of flow and density and motion, based on the video images that are sent by the capture module over the LAN. The crowd monitor module attempts to detect specific potentially dangerous situations. This module is designed to deal with four areas of abnormal behaviour based on the motion information extracted by a motion detection board (hardware called STM1300 and based on a TriMedia 1300 Digital Signal Processor):

- Unusual or forbidden direction of motion.
- Objects that are stationary for unusually long periods of time.
- Individuals inside a forbidden area.
- Overcrowding detection.

The detection of any of these situations makes the crowd module generate and send a message in XML format to the Behaviour Analysis module which generates an alarm. The Behaviour Analysis module then deals with the event, including the fusion with other events if necessary, and routes the event to other relevant parts of the system such as the HCI or the archive. Figure B- 7, 8, 9 and Figure B- 10 present diagrams of the module called Crowd Monitor Module. This module, as seen in Figure B- 7, consists of two main submodules: one of them is called the Load Motion program and is illustrated in Figure B- 8. One of the key functions of the DSP in the STM1300 card is to compute image motion vectors in real-time.. Therefore, the output from this card is sent to another submodule (see Figure B- 9 and Figure B- 10) that together with some thresholds is able to detect the pre-defined events which have been listed above.

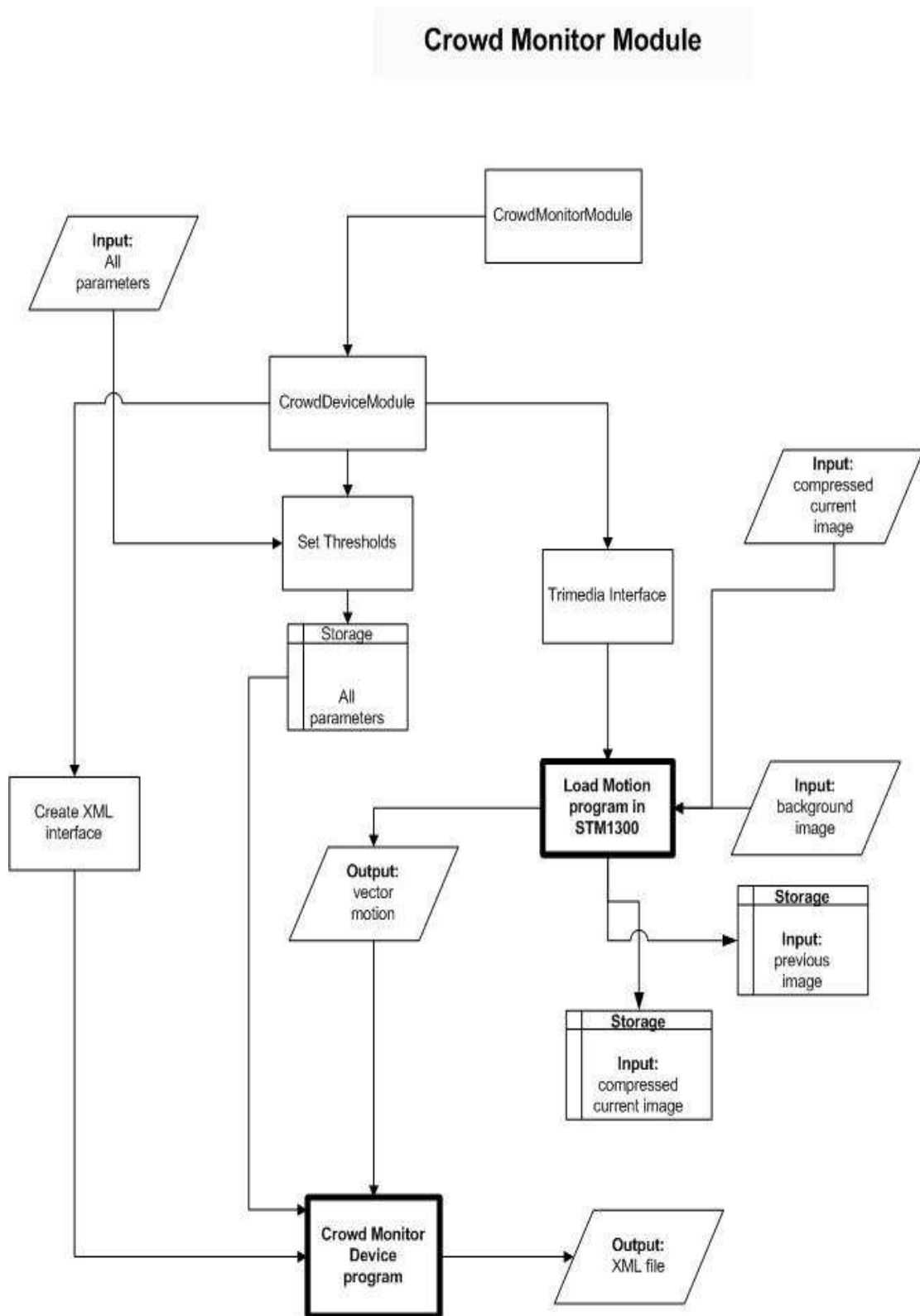


Figure B- 7. Diagram of the Crowd Monitor Module, which has two main submodules: Crowd Device program and Load Motion program in the STM1300 card.

Motion Detection Program Load in STM1300 board

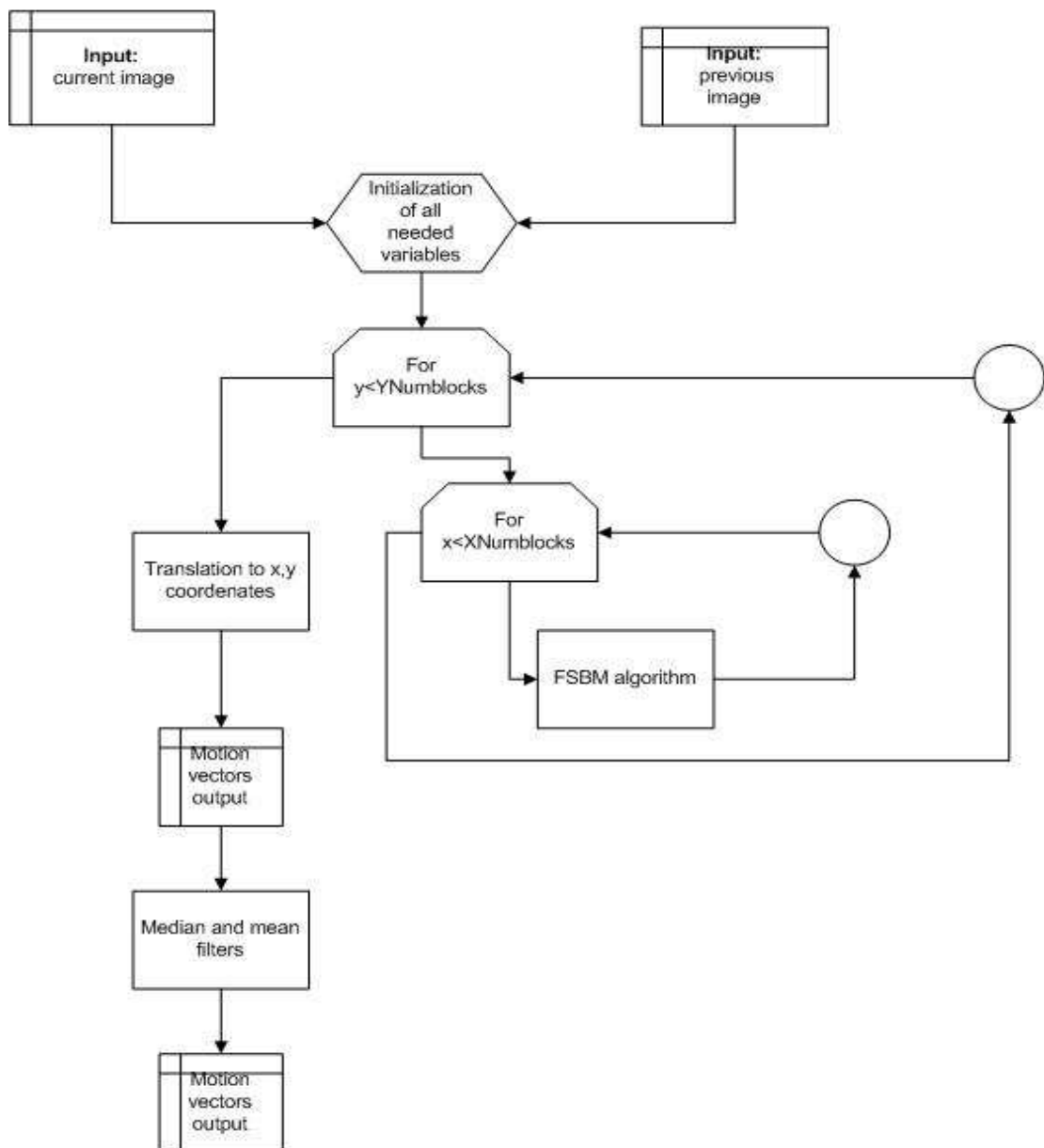


Figure B- 8. Diagram of the submodule called Load Motion Program.

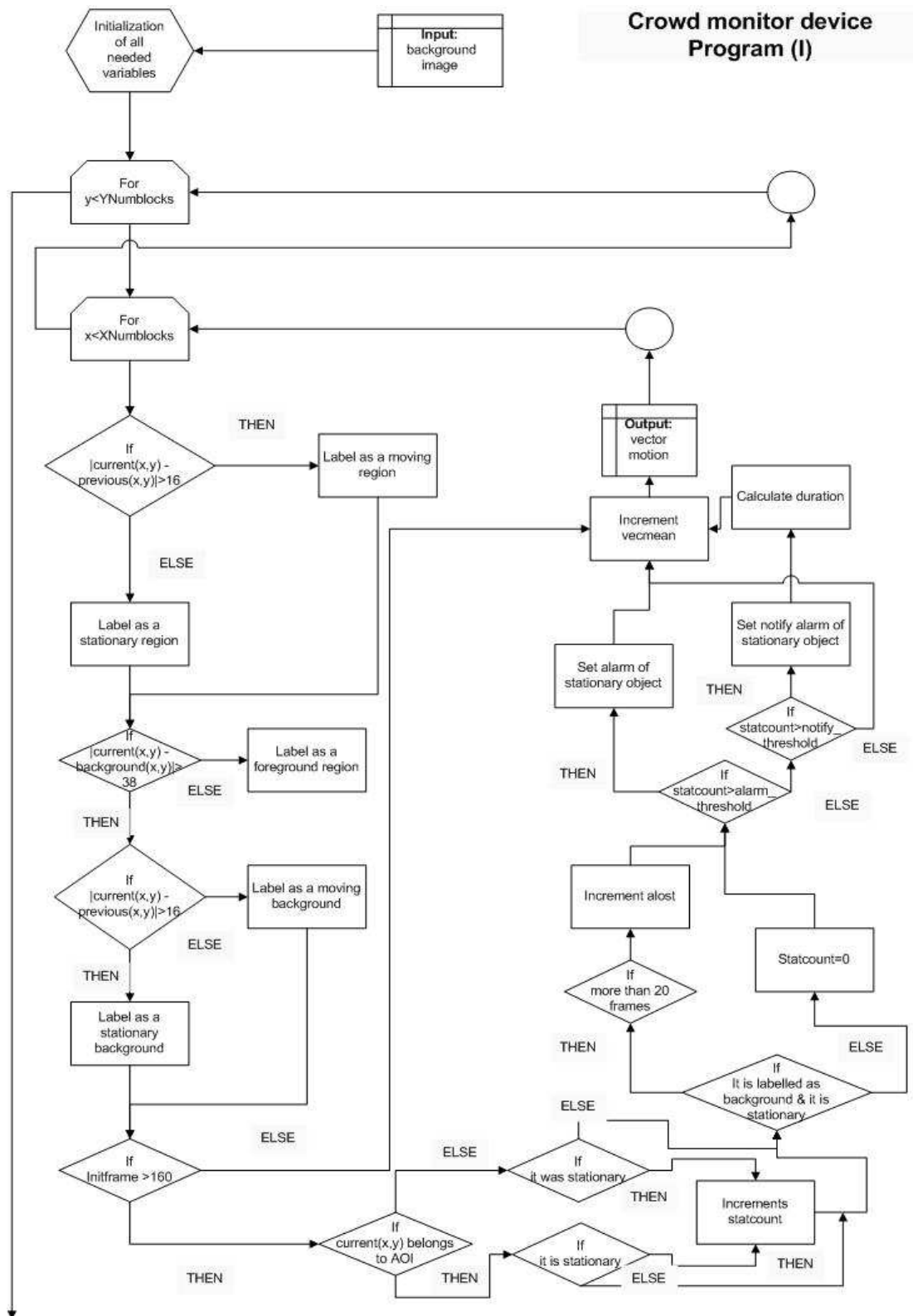


Figure B- 9. First part of the diagram corresponding to the submodule called Crowd Monitor Device program.

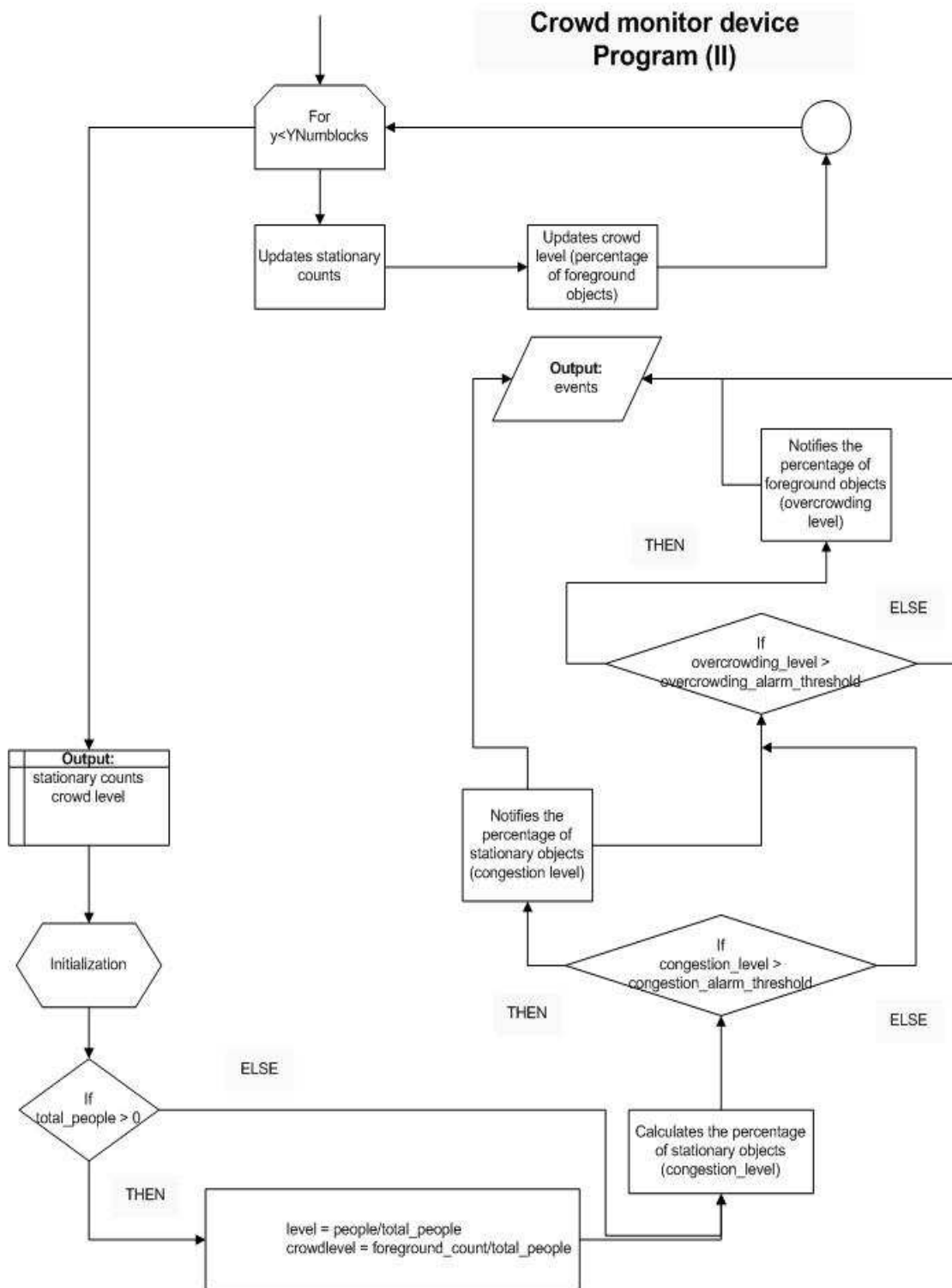


Figure B- 10. Second part of the diagram that describes the submodule called Crowd Monitor Device program.

Appendix C

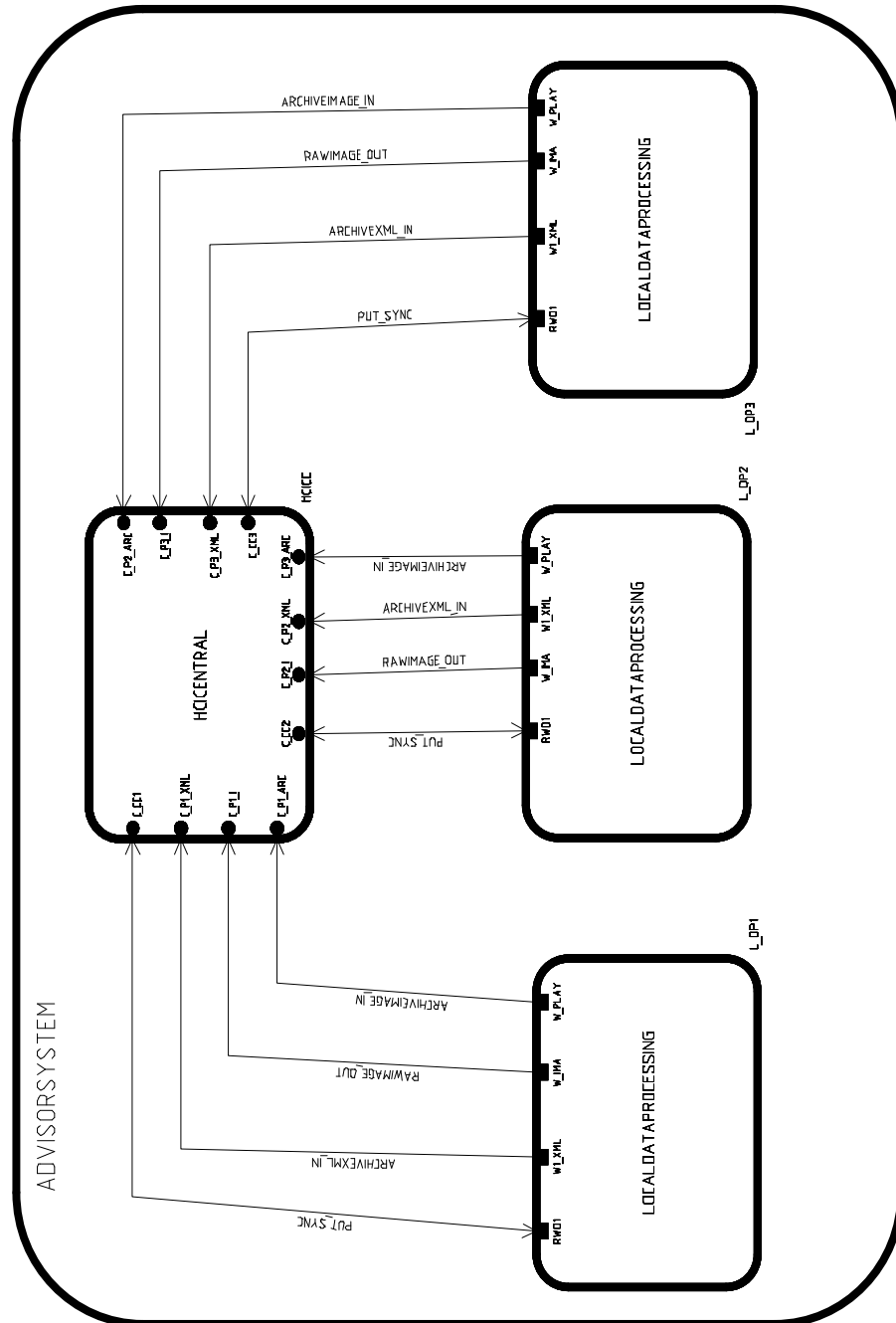


Figure C-1. ADVISORSYSTEM

Figure C-2. HCICENTRAL.

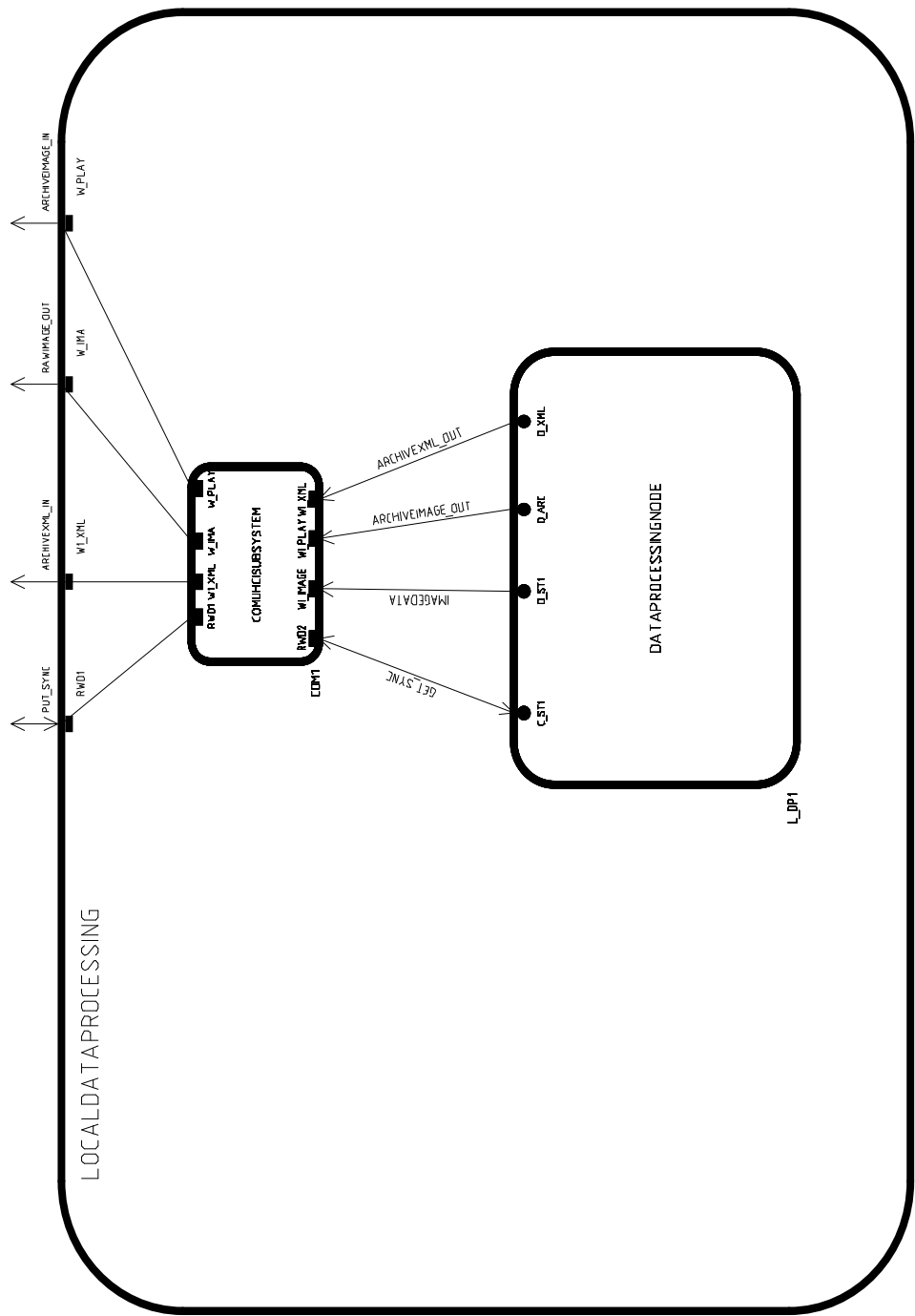


Figure C-3. LOCALDATAPROCESSING.

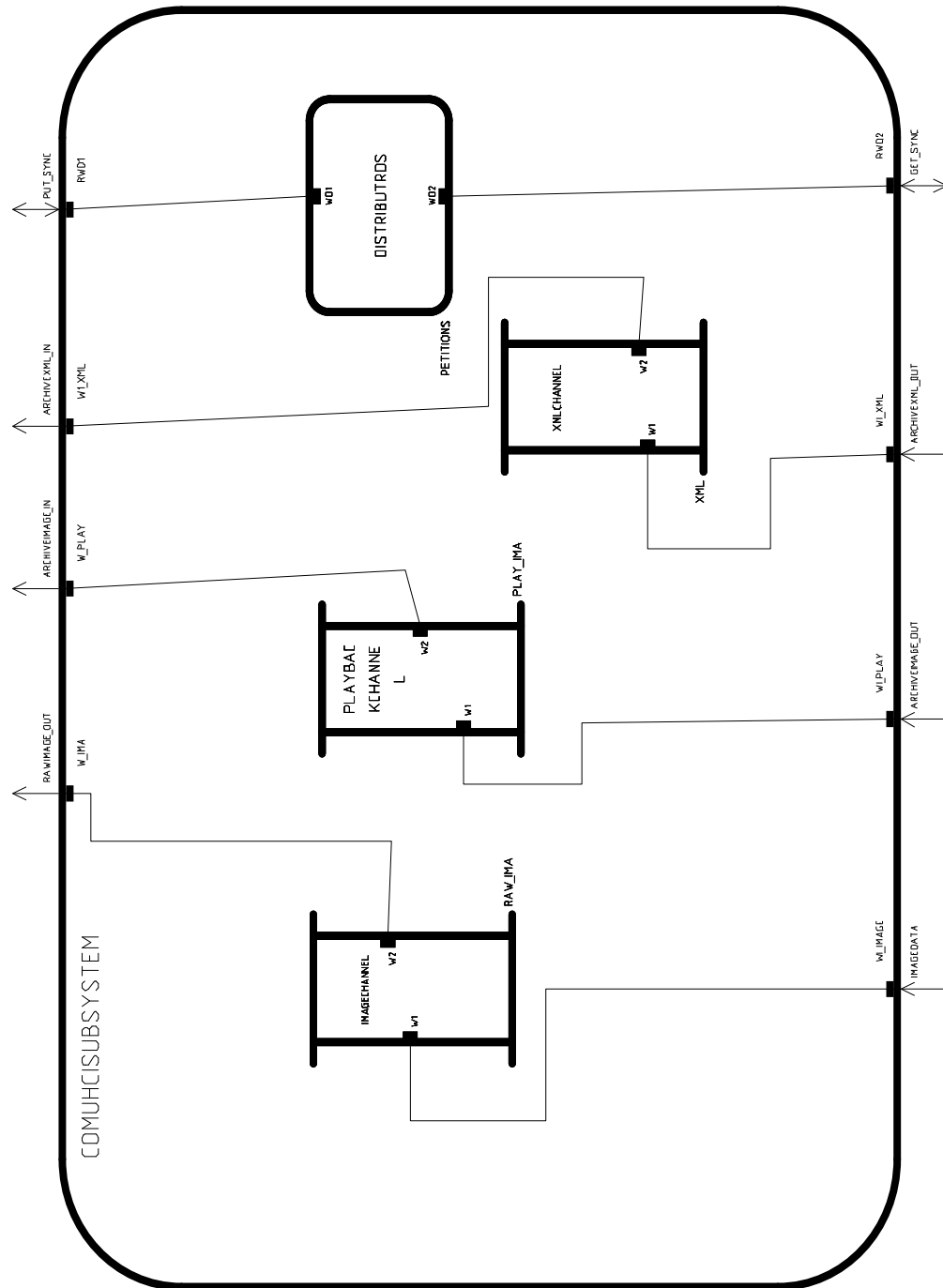


Figure C-4. COMUHCISUBSYSTEM.

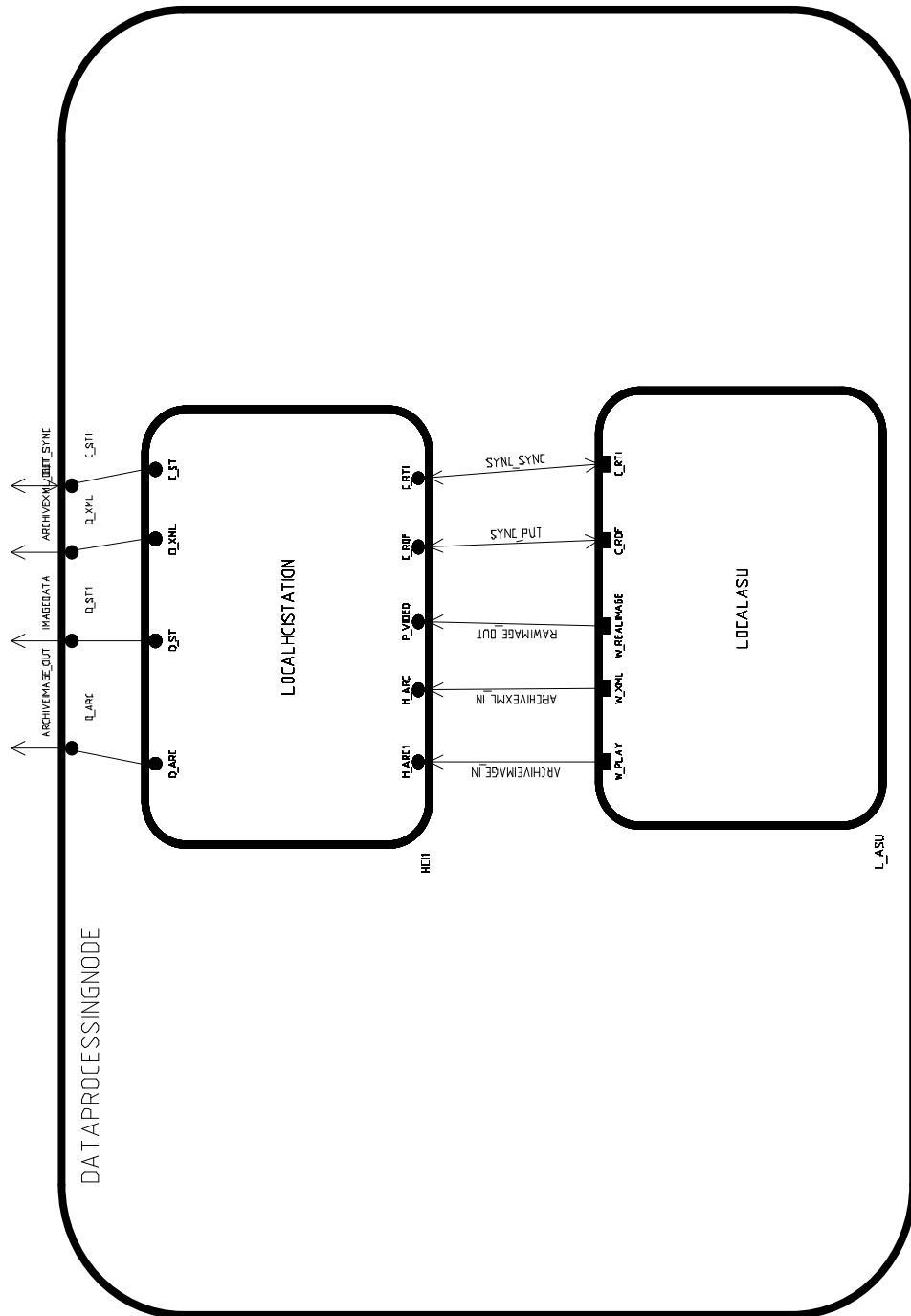


Figure C-5. DATAPROCESSINGNODE.

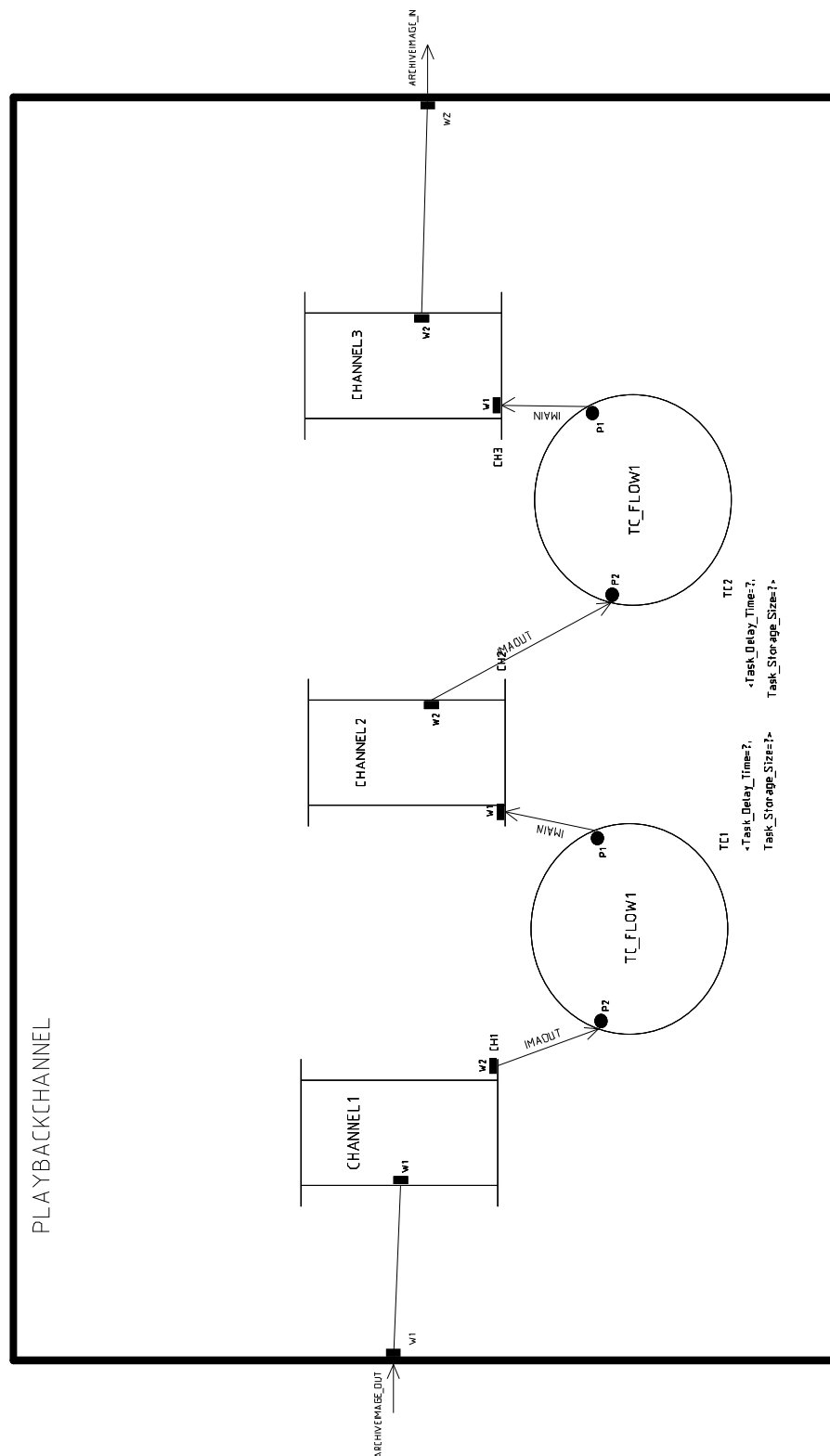


Figure C-6. PLAYBACKCHANNEL.

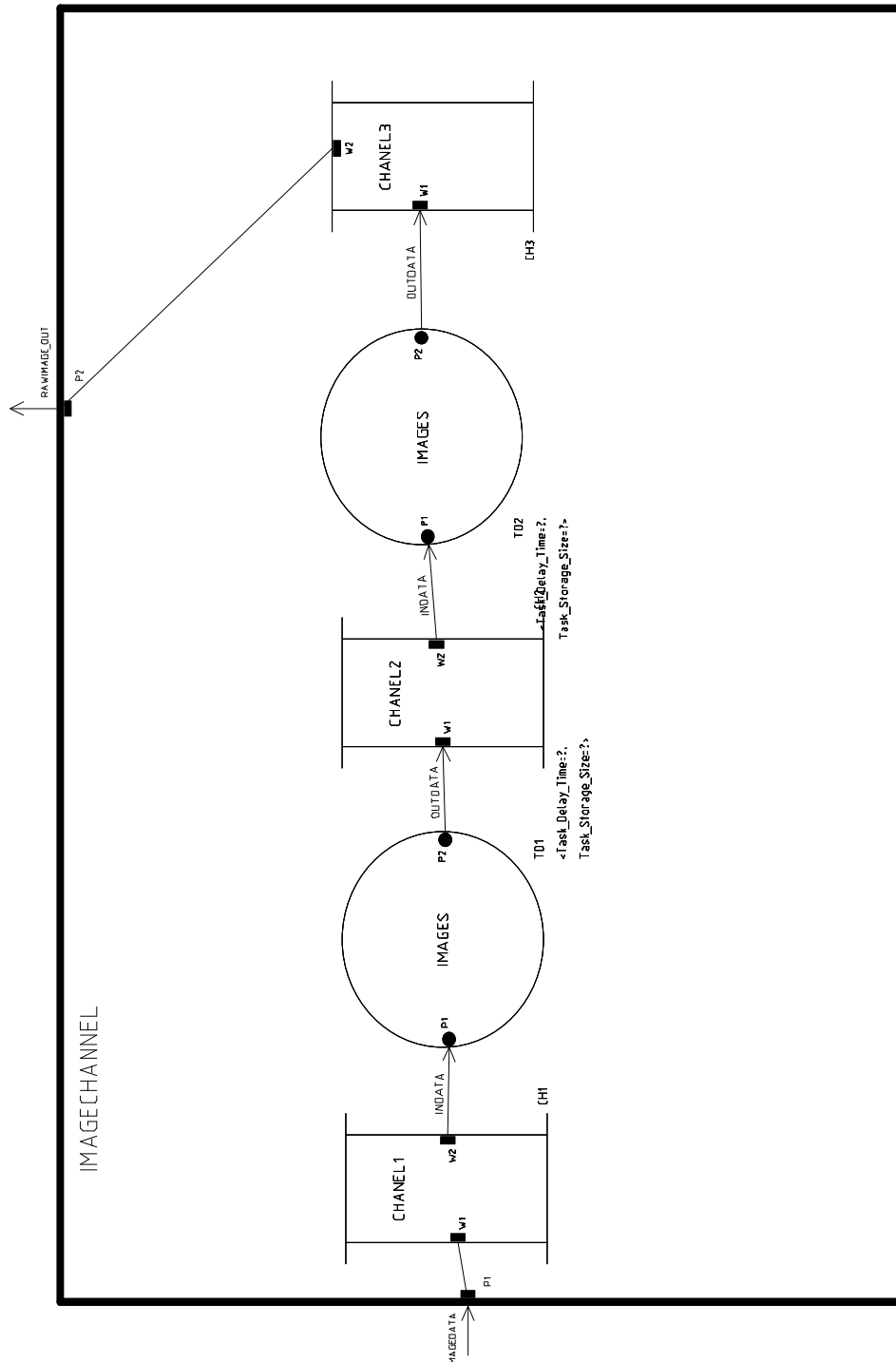


Figure C-7. IMAGECHANNEL.

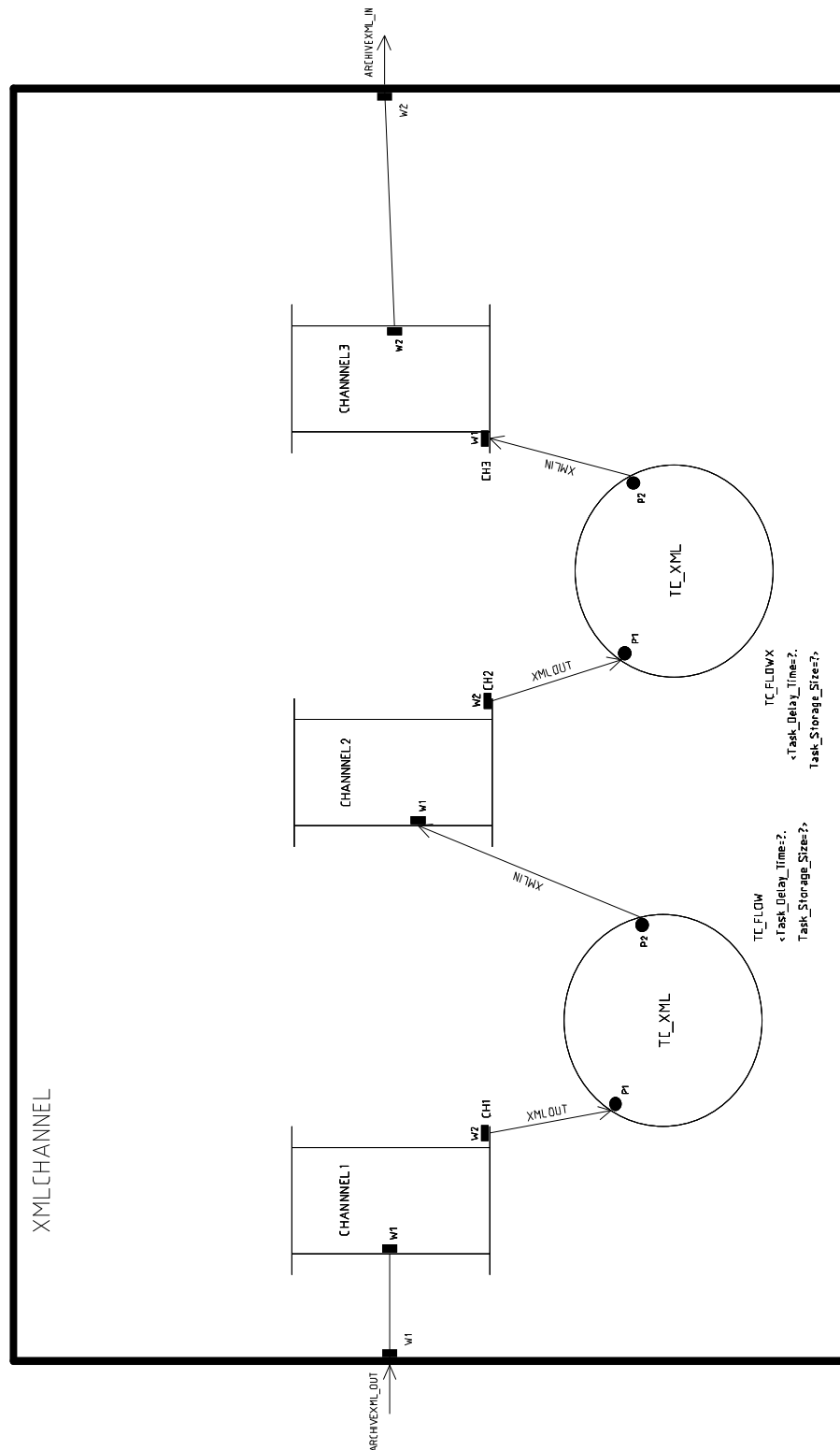


Figure C-8. XMLCHANNEL.

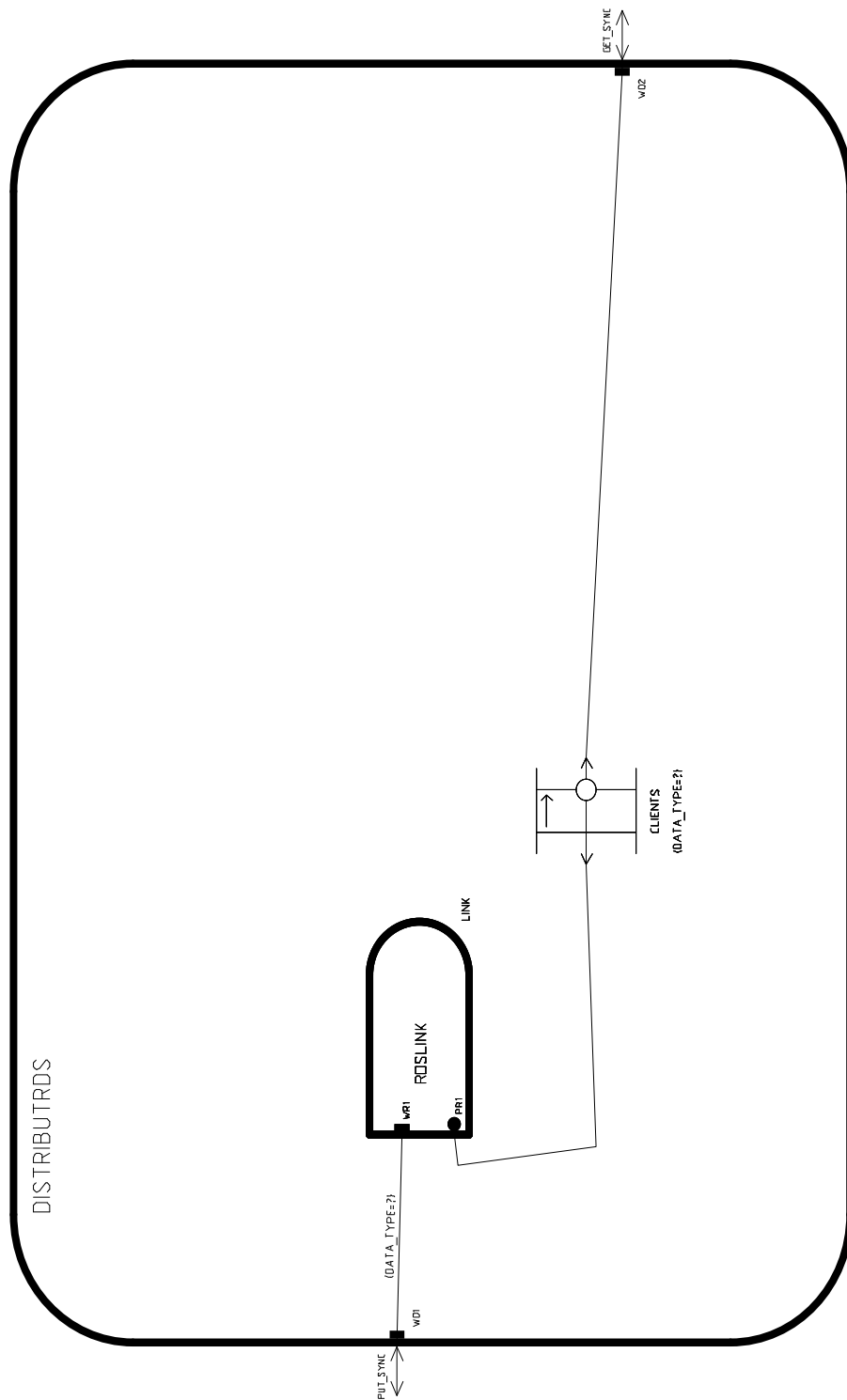


Figure C-9. DISTRIBUTRDS.

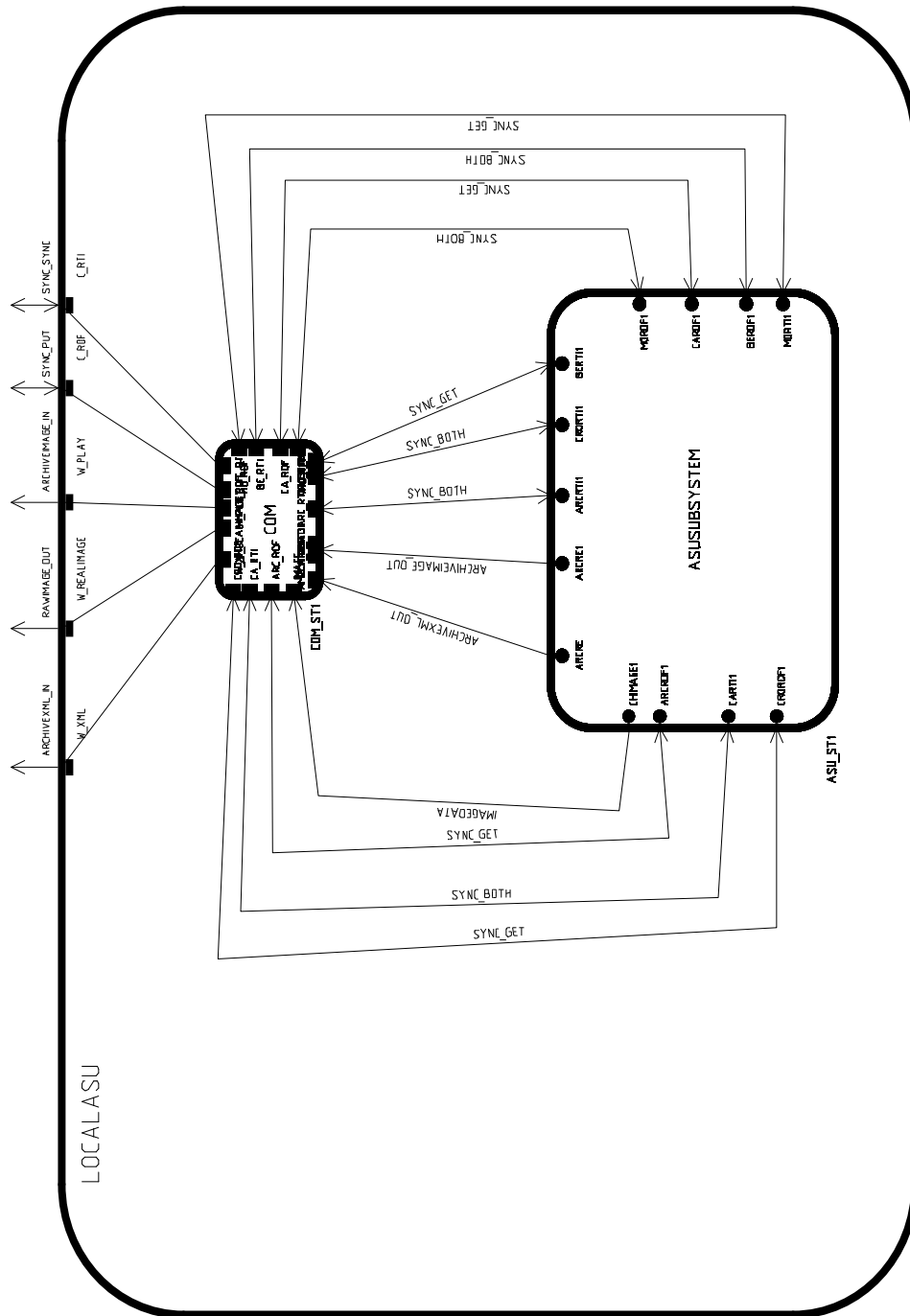


Figure C-10. LOCALASU.

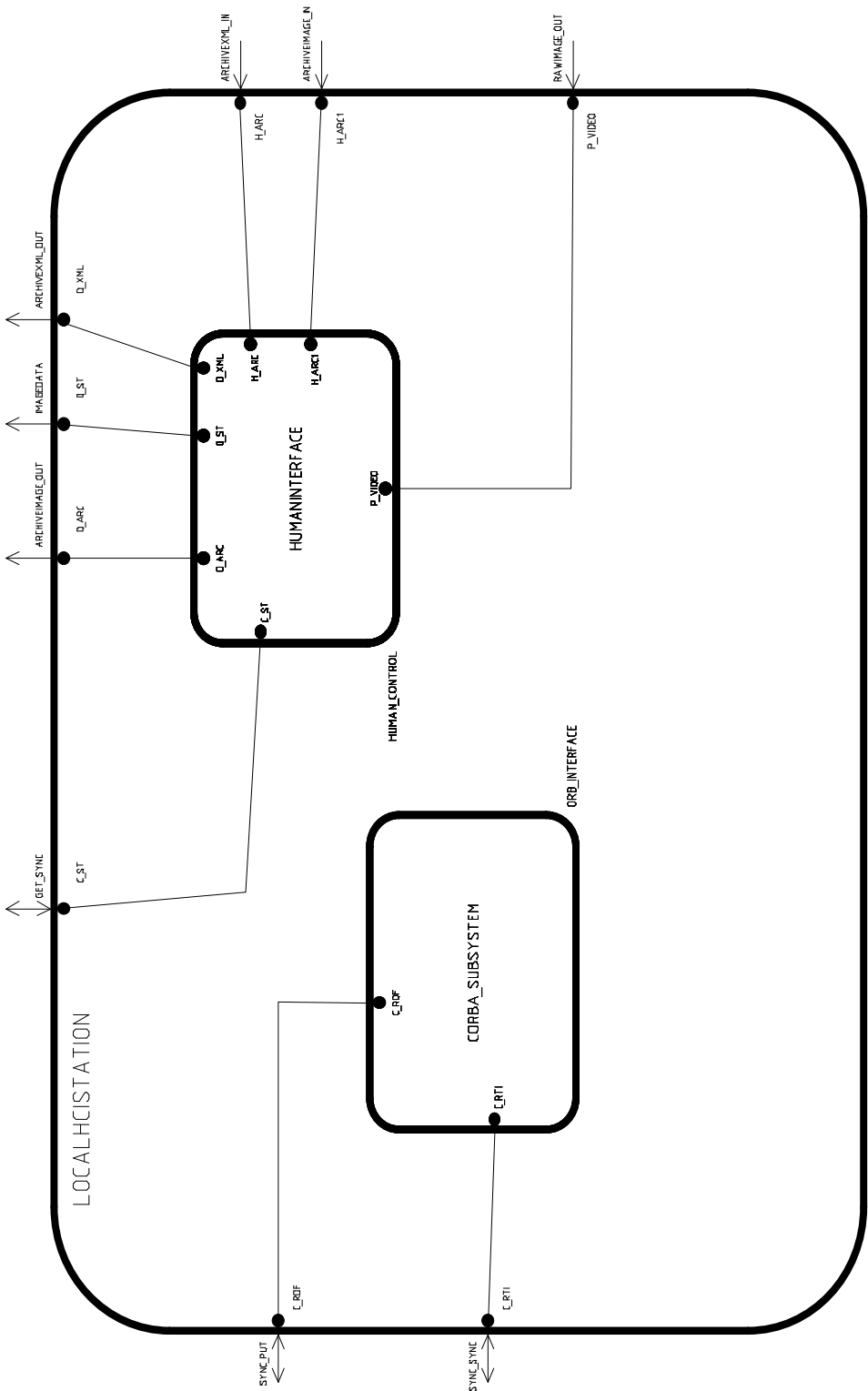


Figure C-11. LOCALHCISTATION.

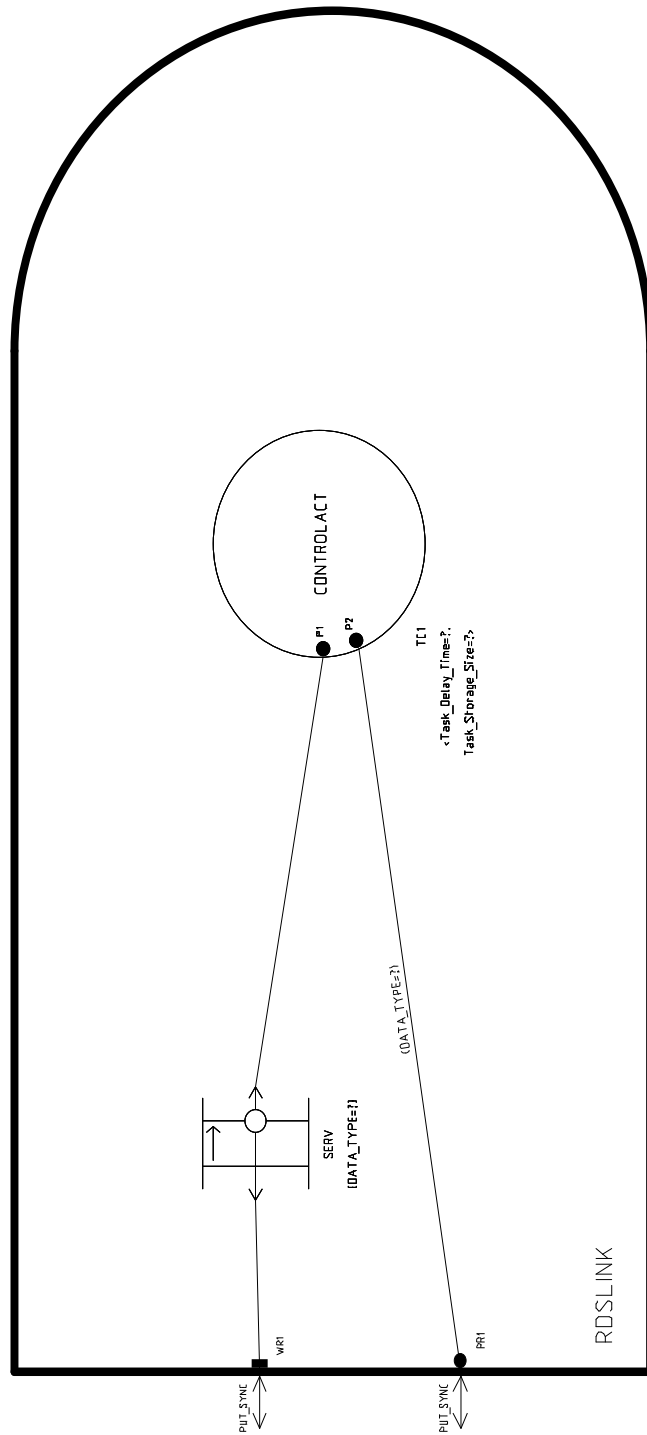


Figure C-12. RDSLINK.

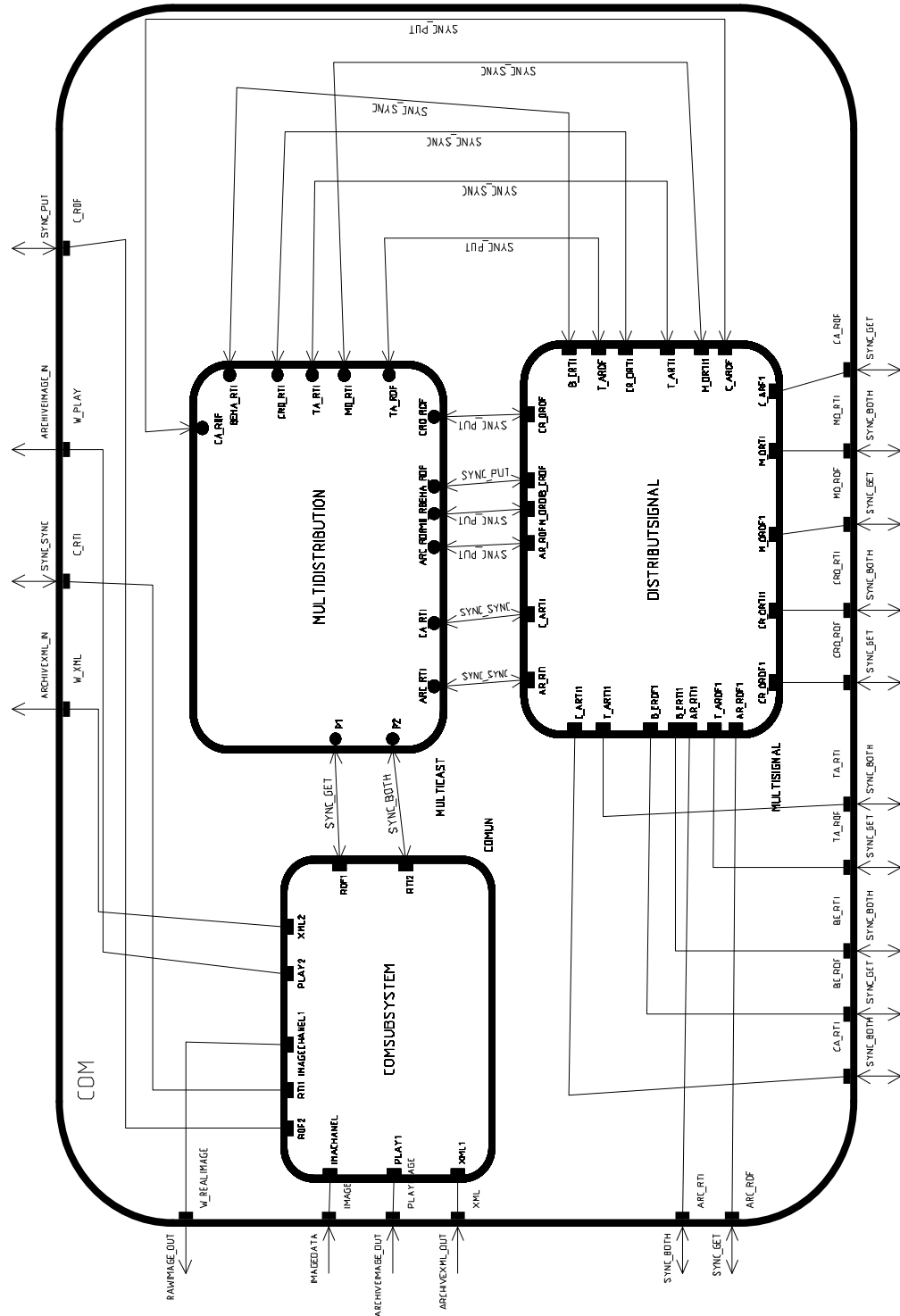


Figure C-13. COM.

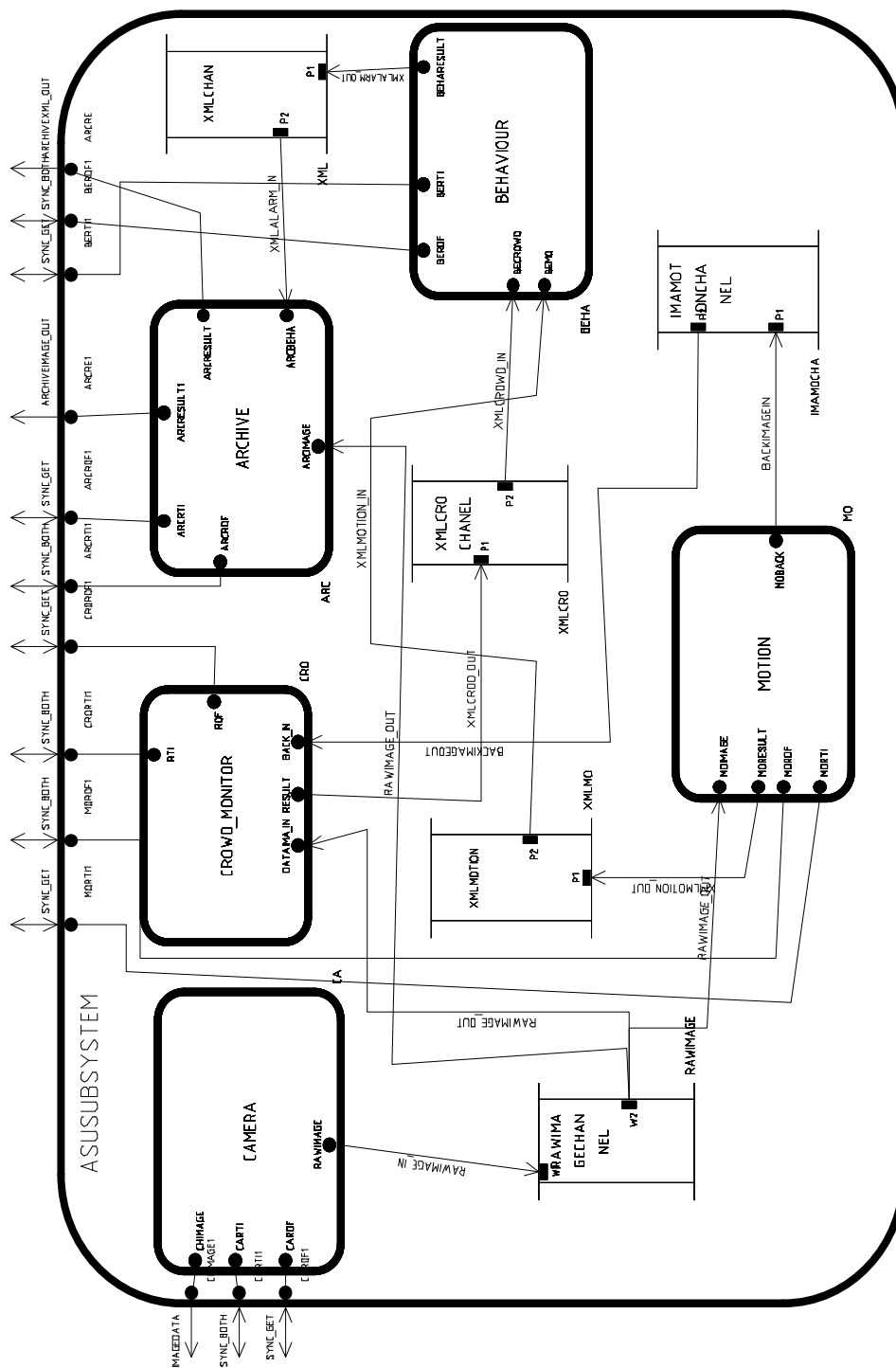


Figure C-14. ASUSUBSYSTEM.

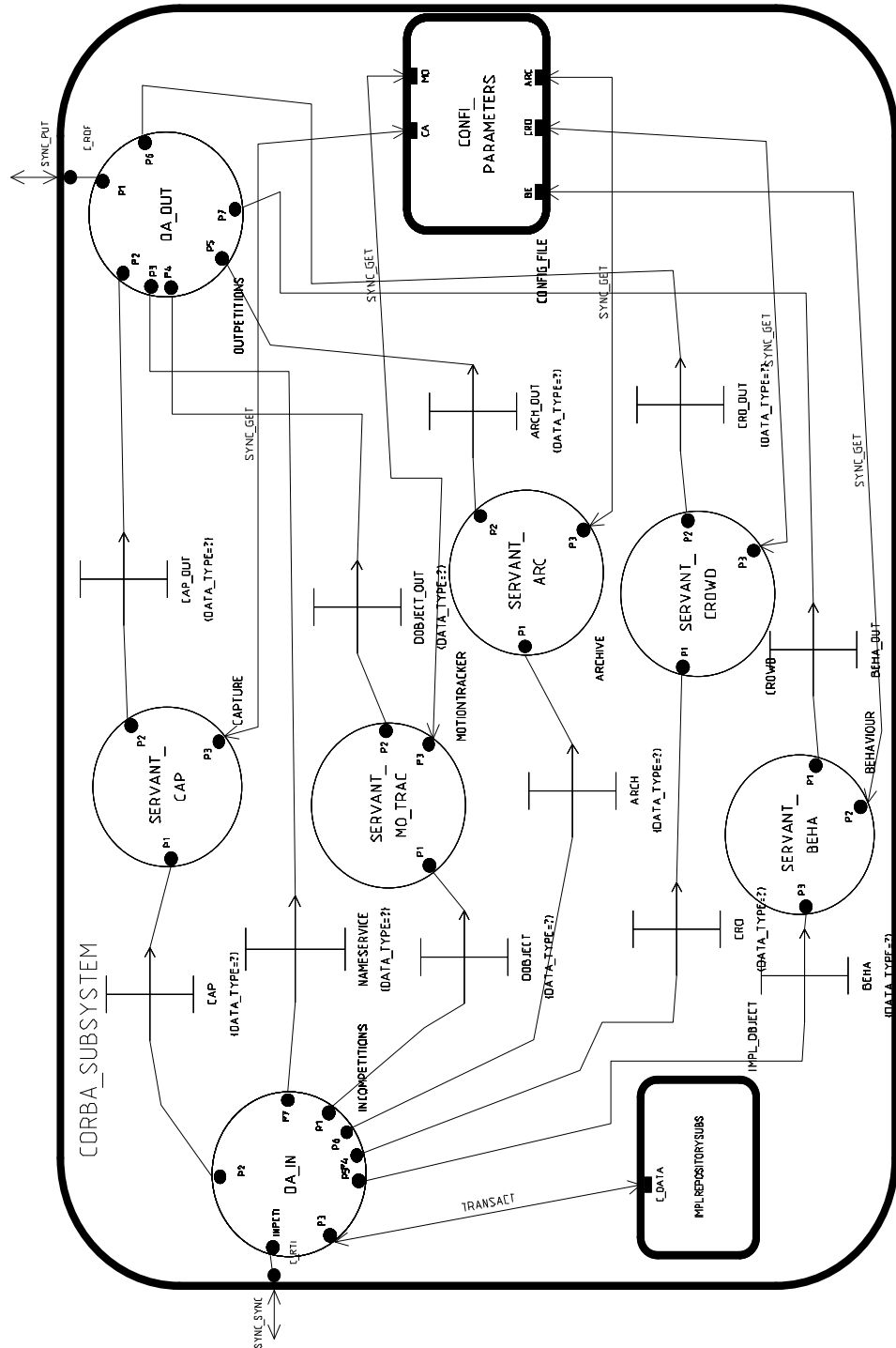


Figure C-15. CORBA_SUBSYSTEM.

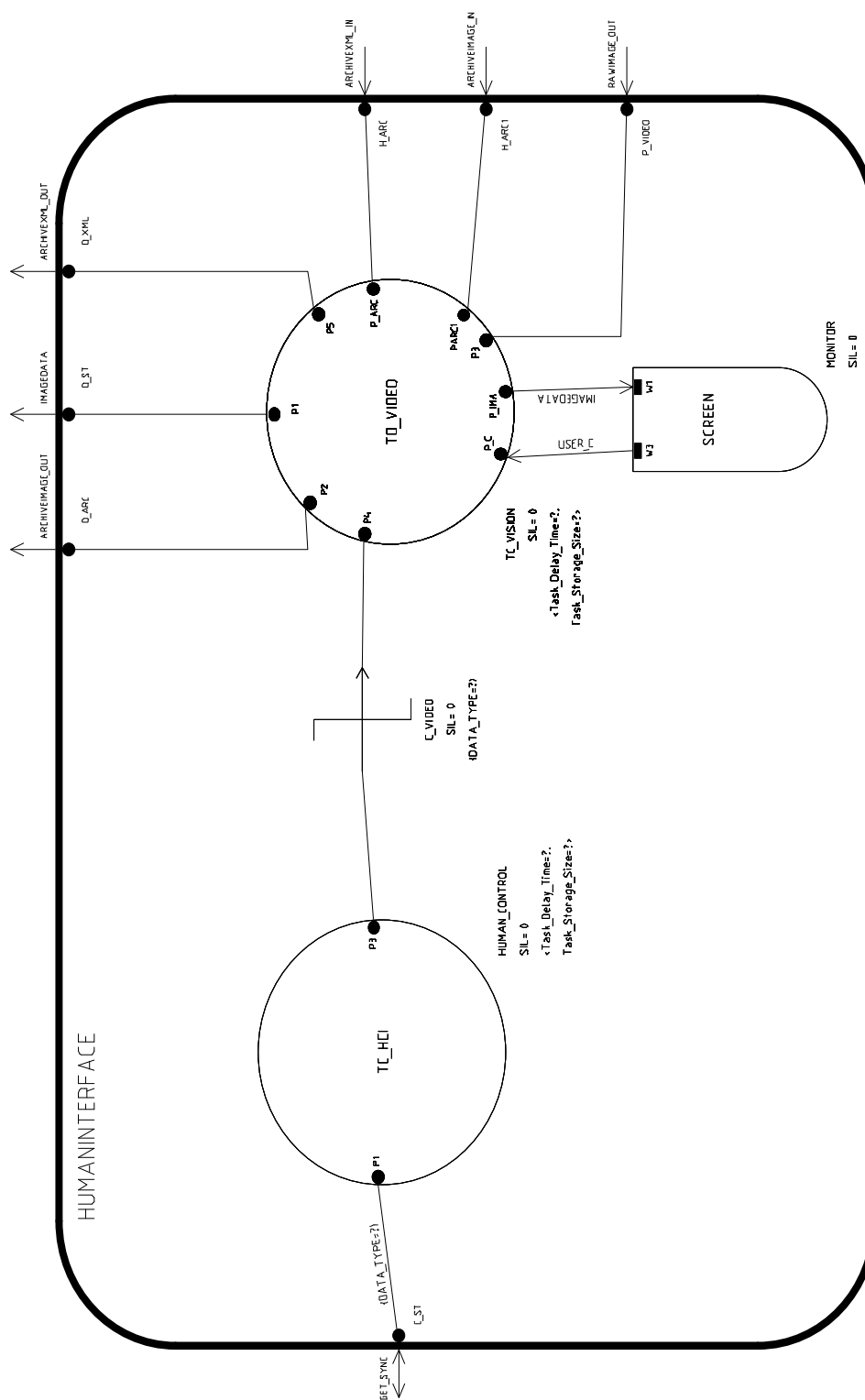


Figure C-16. HUMANINTERFACE.

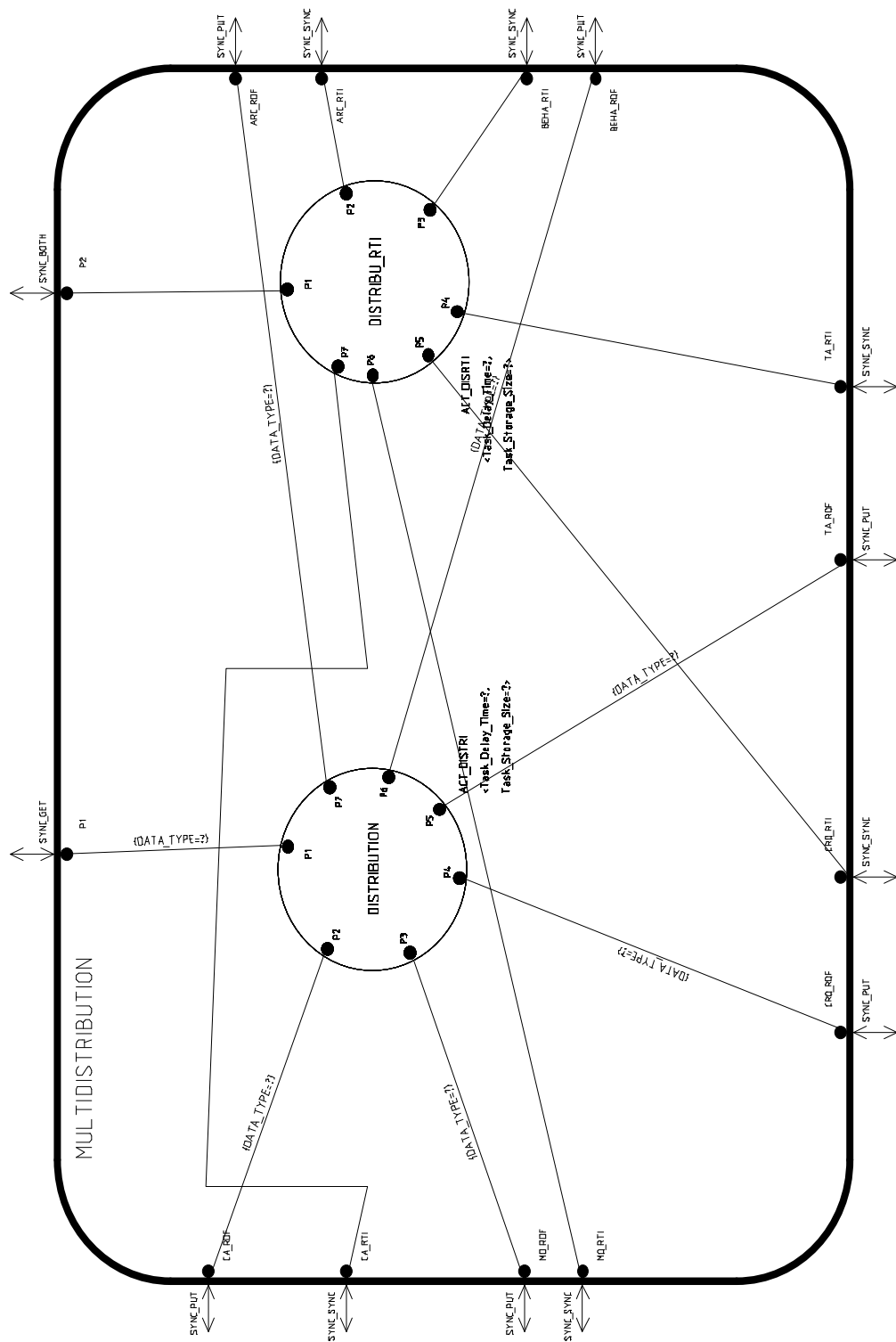


Figure C-17. MULTIDISTRIBUTION.

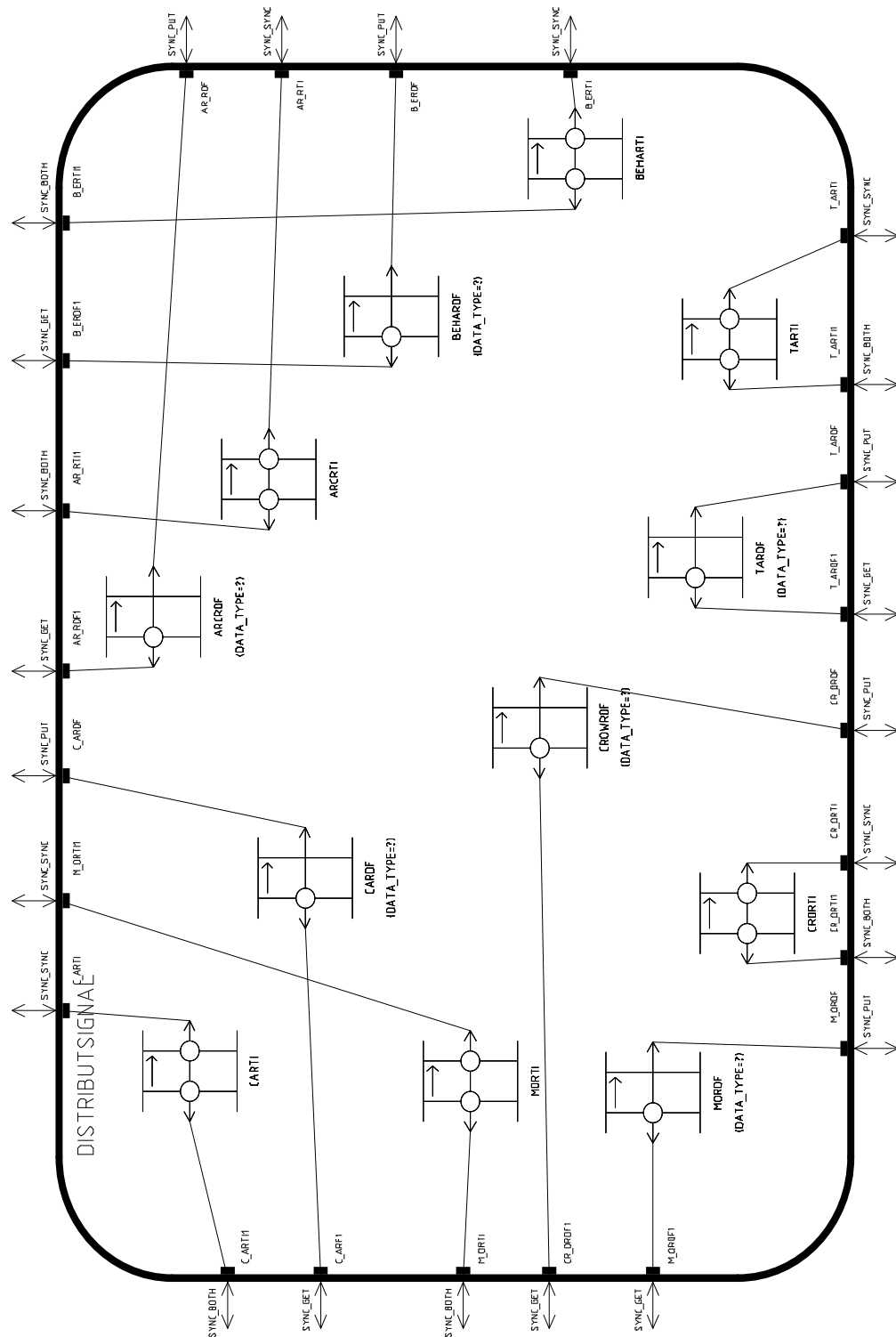


Figure C-18. DISTRIBUTSIGNAL.

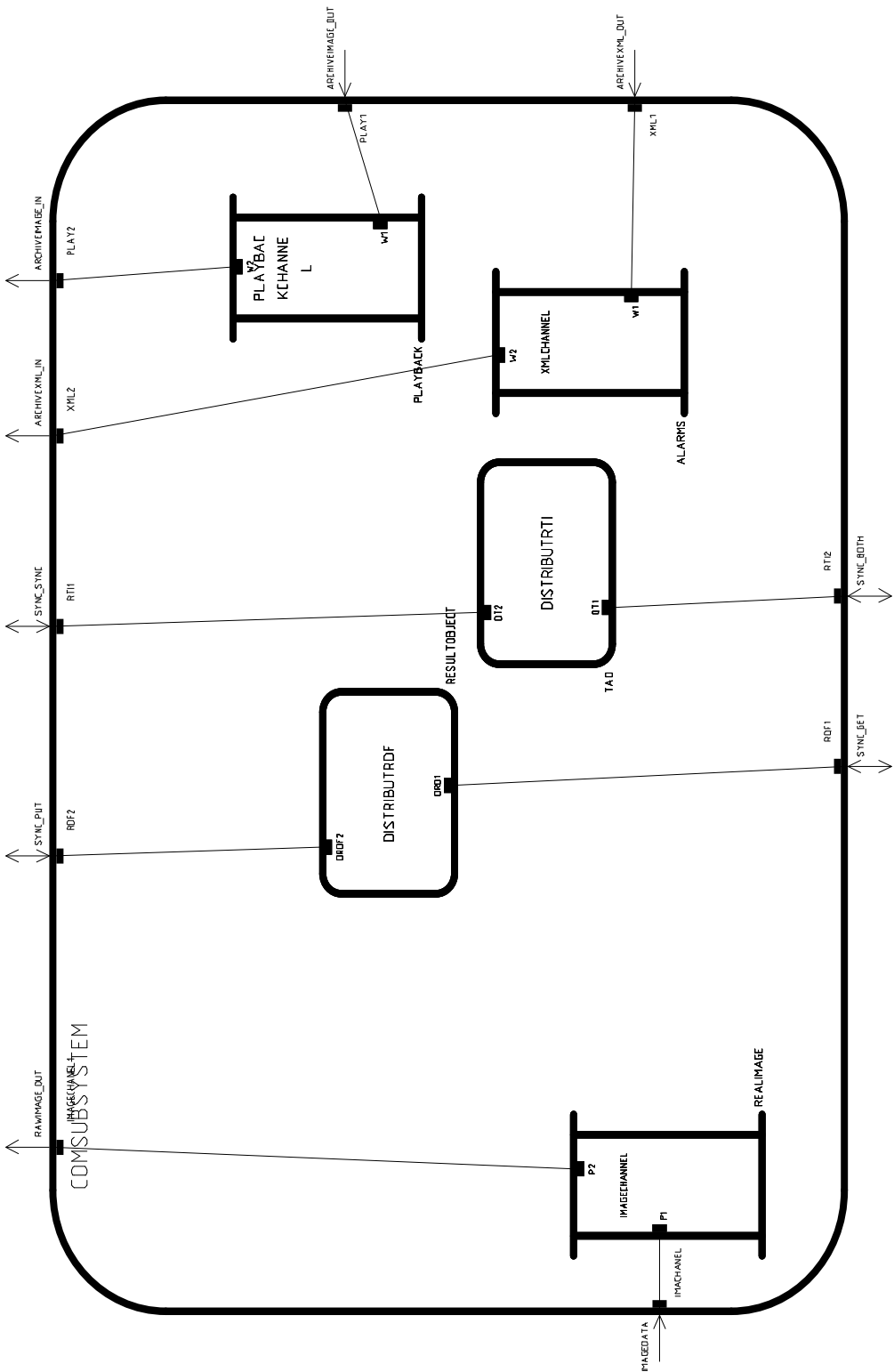


Figure C-19. COMSUBSYSTEM.

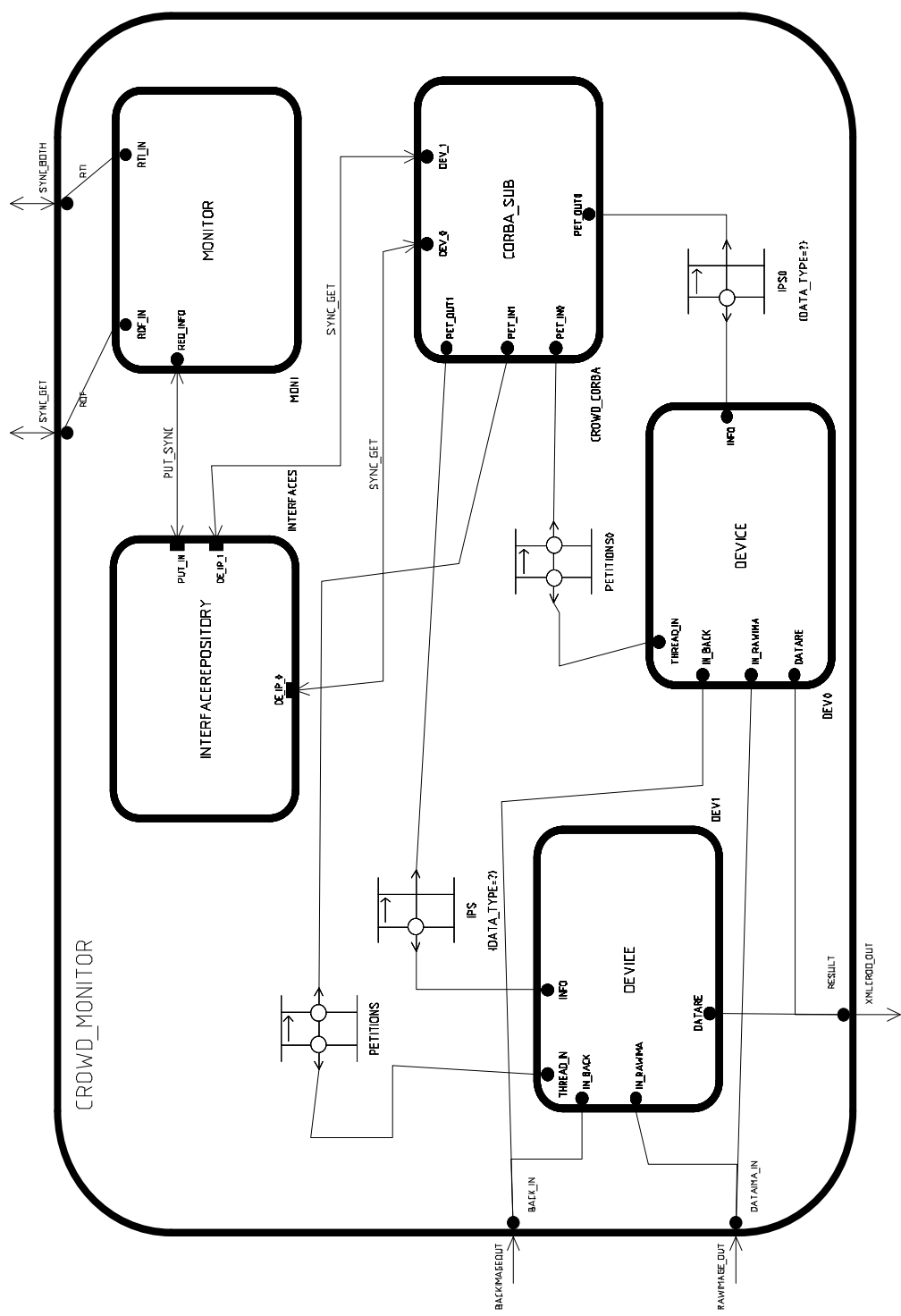


Figure C-20. CROWD_MONITOR.

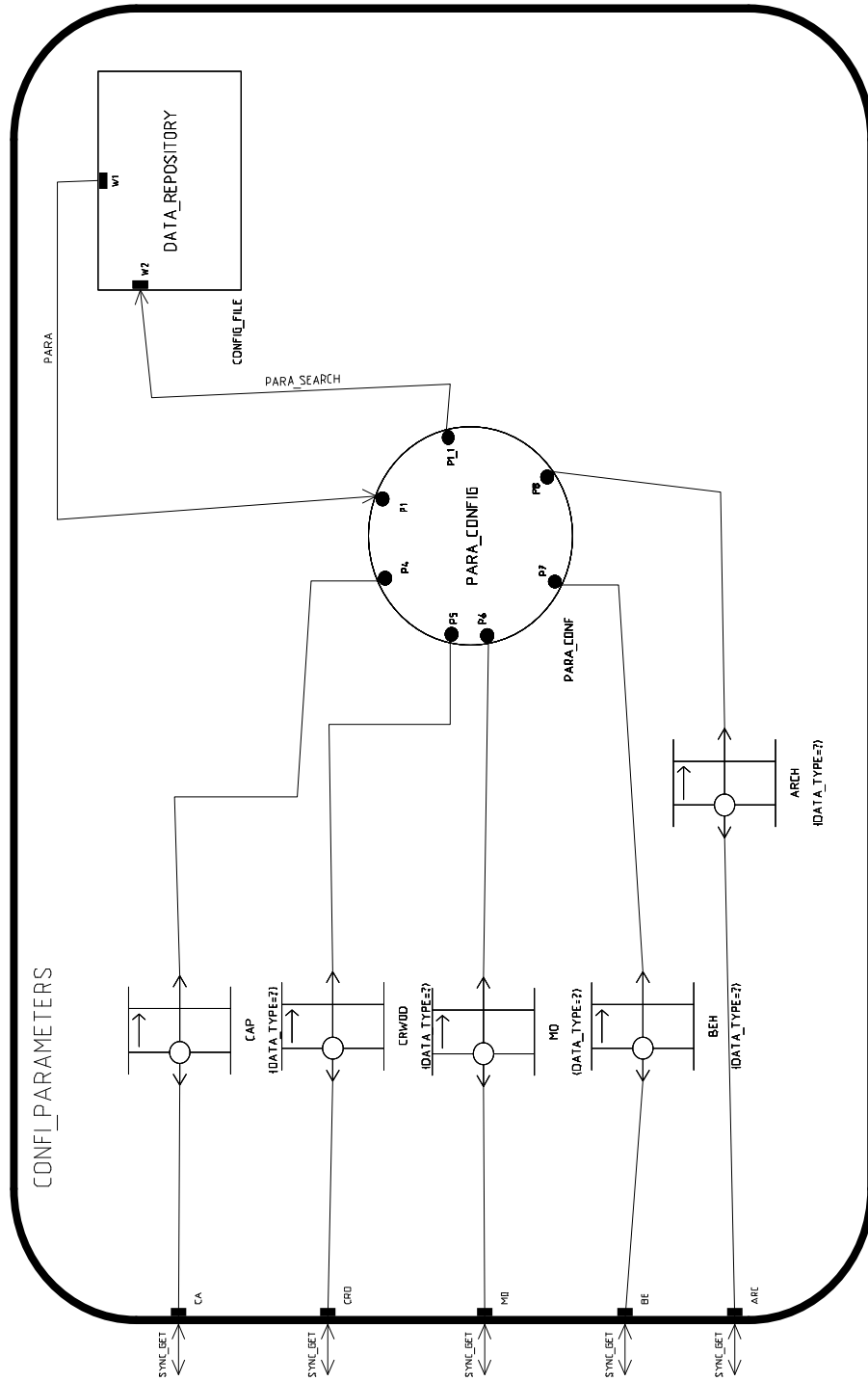


Figure C-21. CONFI_PARAMETERS.

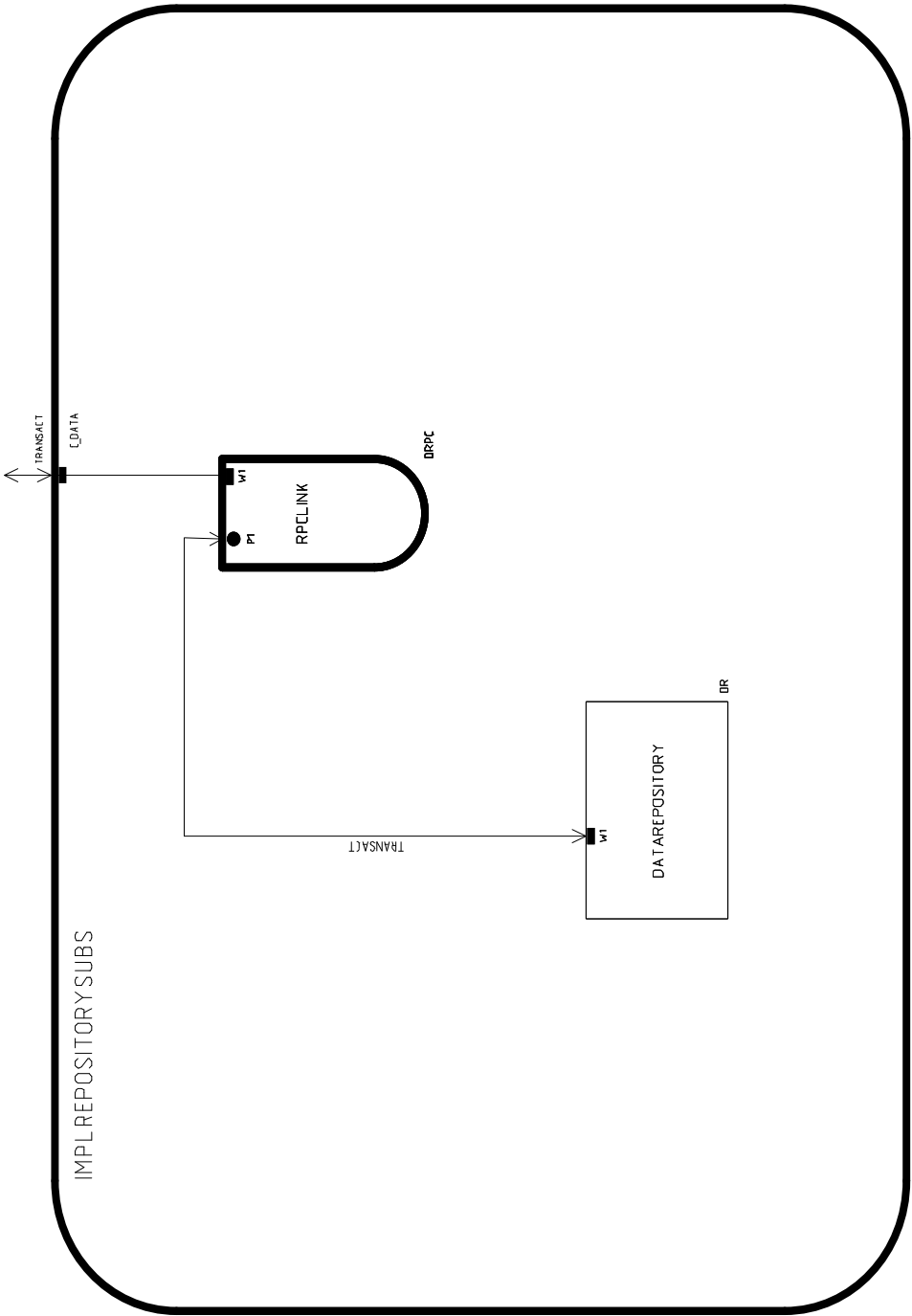


Figure C-22. IMPLREPOSITORYSUBS.

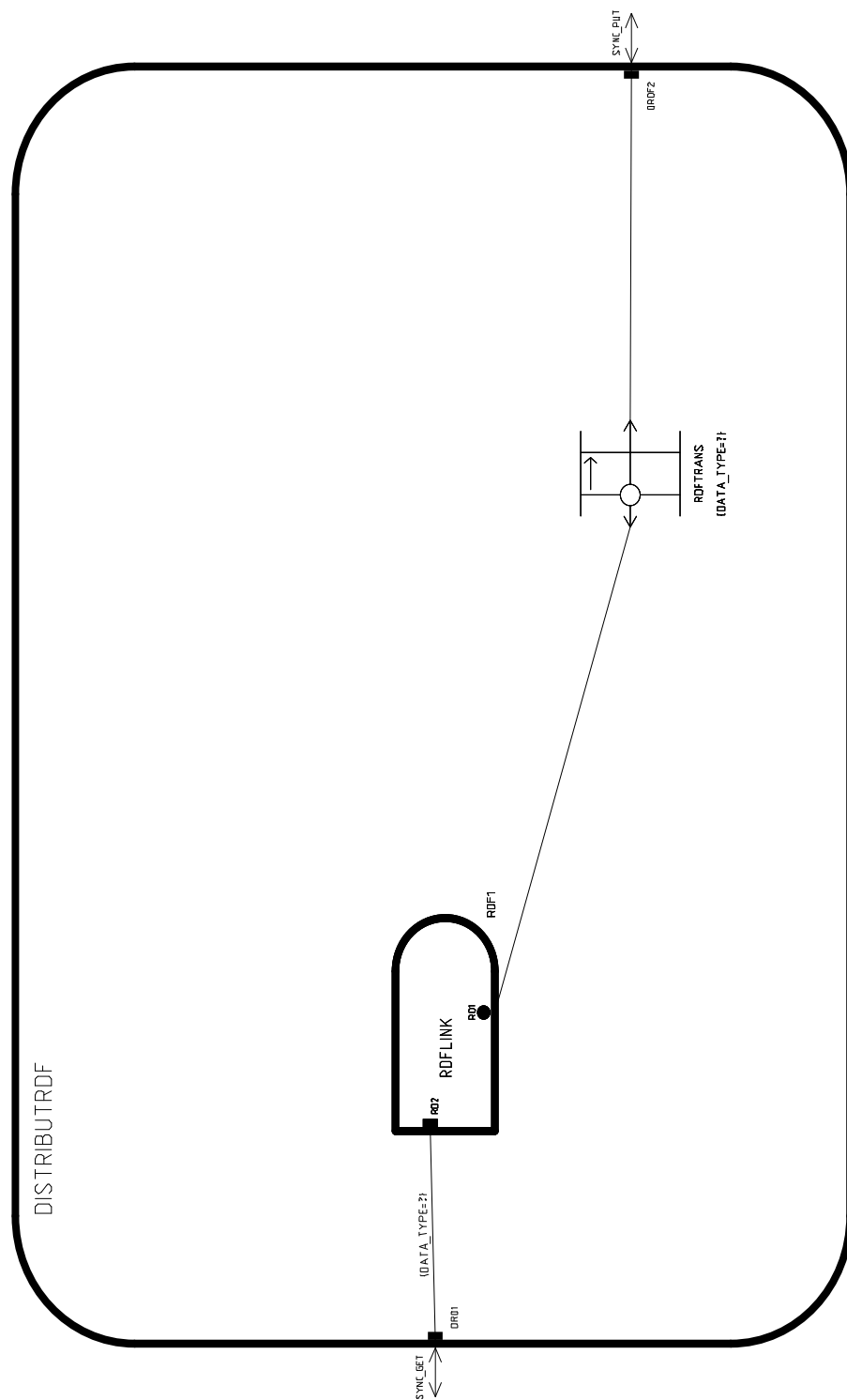


Figure C-23. DISTRIBUTRDF.

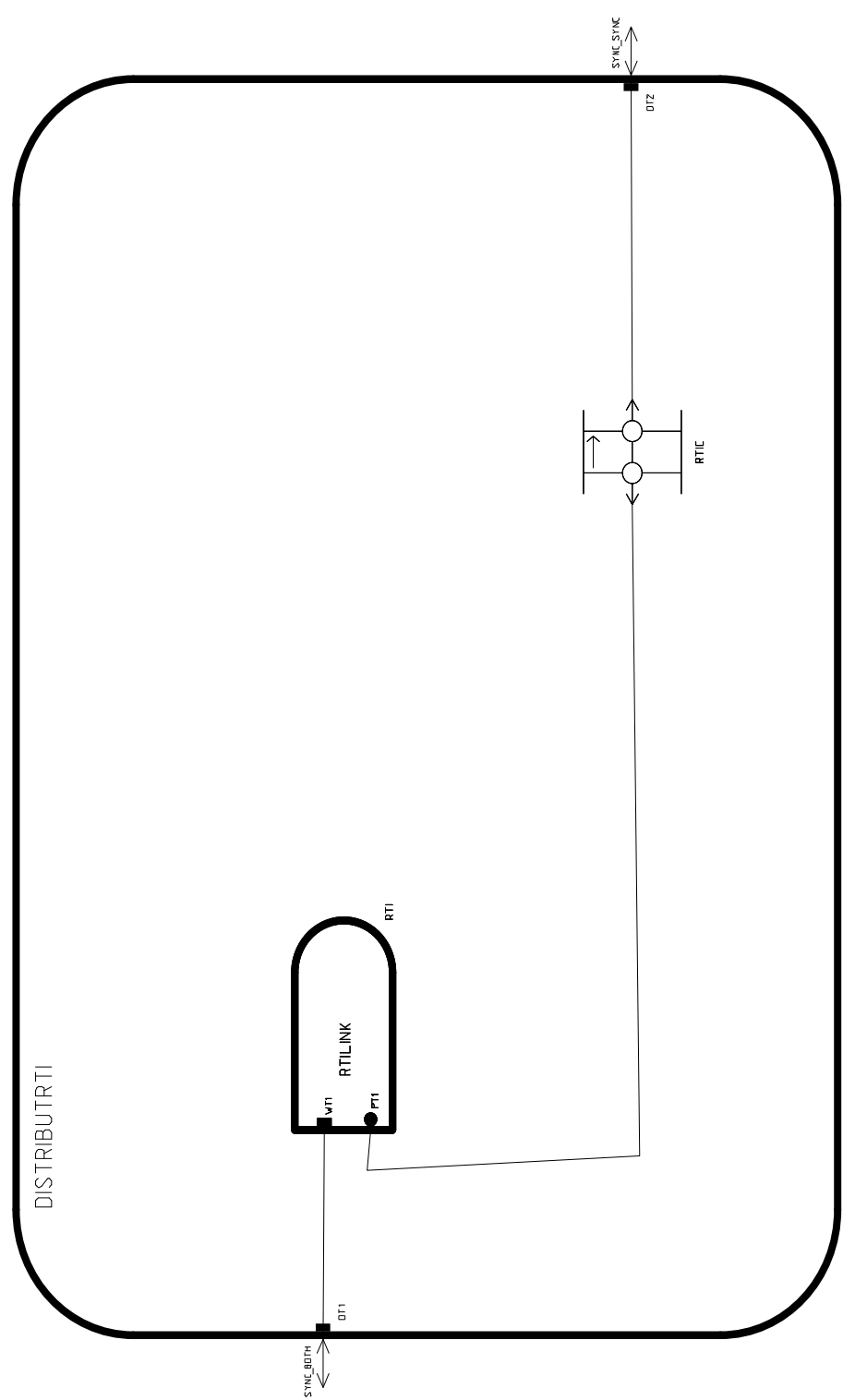


Figure C-24. DISTRIBUTRTI.

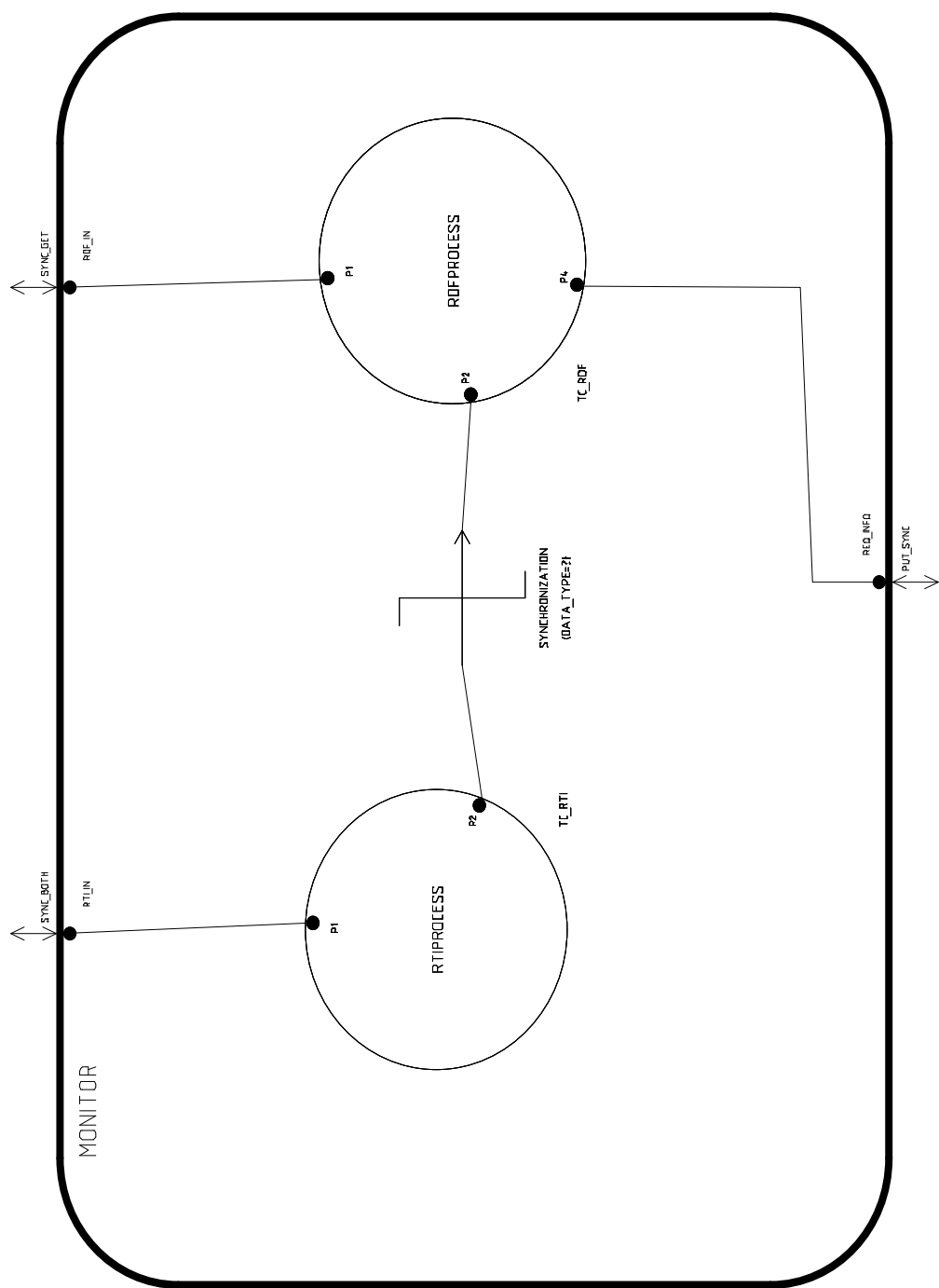


Figure C-25. MONITOR.

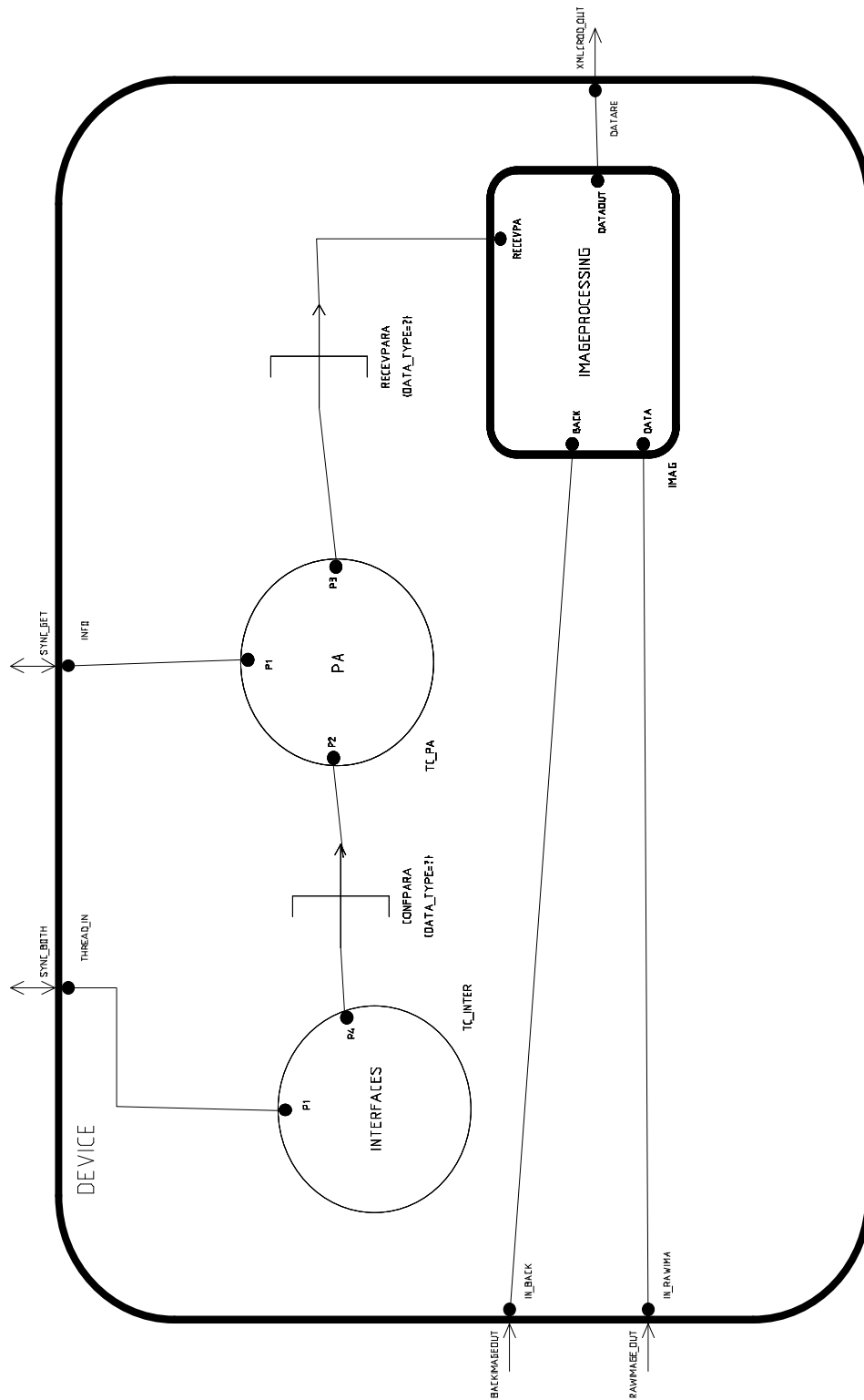


Figure C-26. DEVICE.

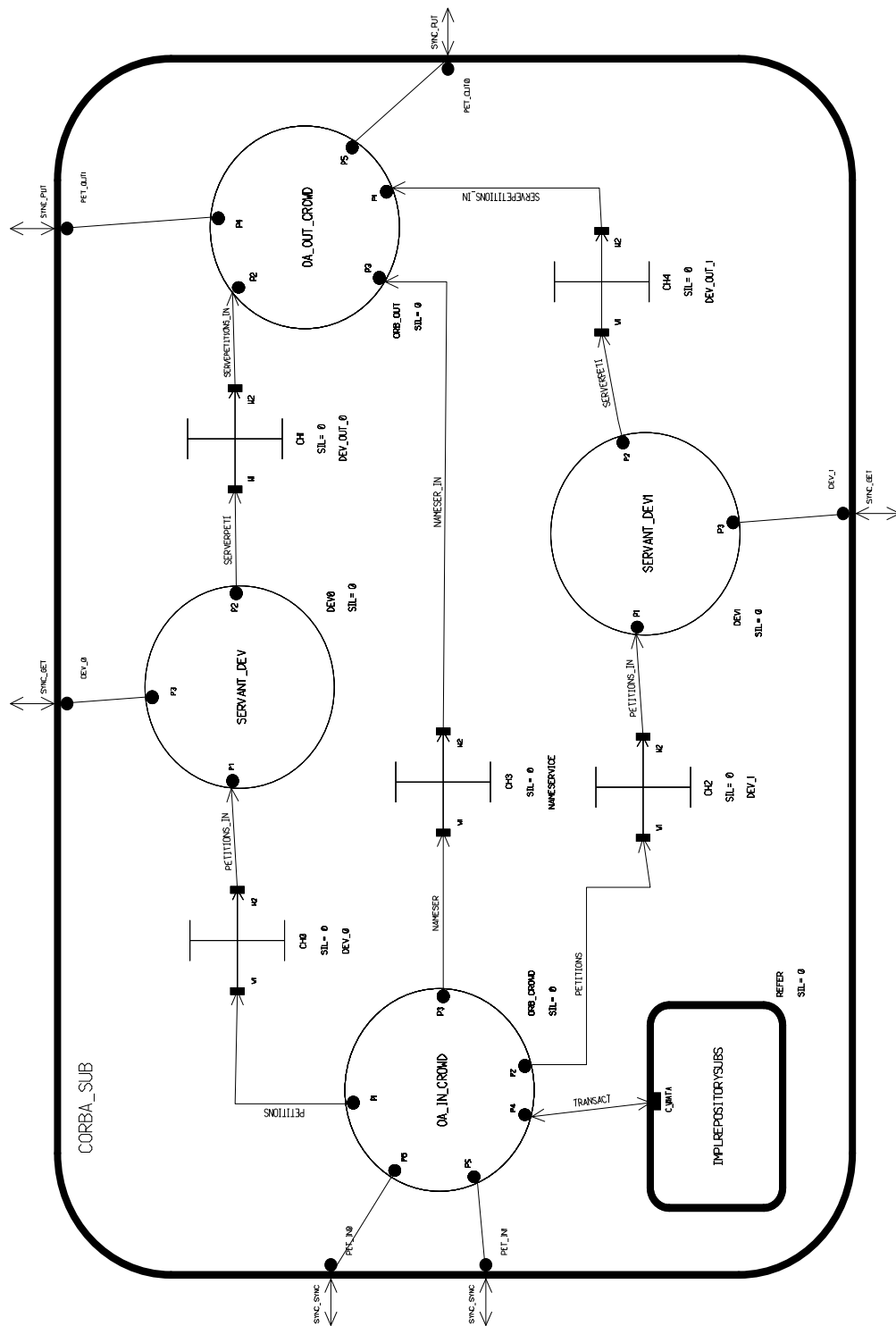


Figure C-27. CORBA_SUB.

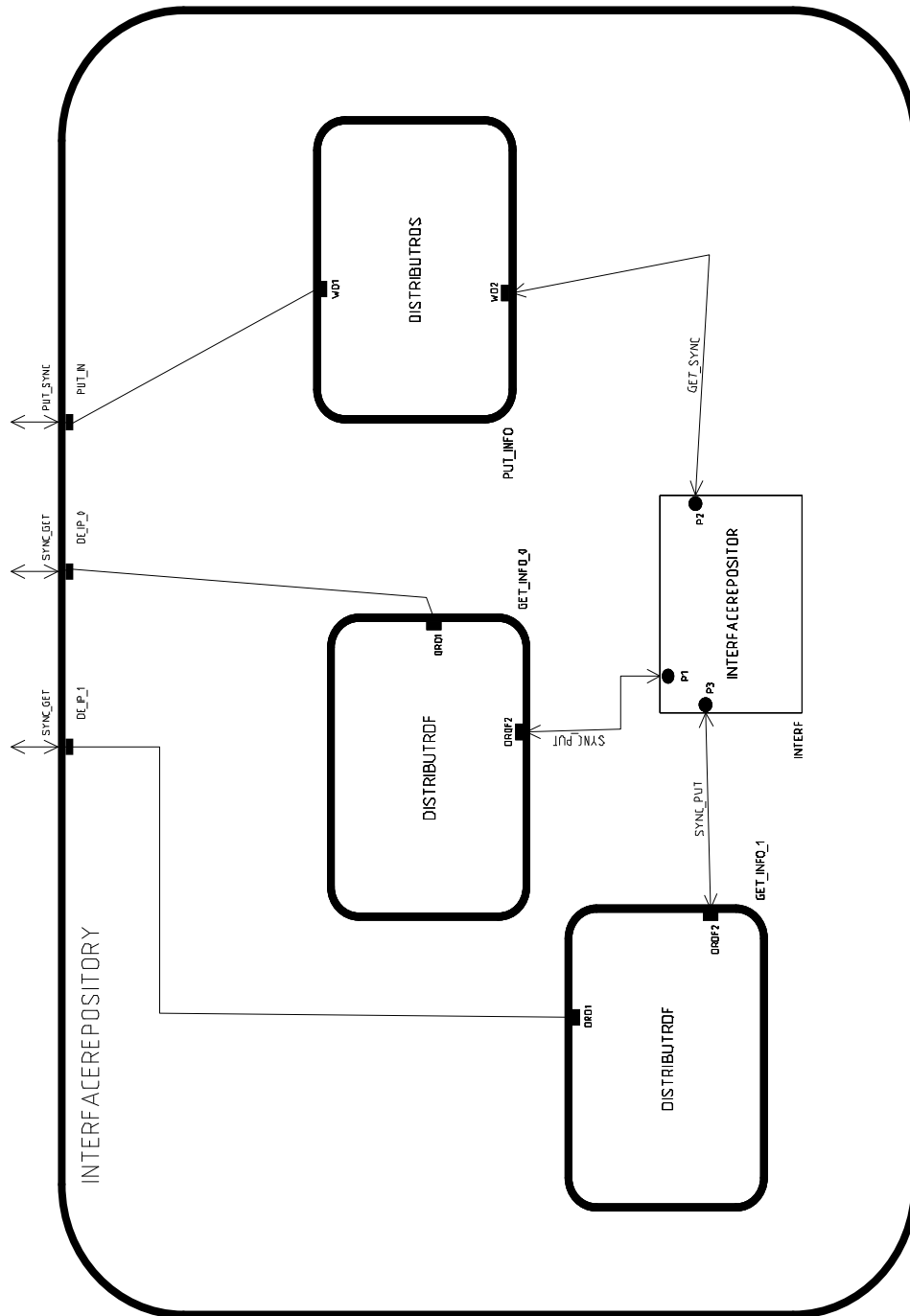


Figure C-28. INTERFACEREPOSITORY.

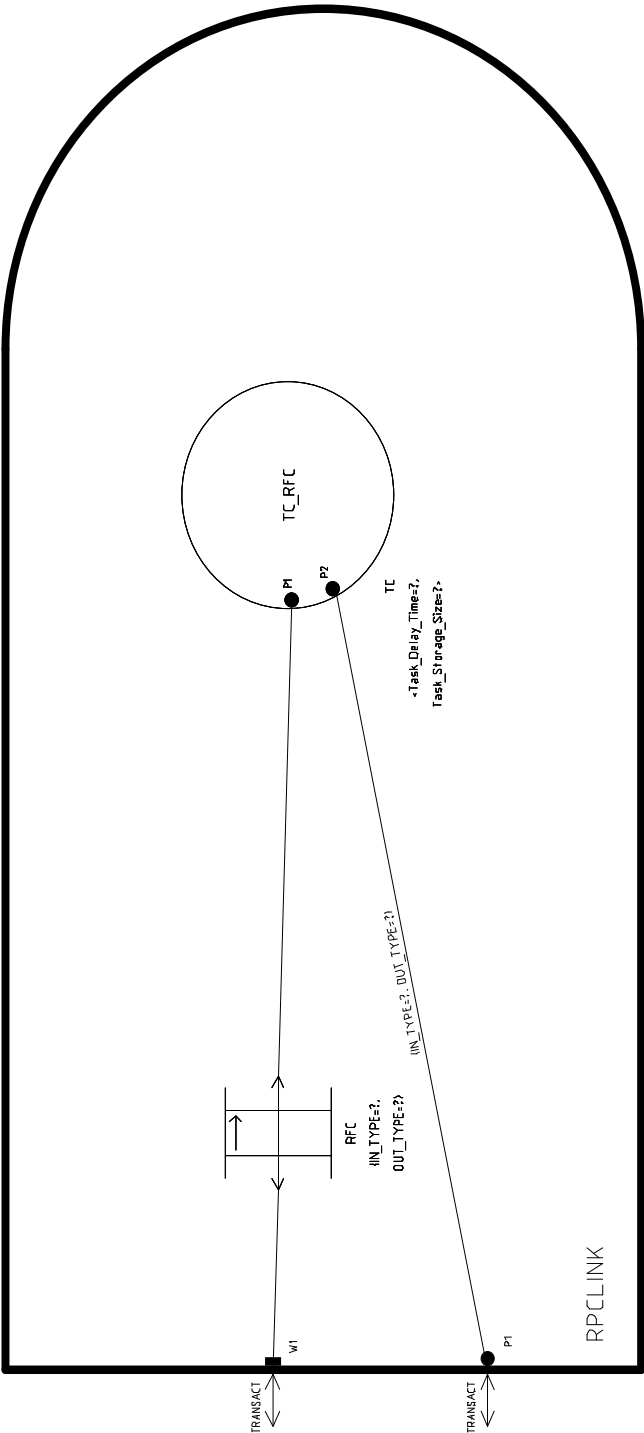


Figure C-29. RPCLINK.

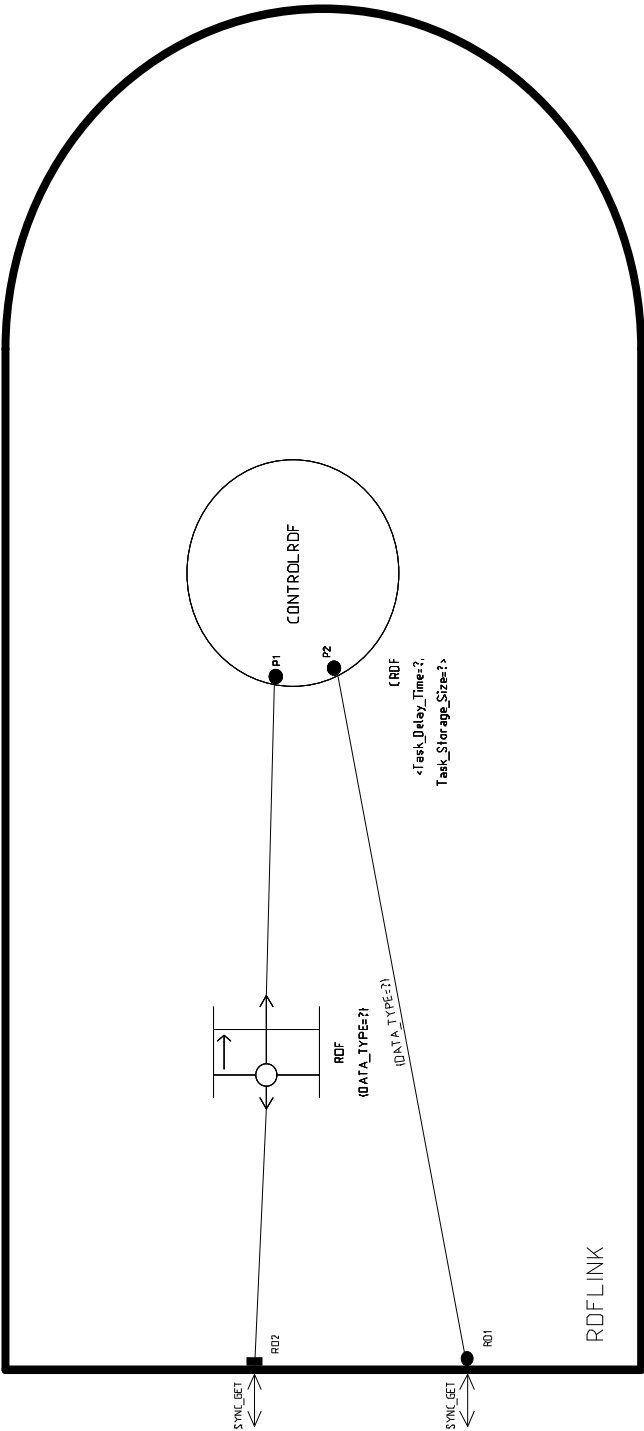


Figure C-30. RDFLINK.

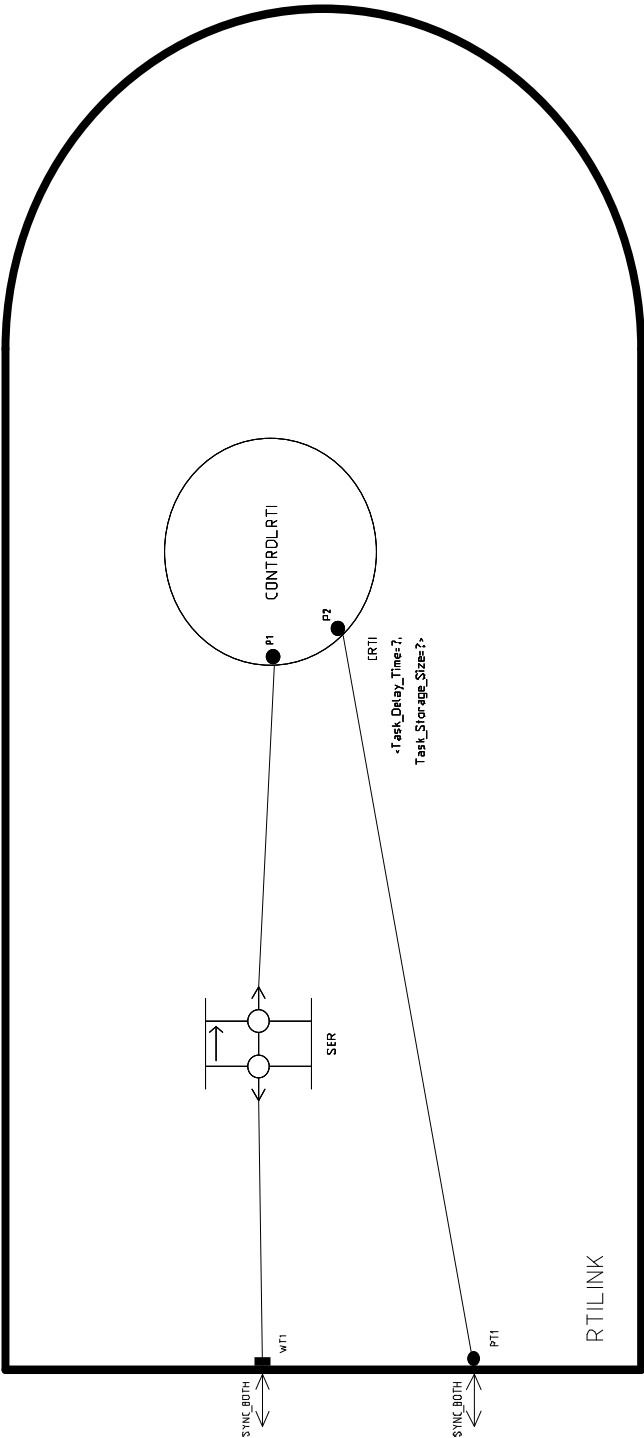


Figure C-31. RTILINK.

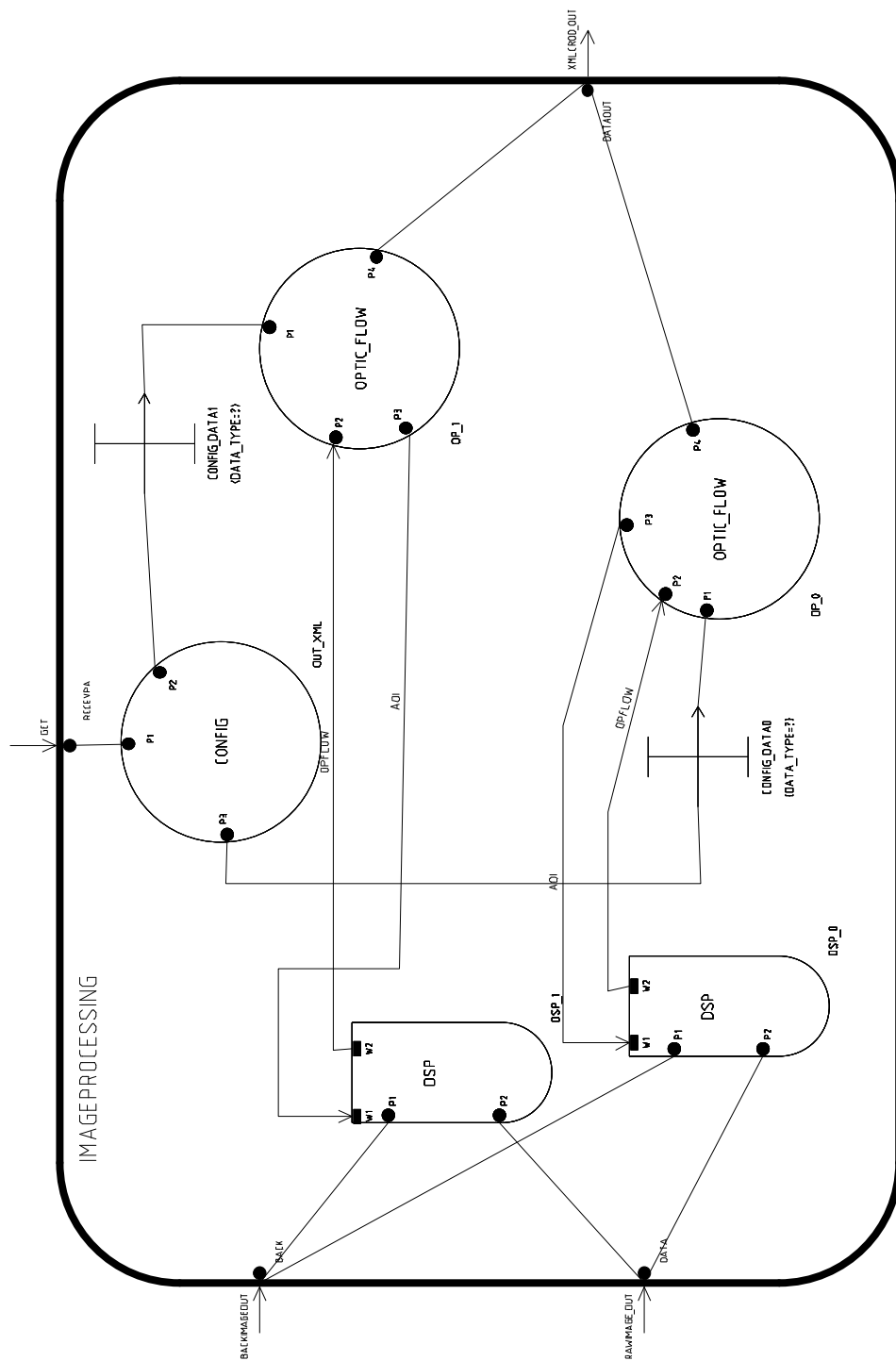


Figure C-32. IMAGEPROCESSING.

The ADVISOR Prototype architecture using the DORIS method

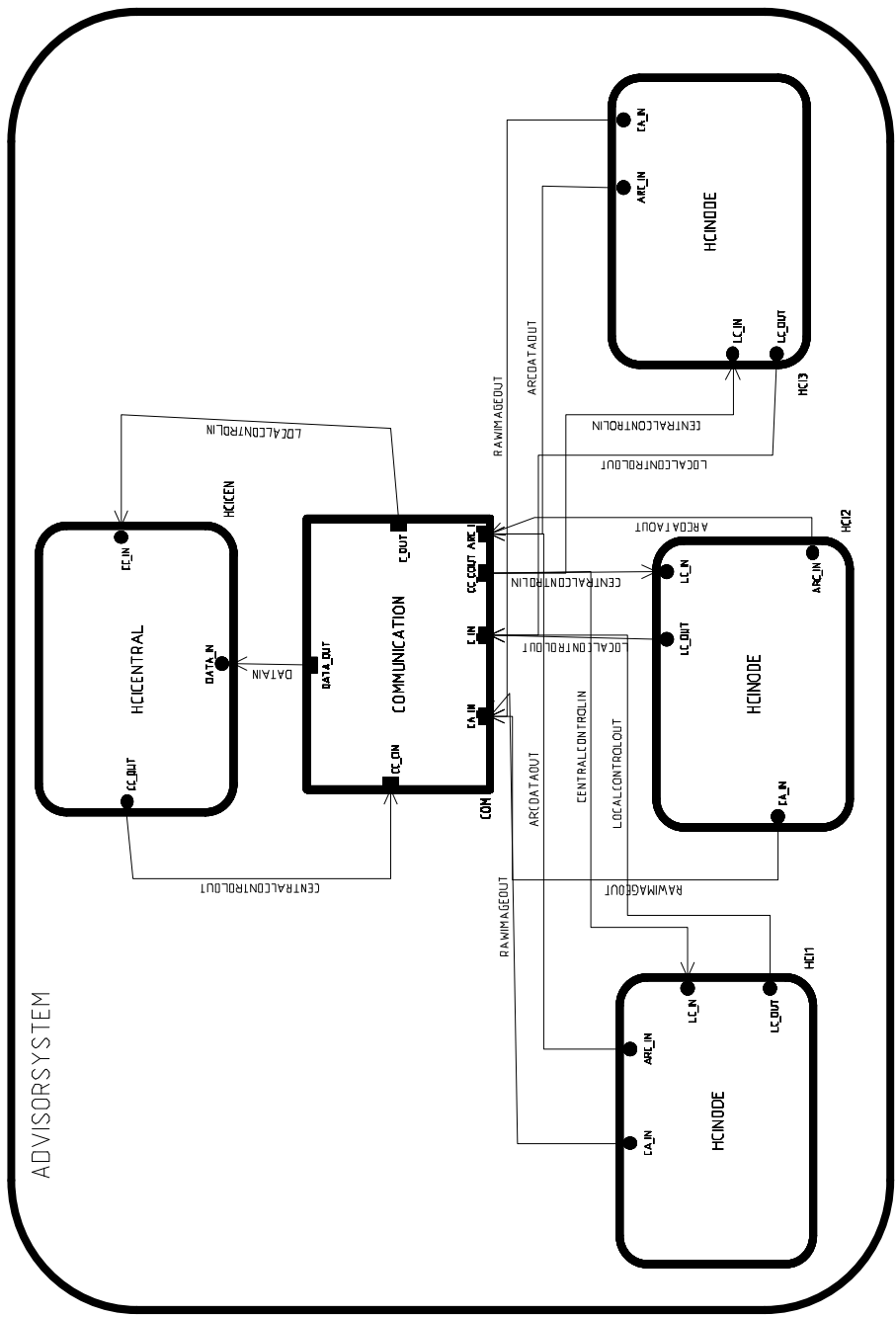


Figure C-33. ADVISORSYSTEM.

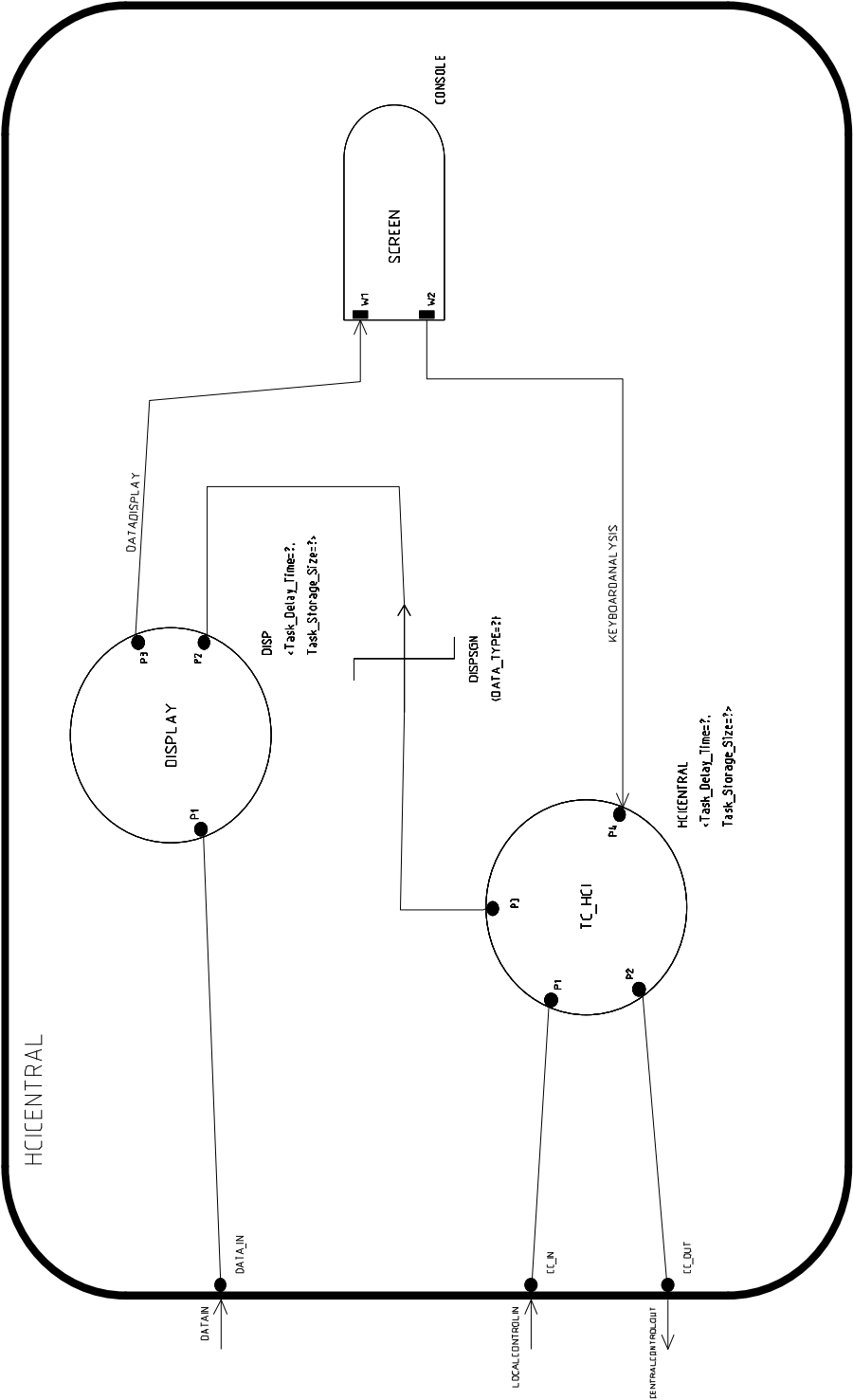


Figure C-34. HCICENTRAL.

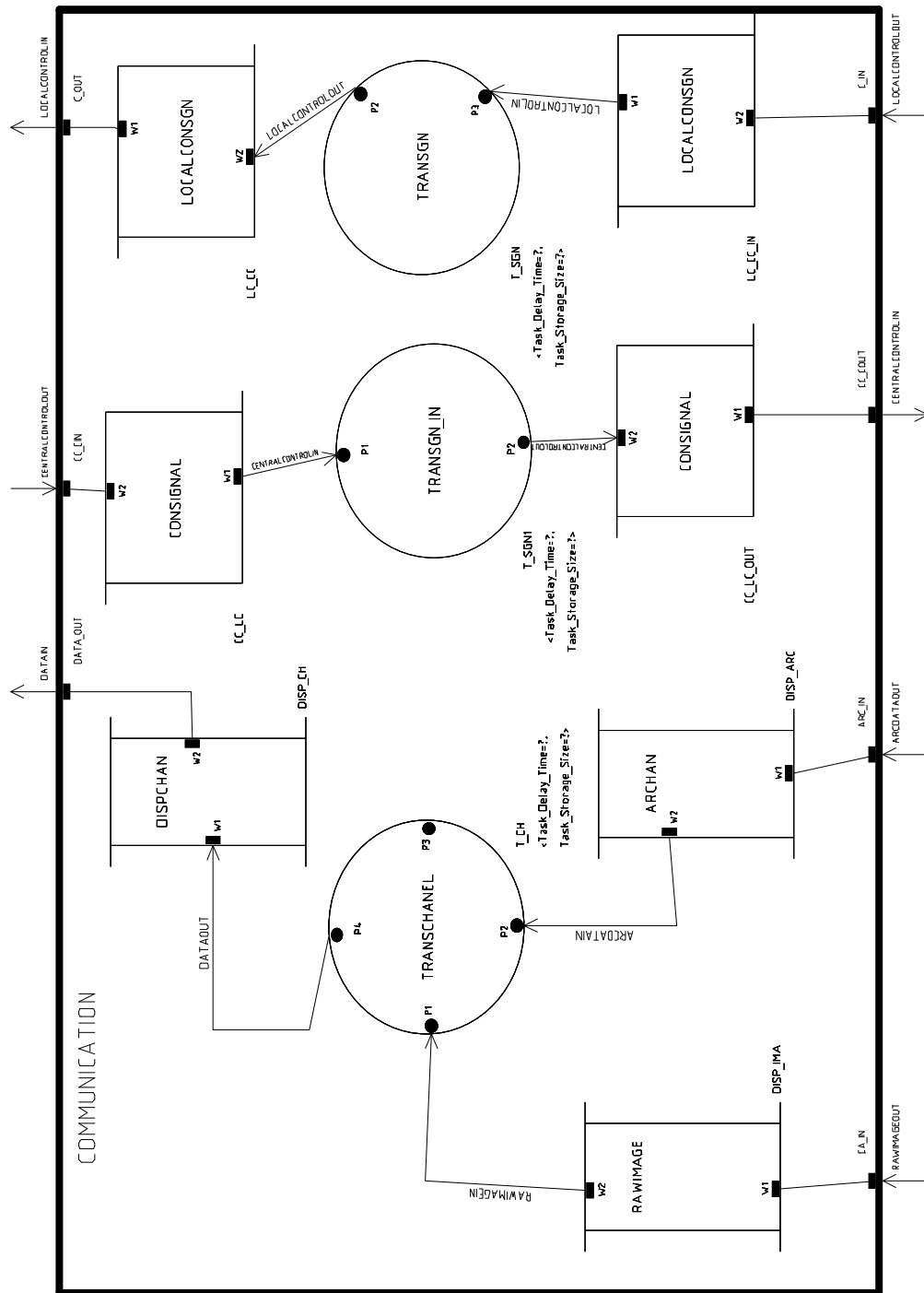


Figure C-35. COMMUNICATION.

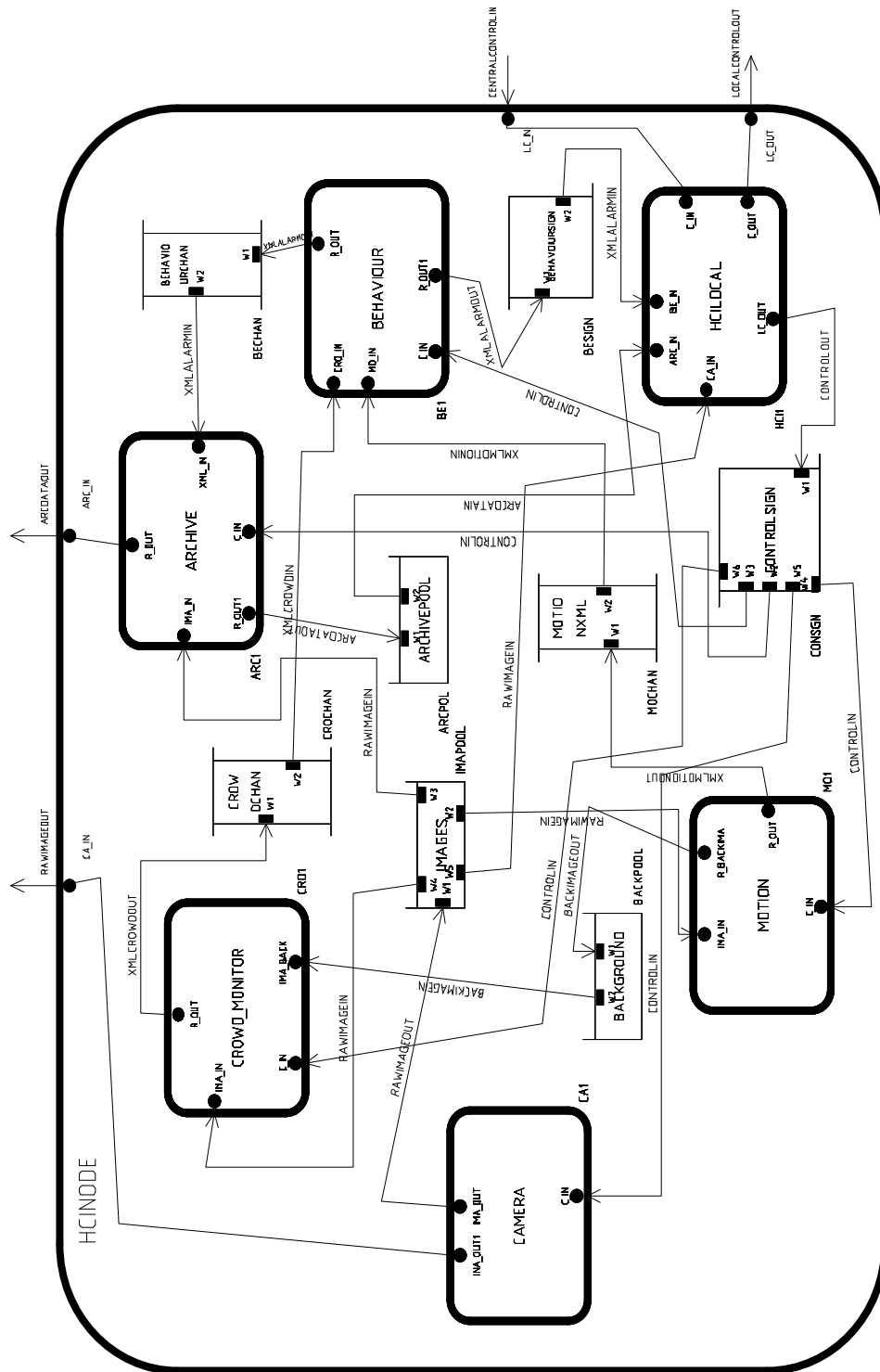


Figure C-36. HCINODE.

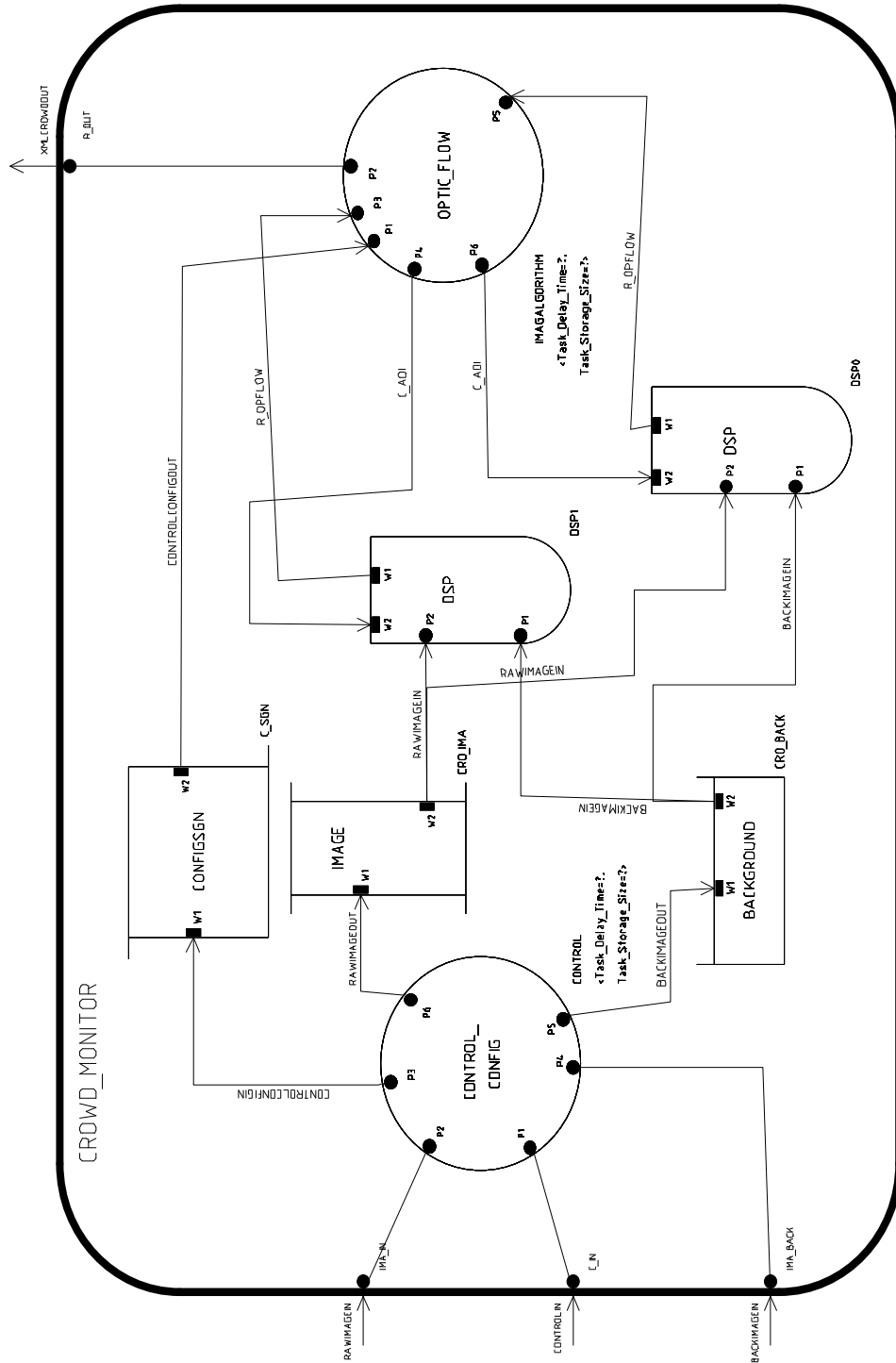


Figure C-37. CROWD_MONITOR.

Appendix D

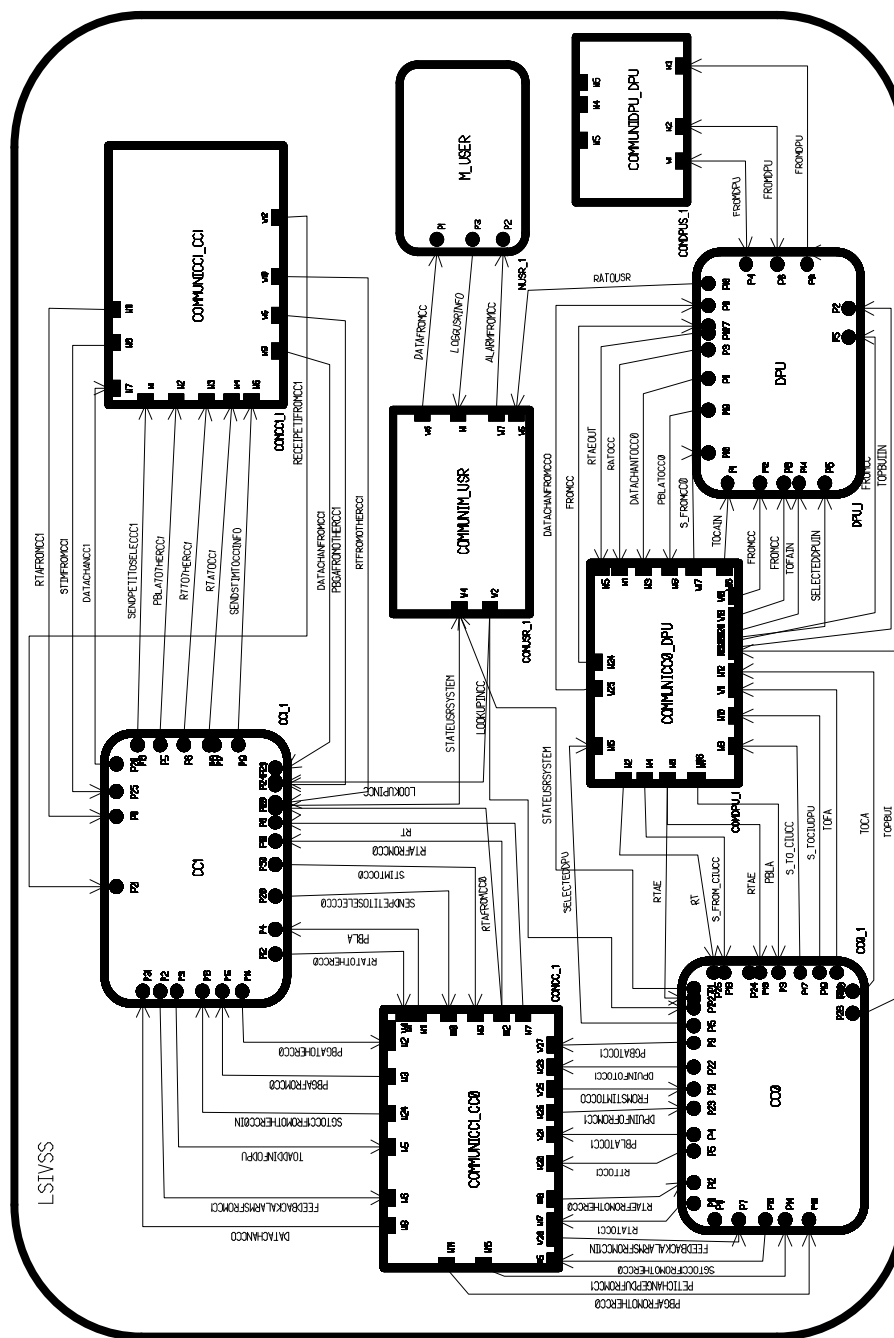


Figure D-1. LSIVSS.

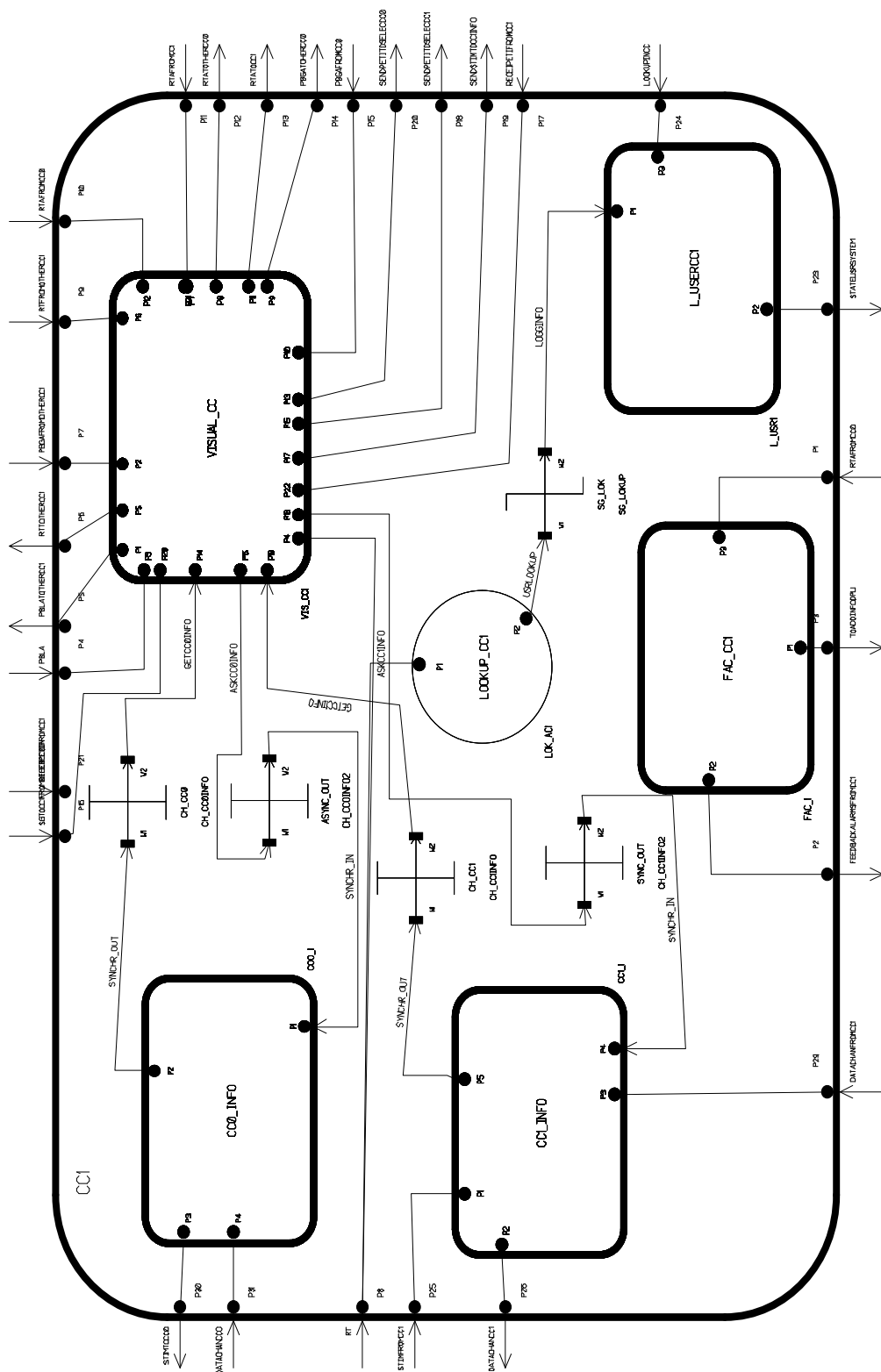


Figure D-2. CC1.

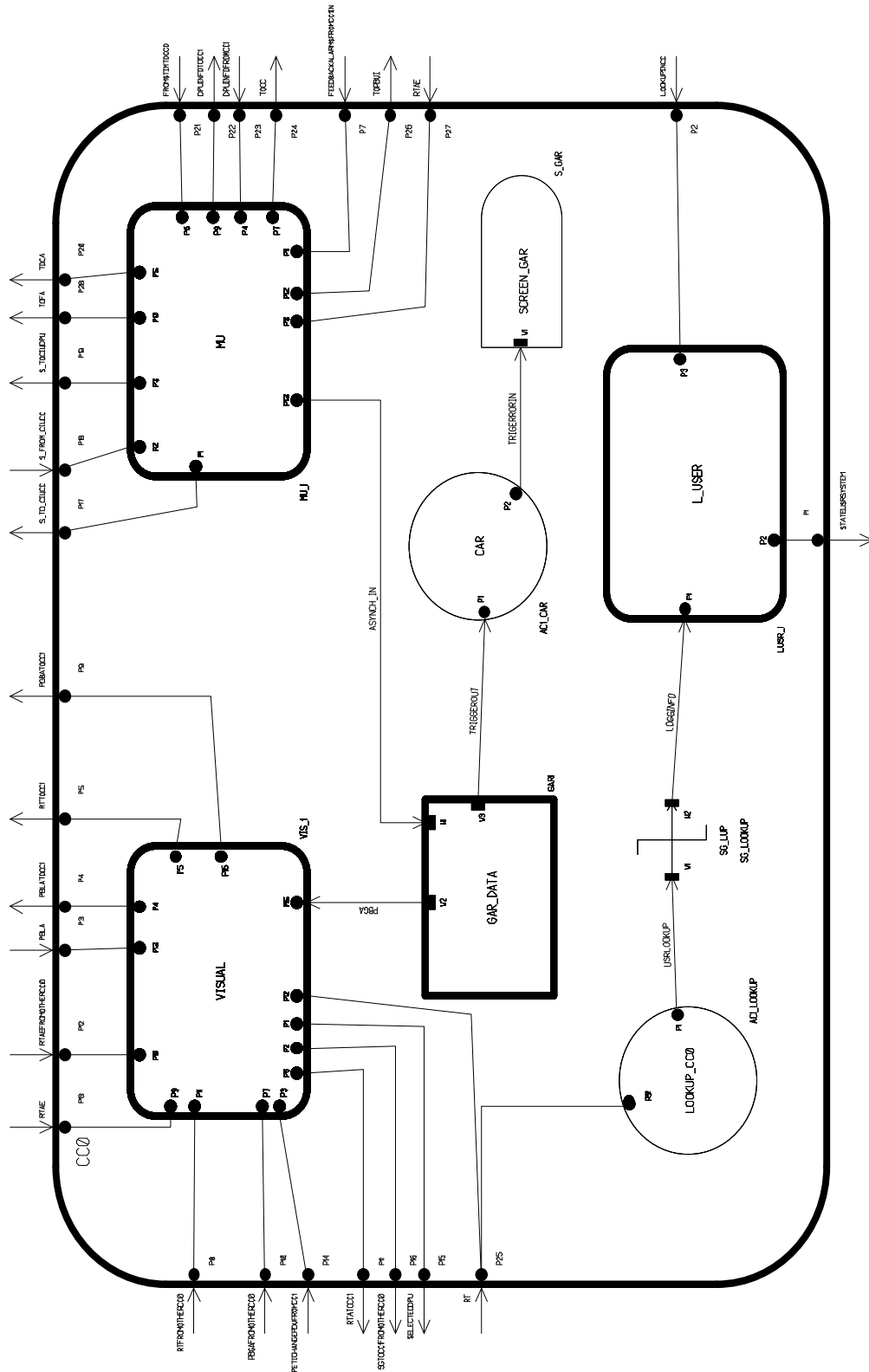


Figure D-3. CC0.

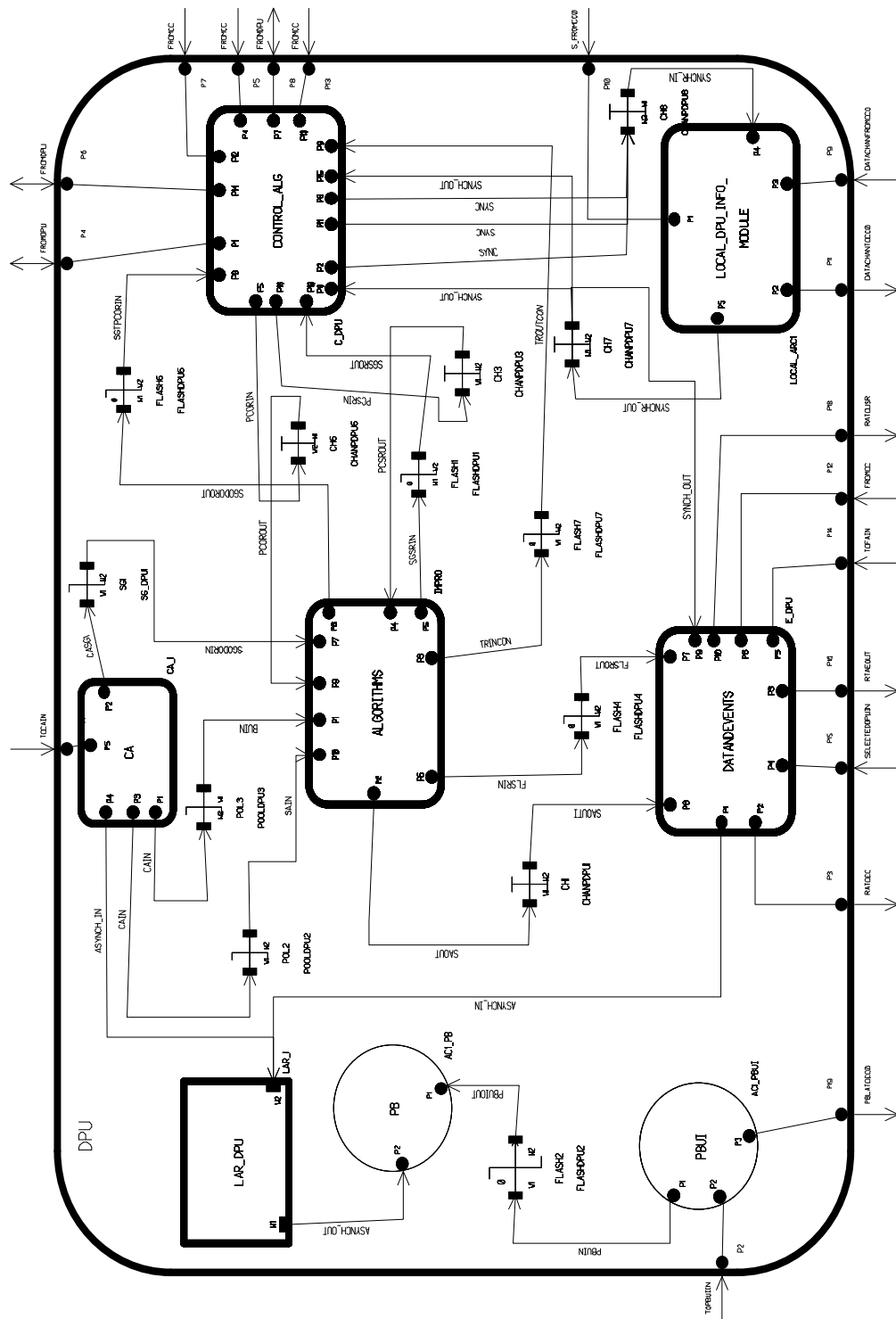


Figure D-4. DPU.

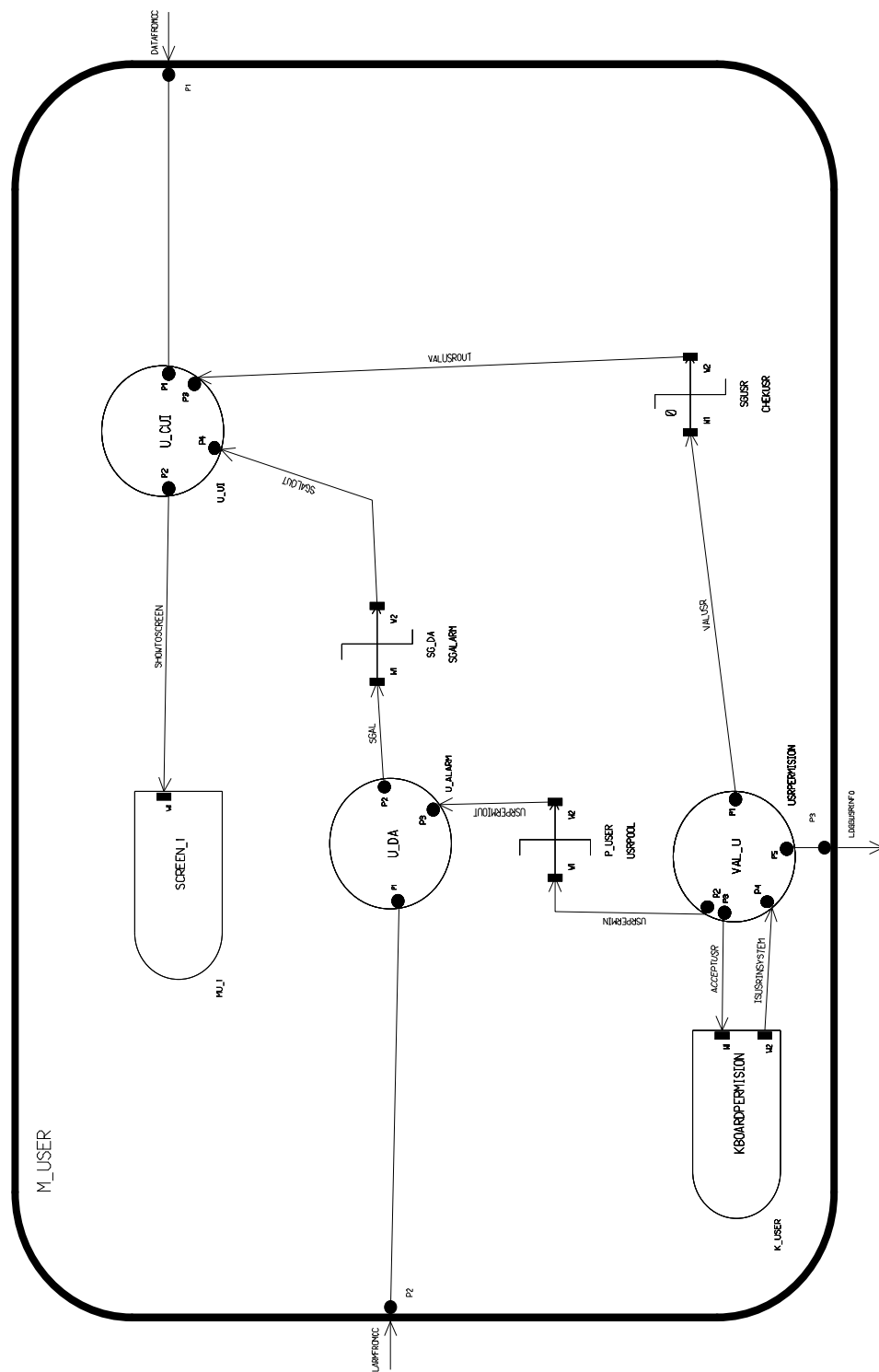


Figure D-5. M_USER.

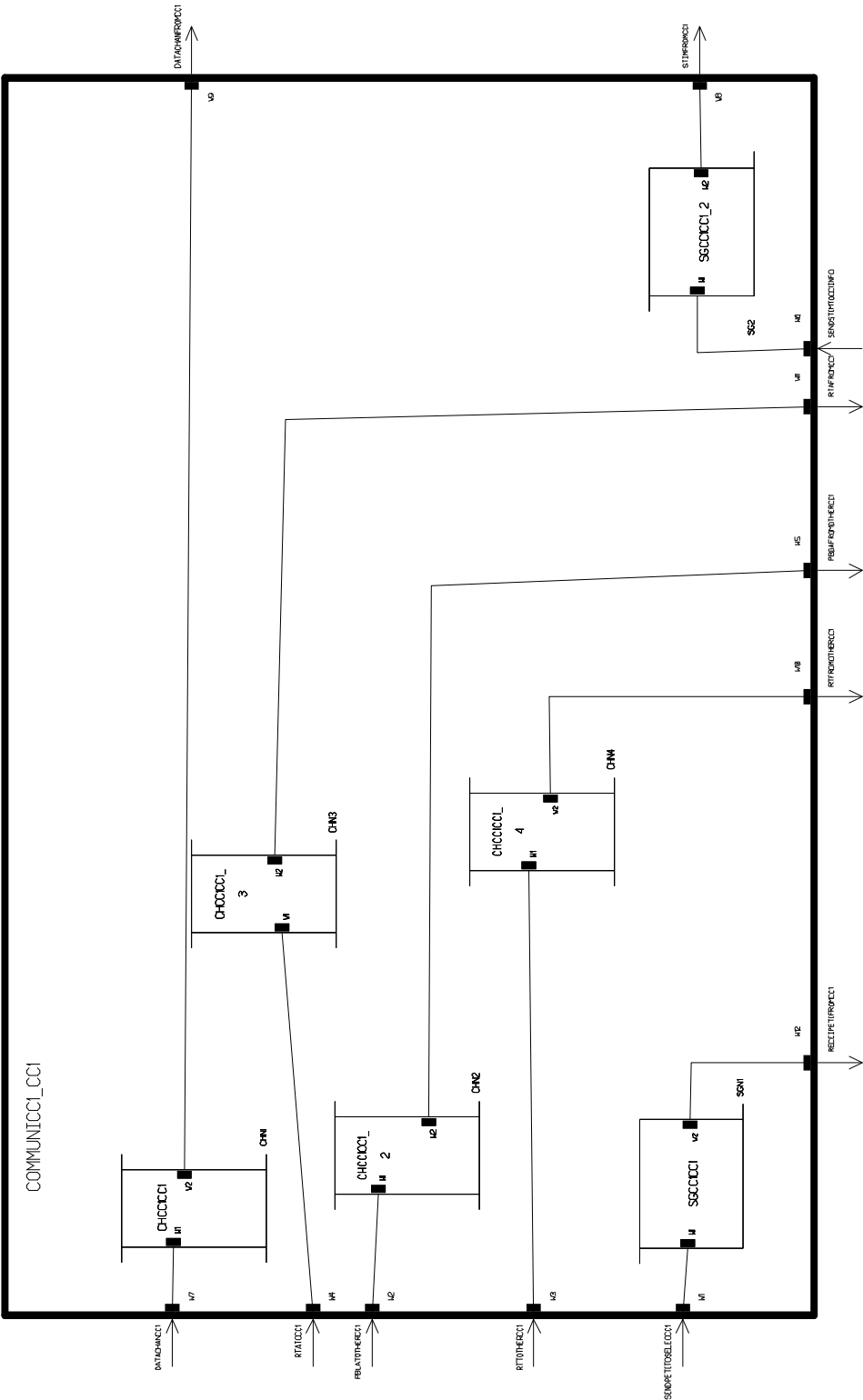


Figure D-6. COMMUNICC1_CC1.

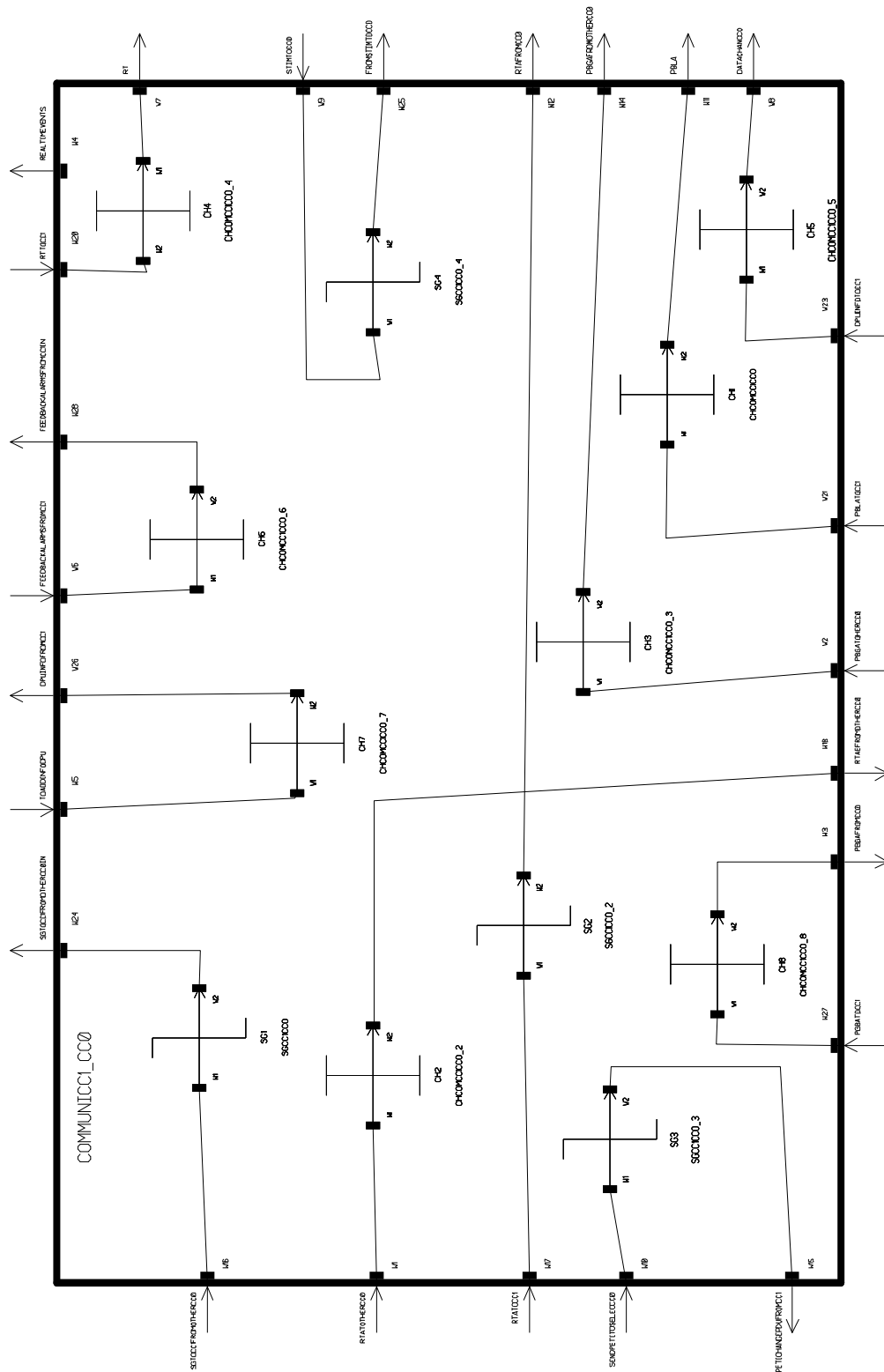


Figure D-7. COMMUNICC1_CC0.

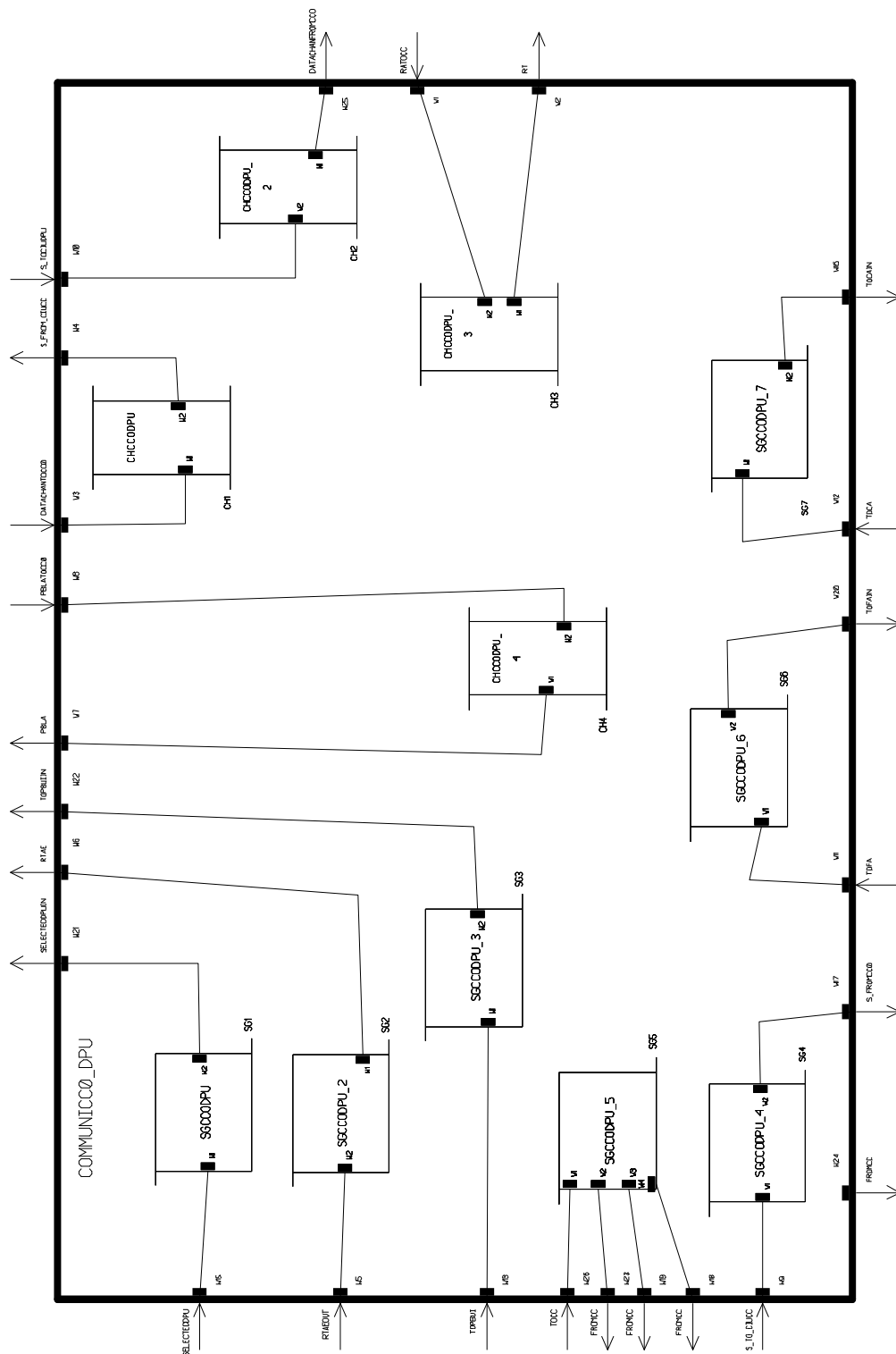


Figure D-8. COMMUNICC0_DPU.

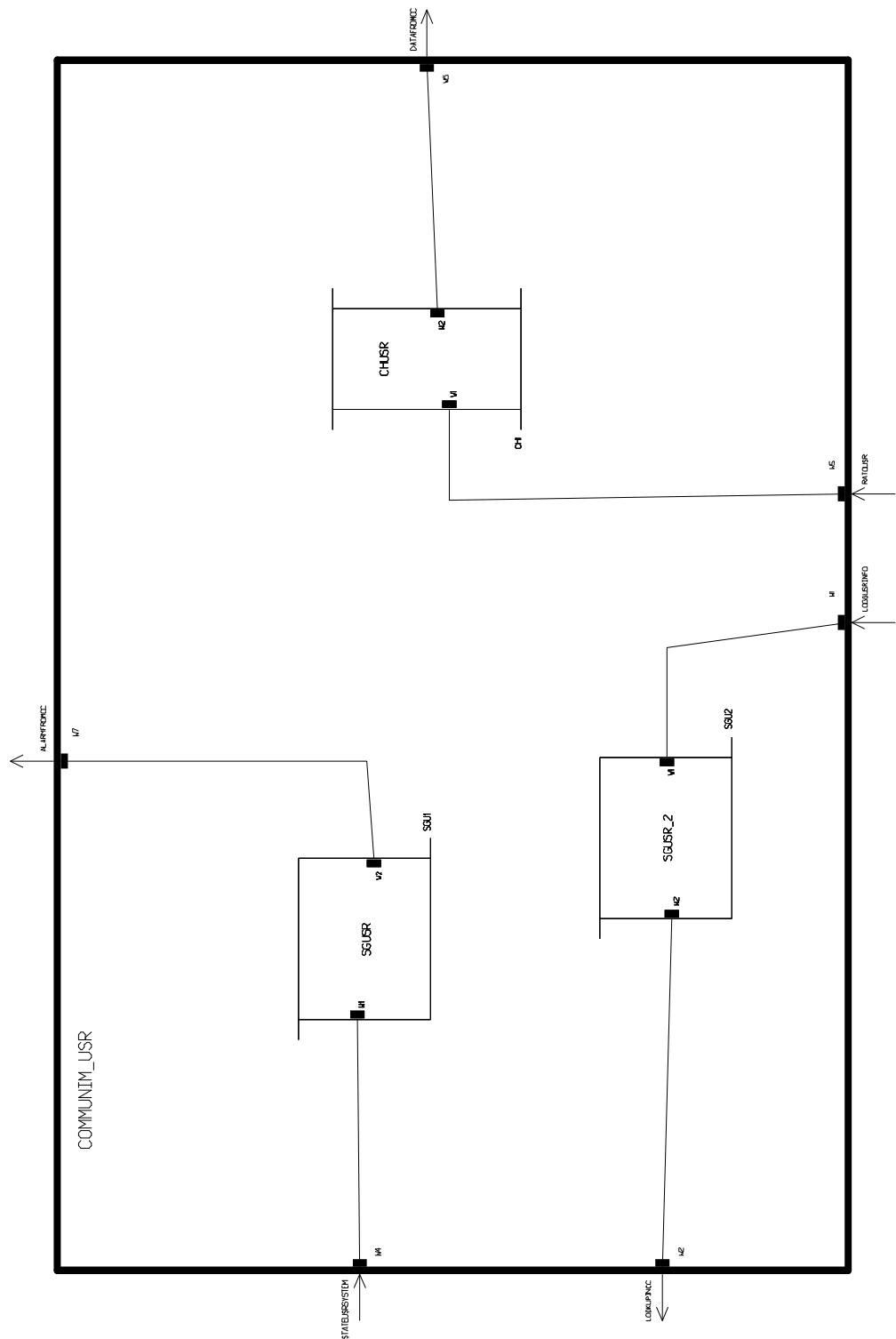


Figure D-9. COMMUNIM_USR.

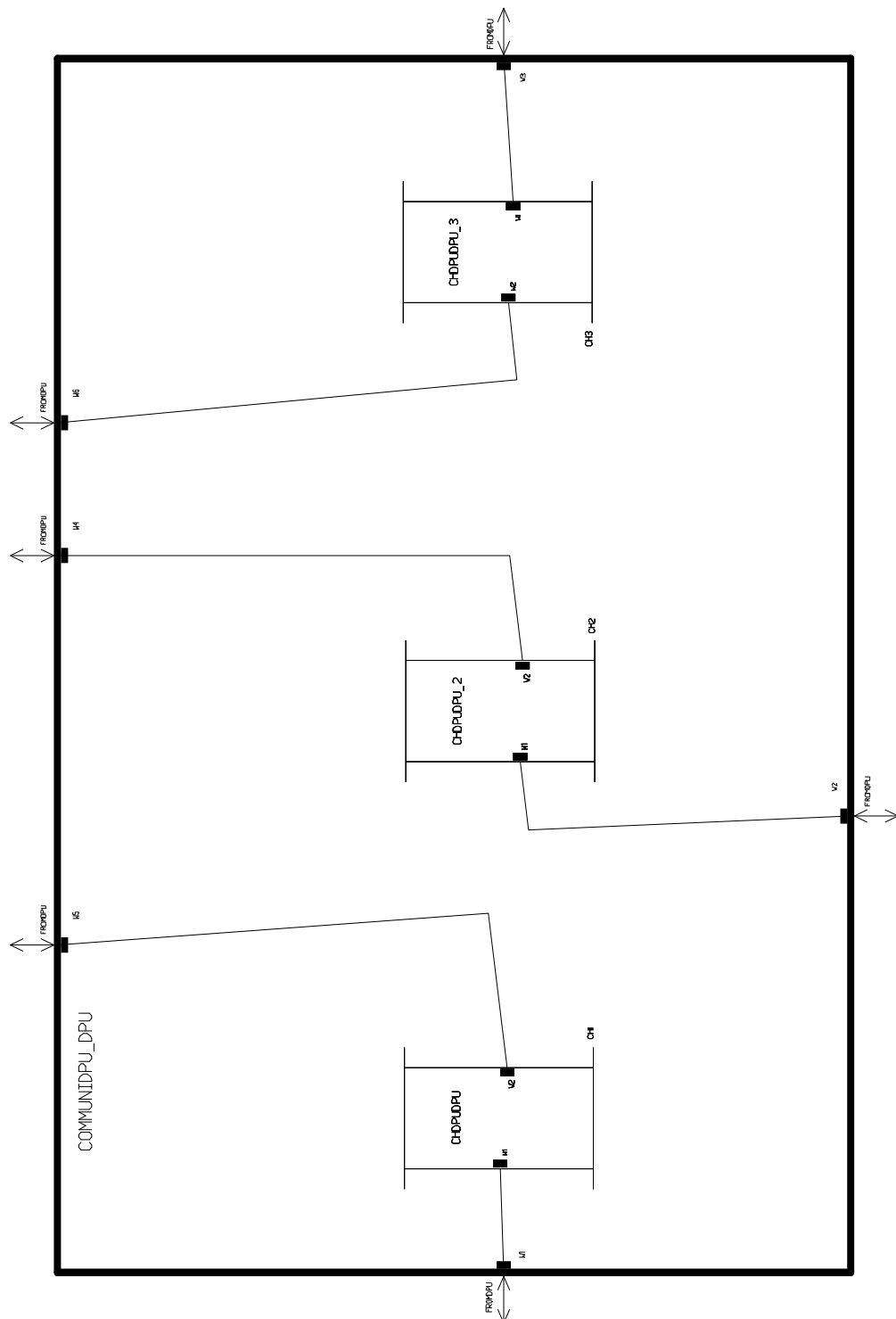


Figure D-10. COMMUNIDPU_DPU.

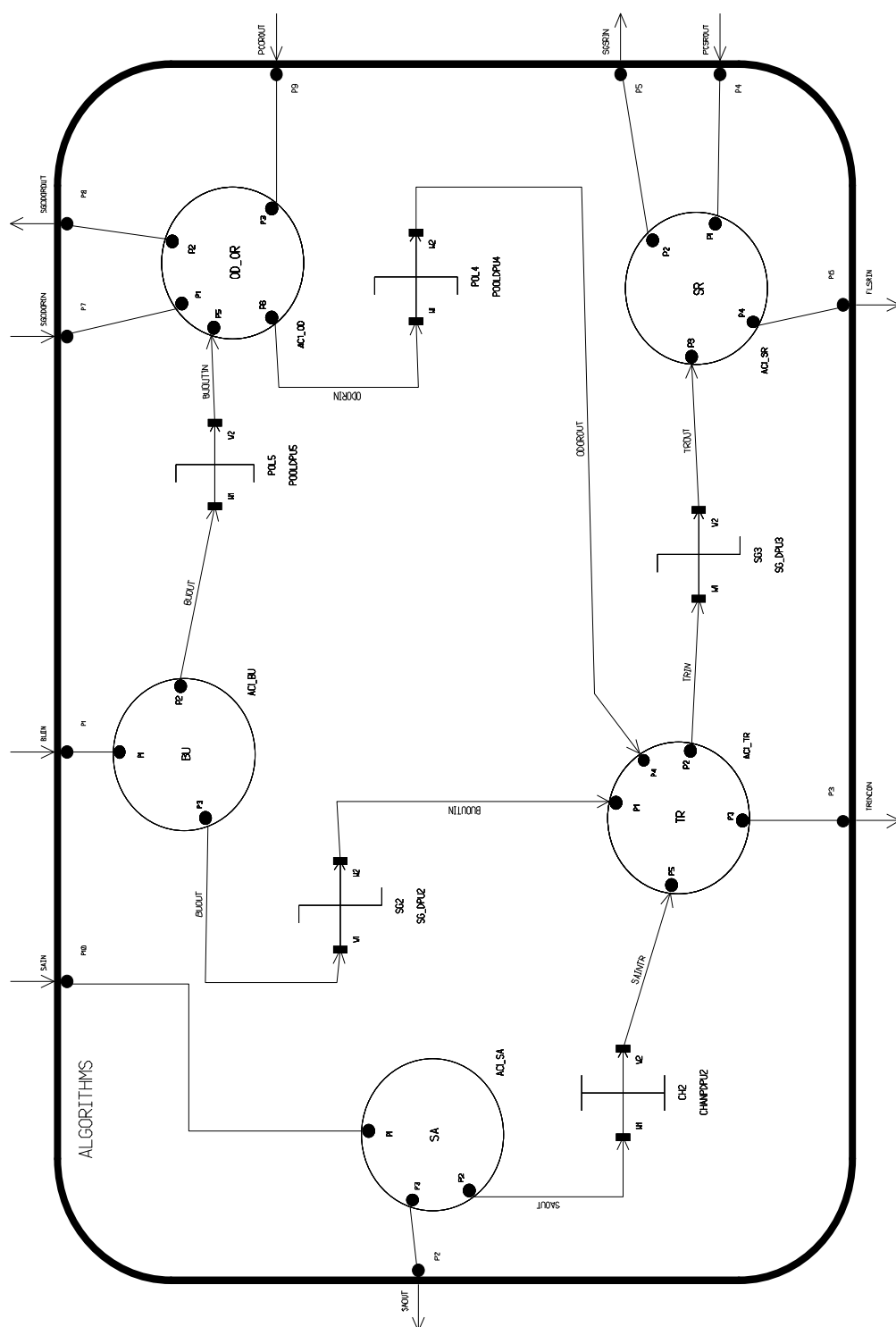


Figure D-11. ALGORITHMS.

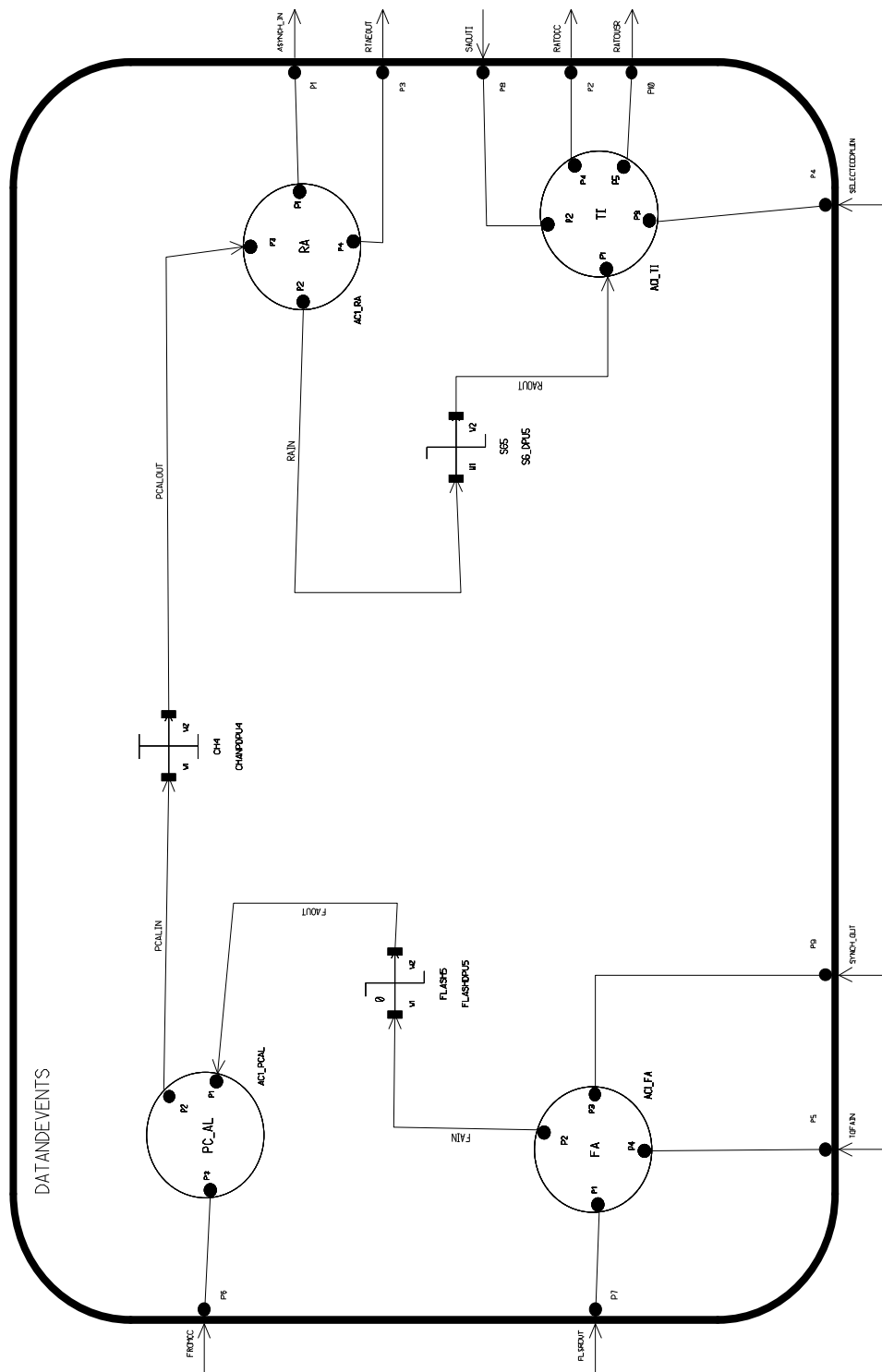


Figure D-13. DATANEVENTS.

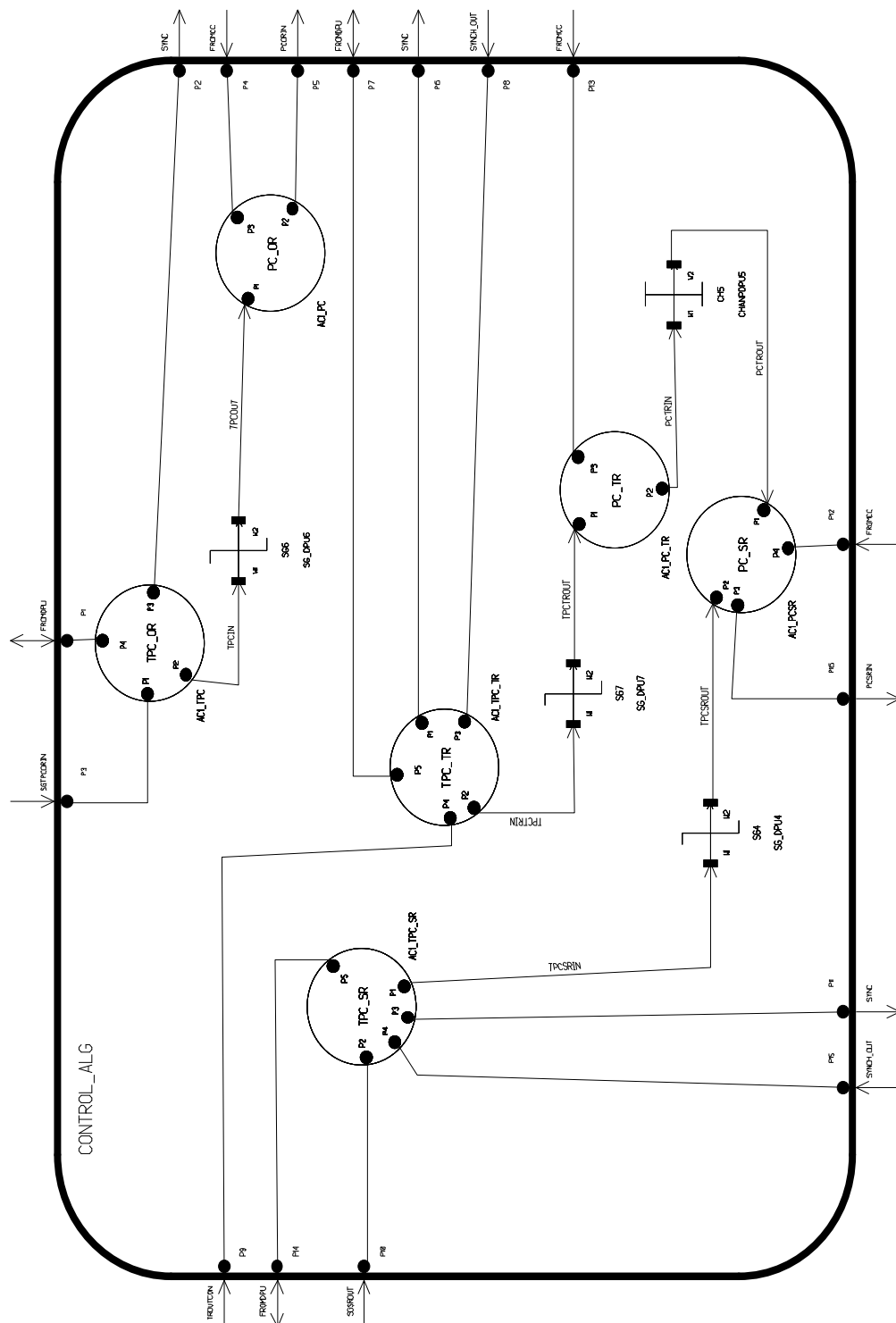


Figure D-14. CONTROL_ALG.

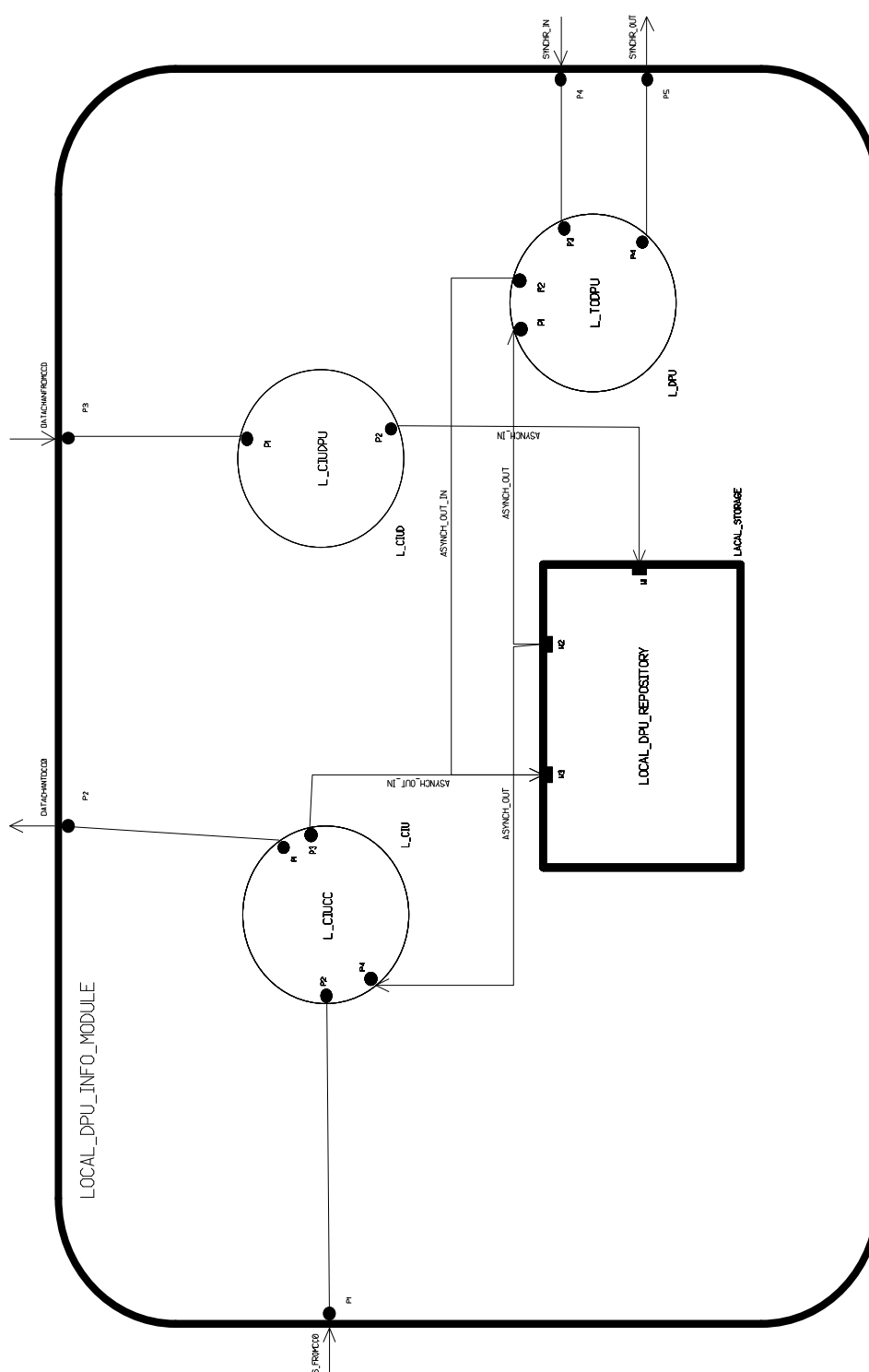


Figure D-15. LOCAL_DPU_INFO_MODULE.

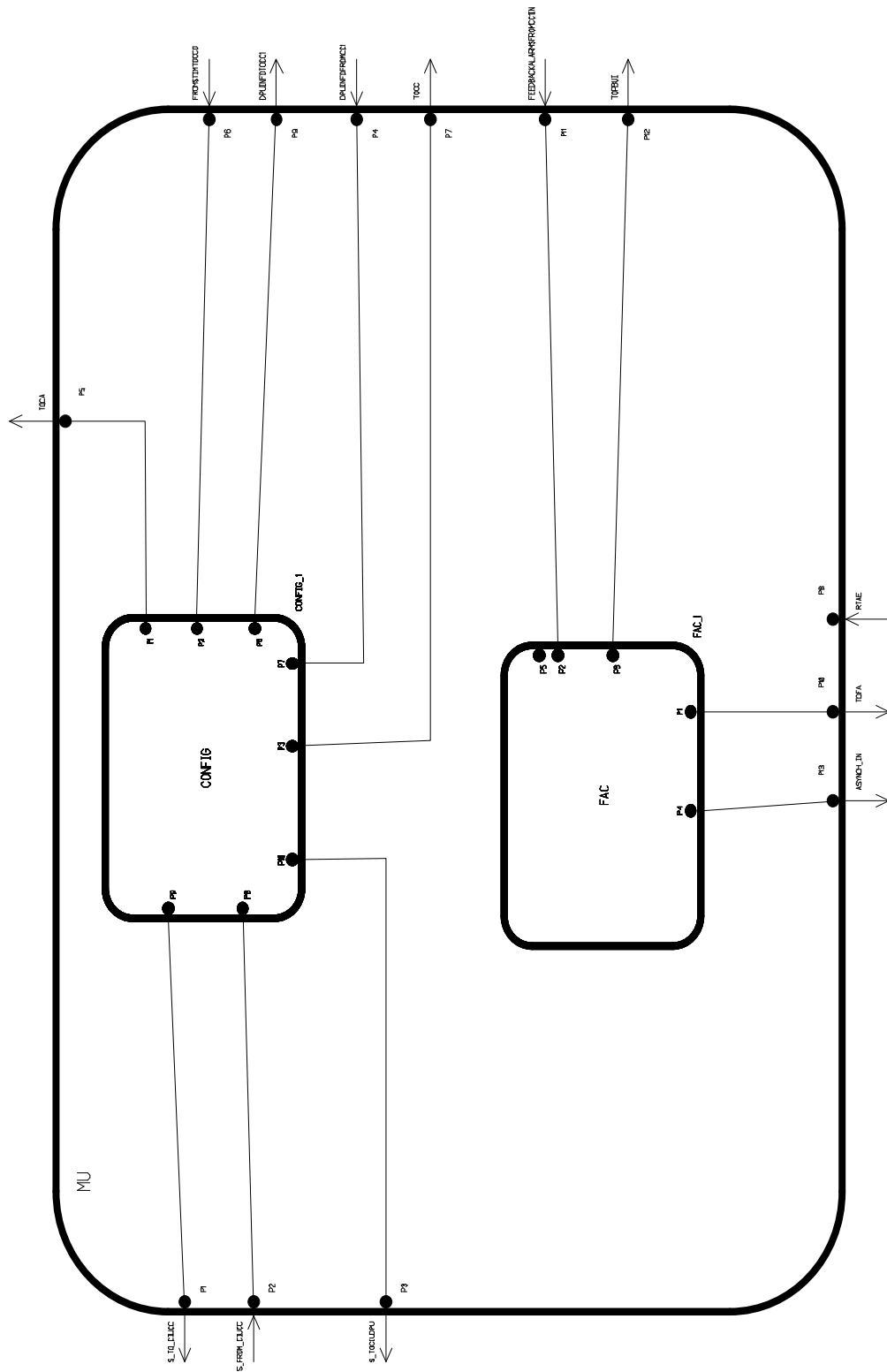


Figure D-16. MU.

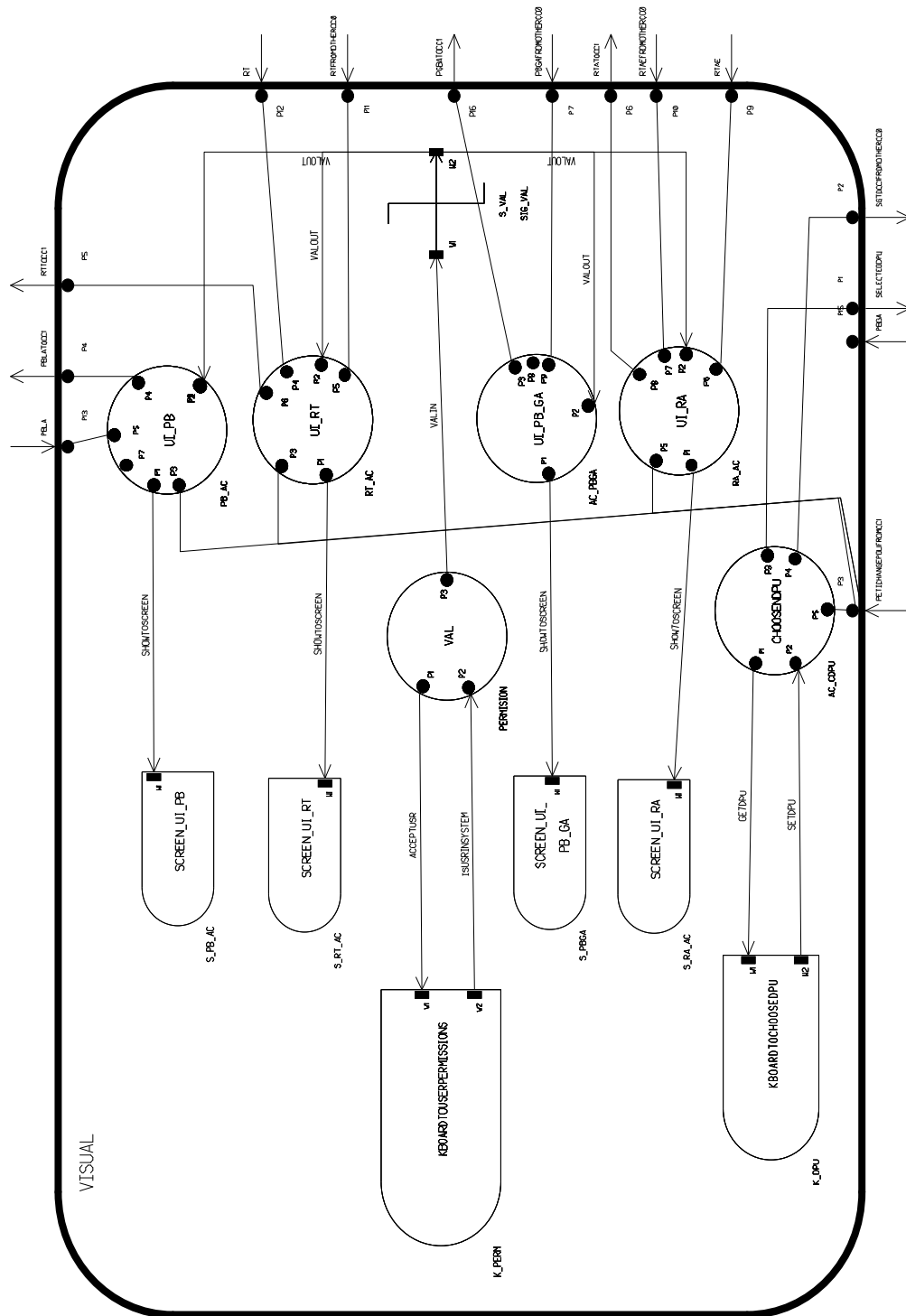


Figure D-17. VISUAL.

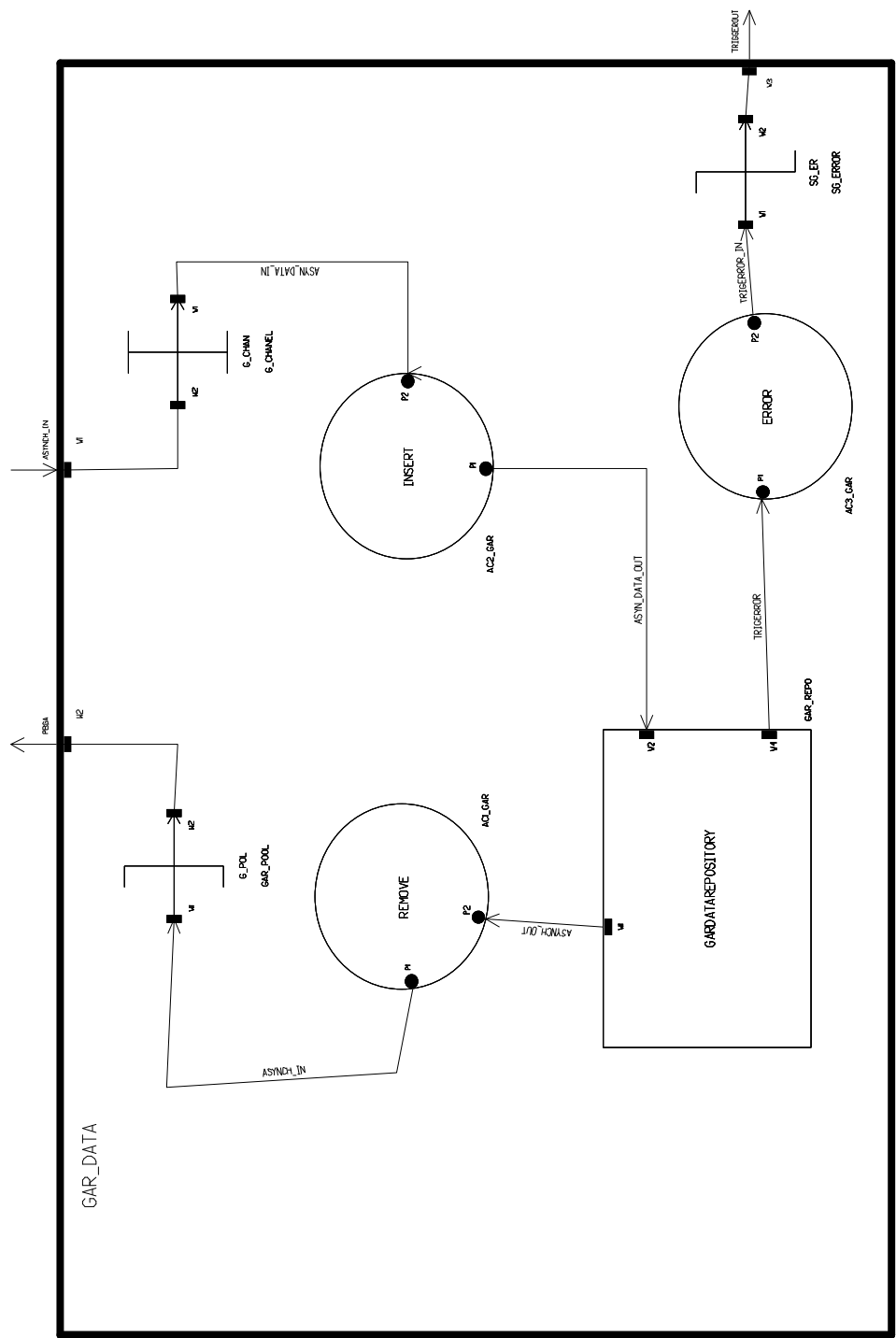


Figure D-18. GAR_DATA.

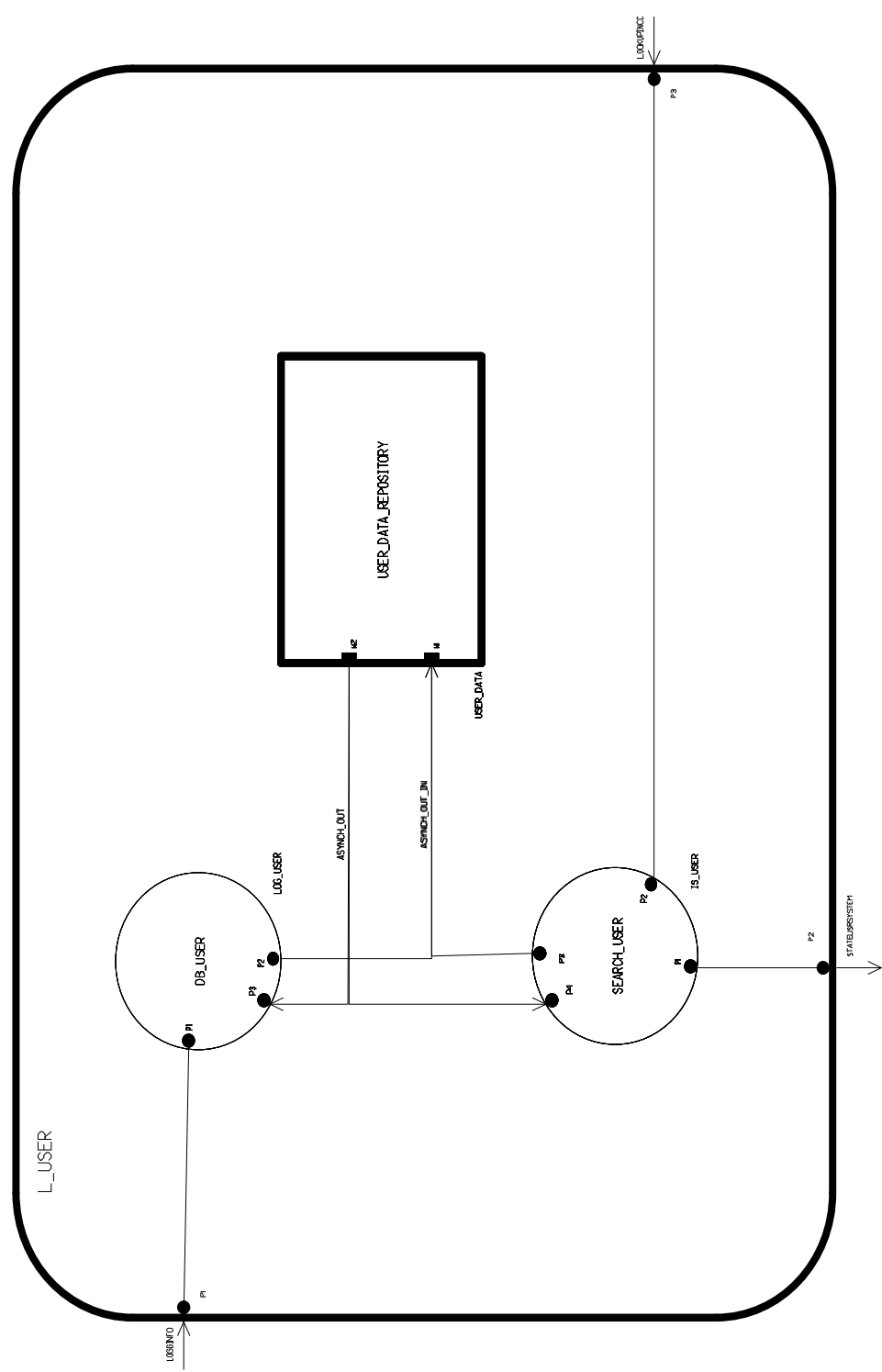


Figure D-19. L_USER.

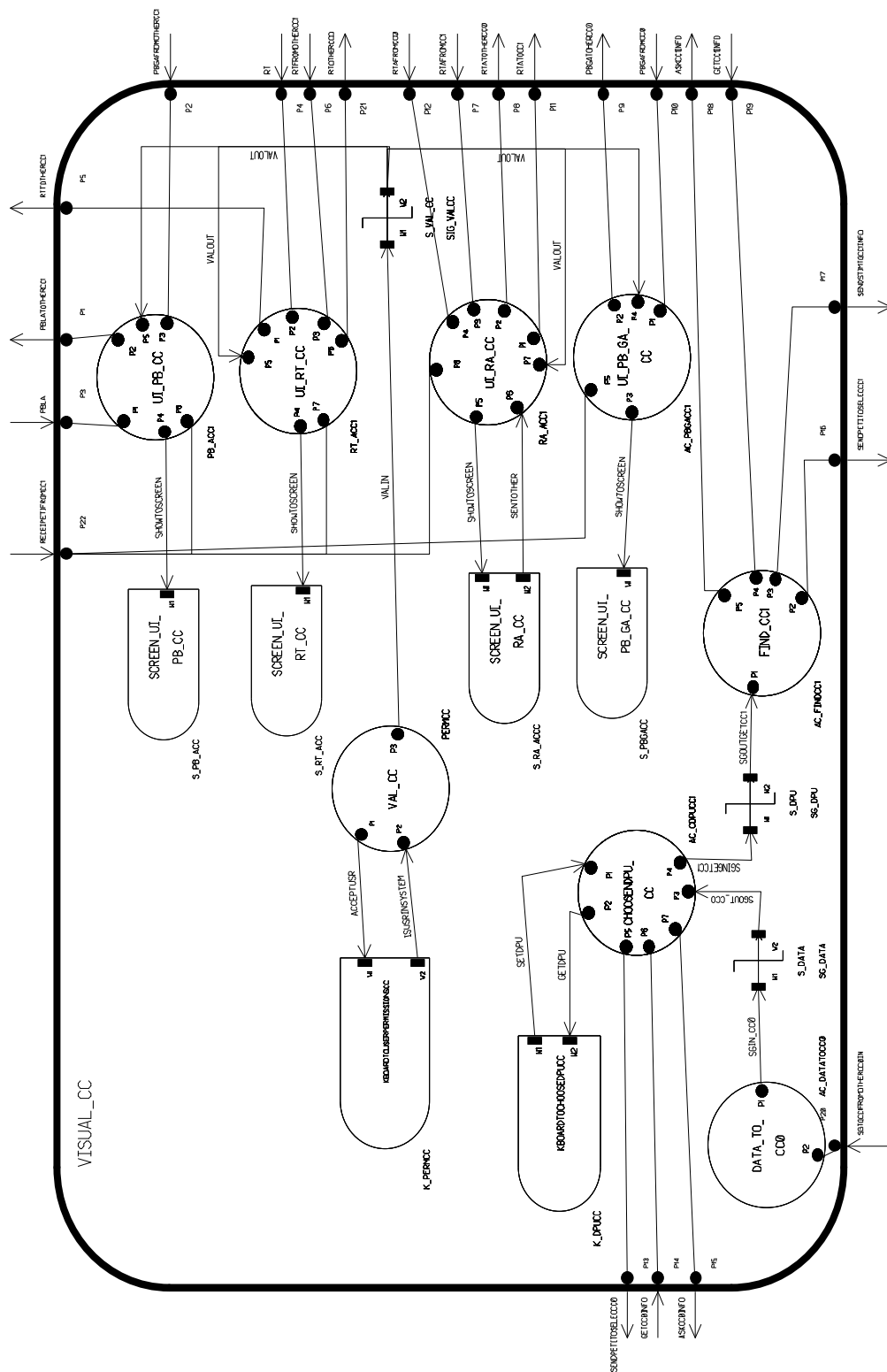


Figure D-20. VISUAL_CC.

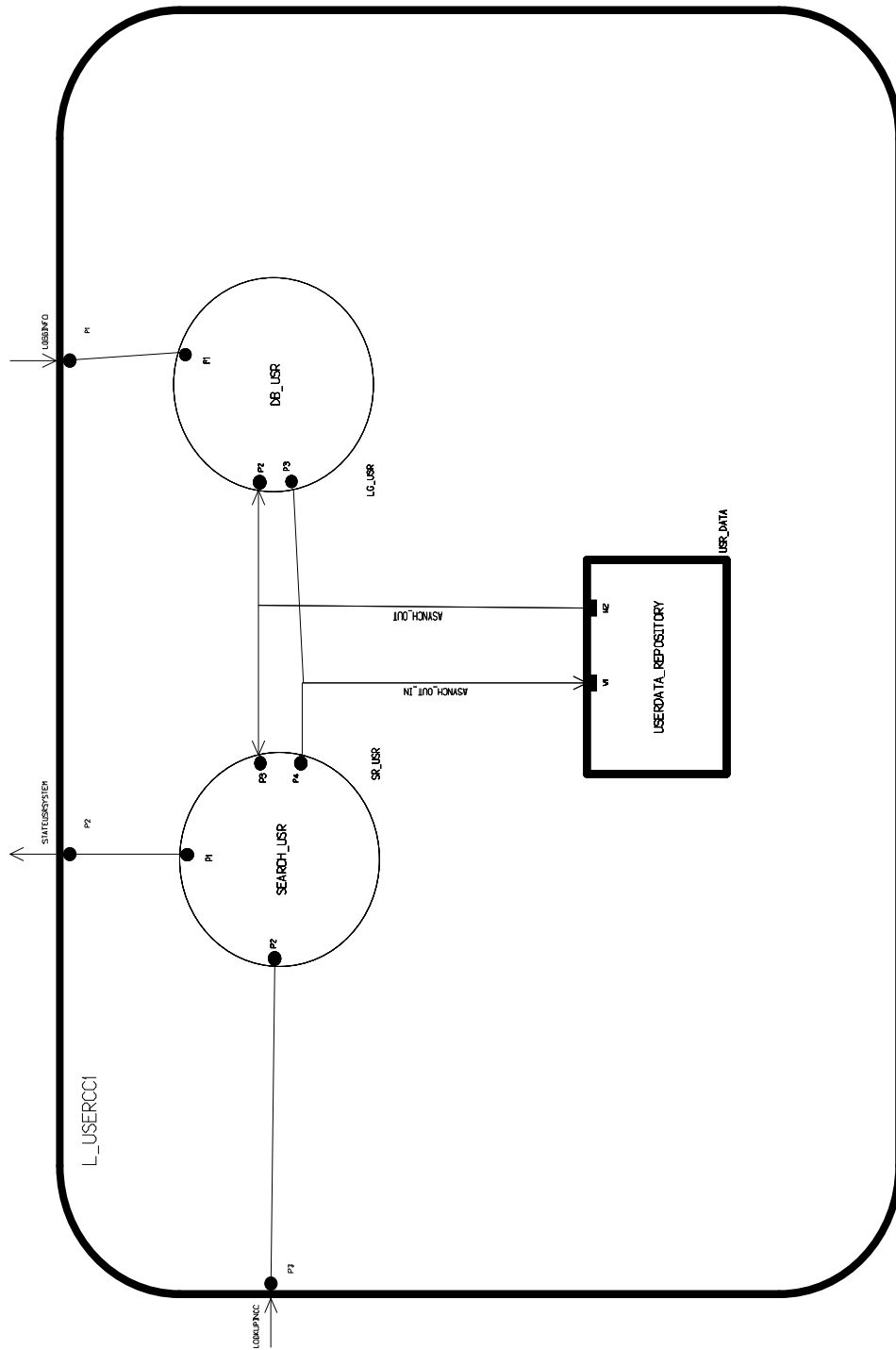


Figure D-21. L_USERCC1.

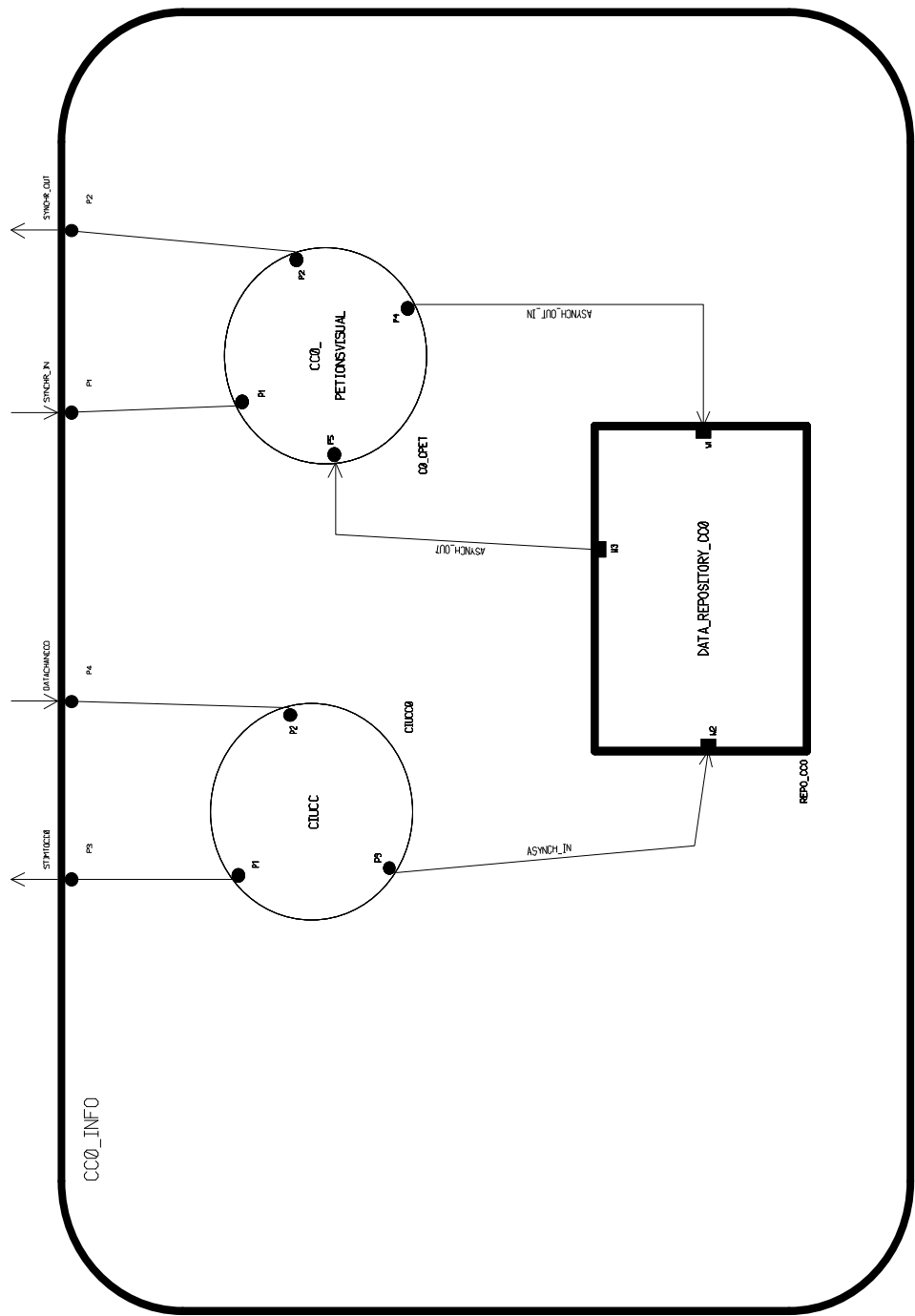


Figure D-22. CC0_INFO.

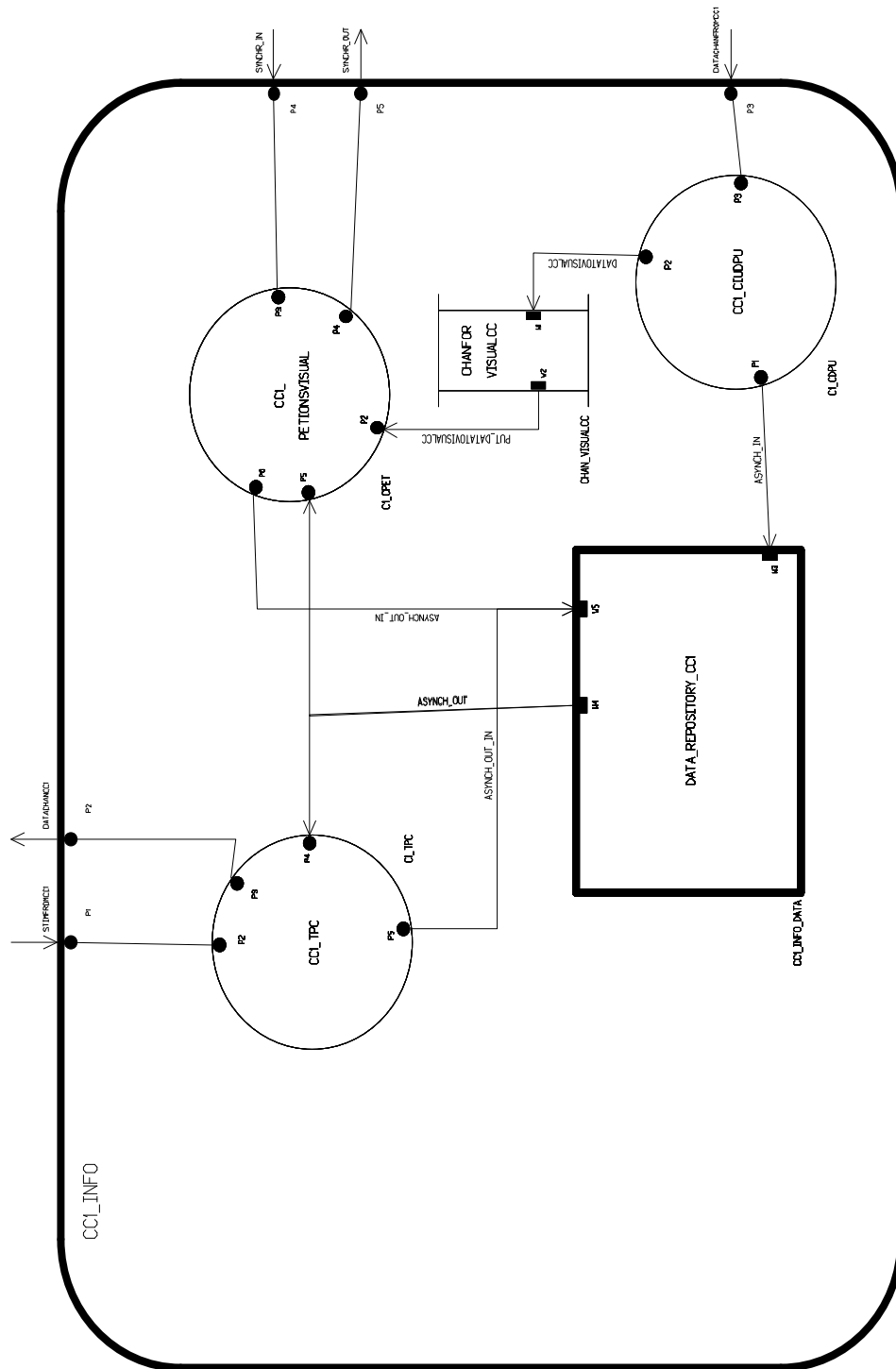


Figure D-23. CC1_INFO.

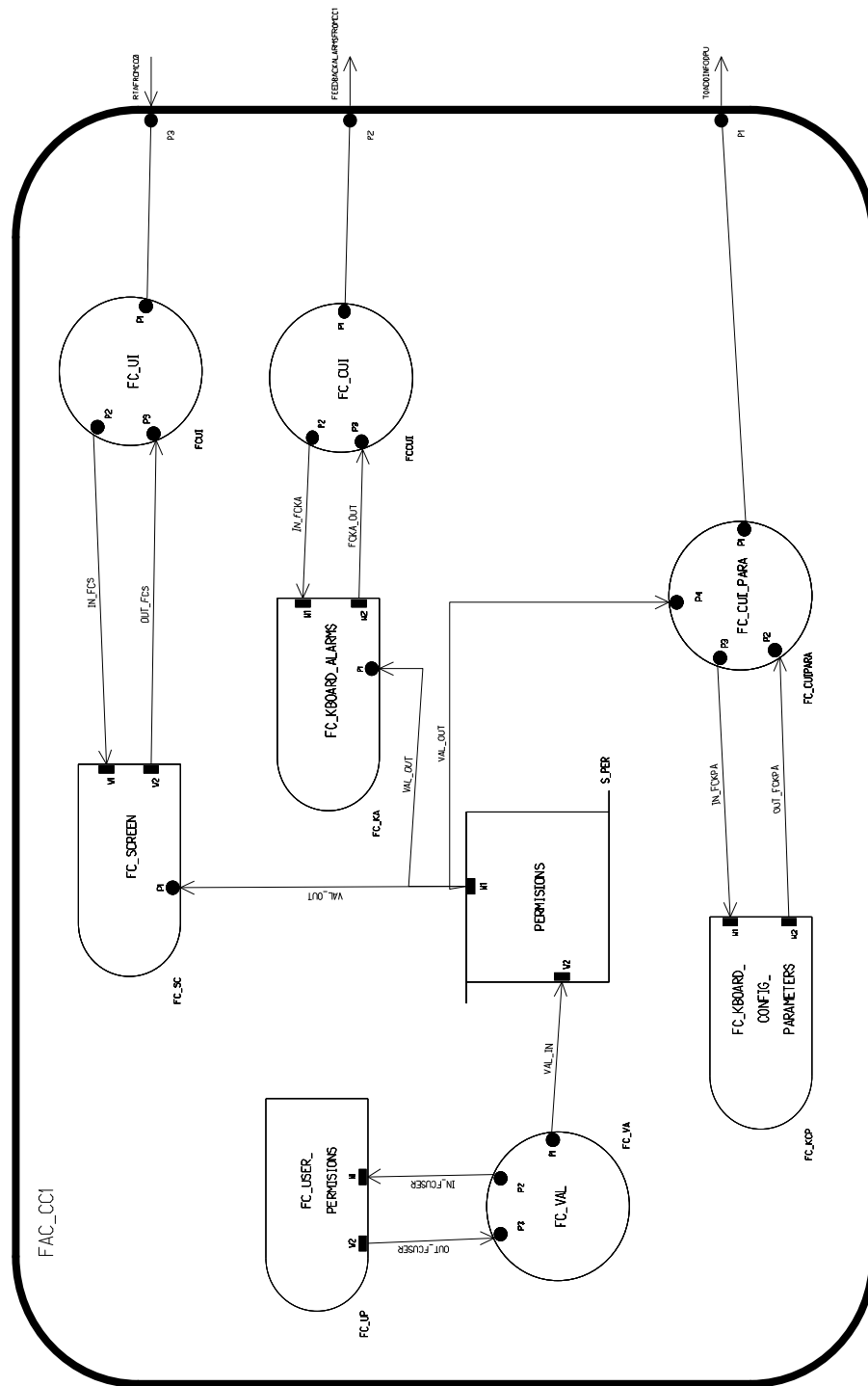


Figure D-24. FAC_CC1.

Figure D-25. CONFIG.

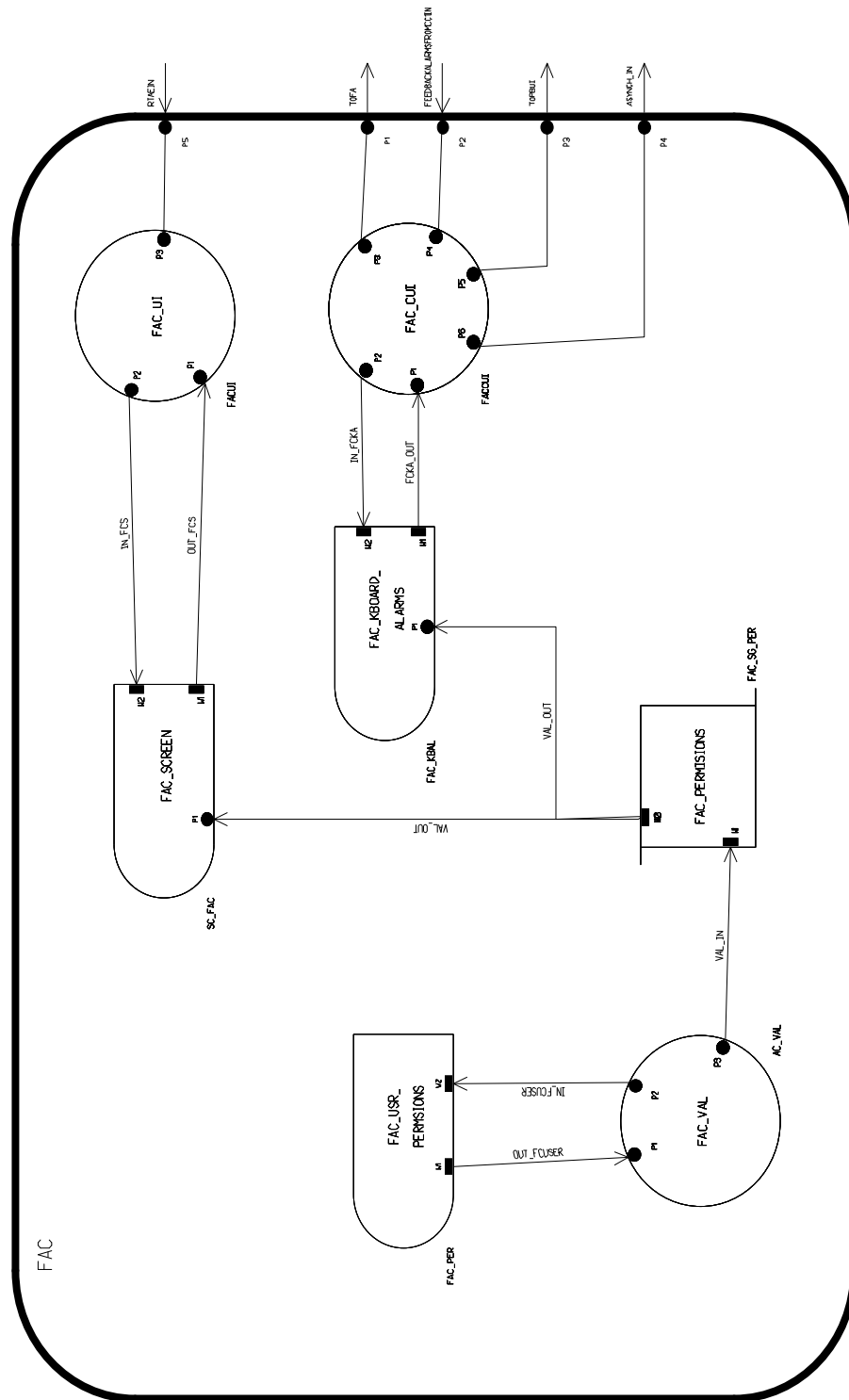


Figure D-26. FAC.

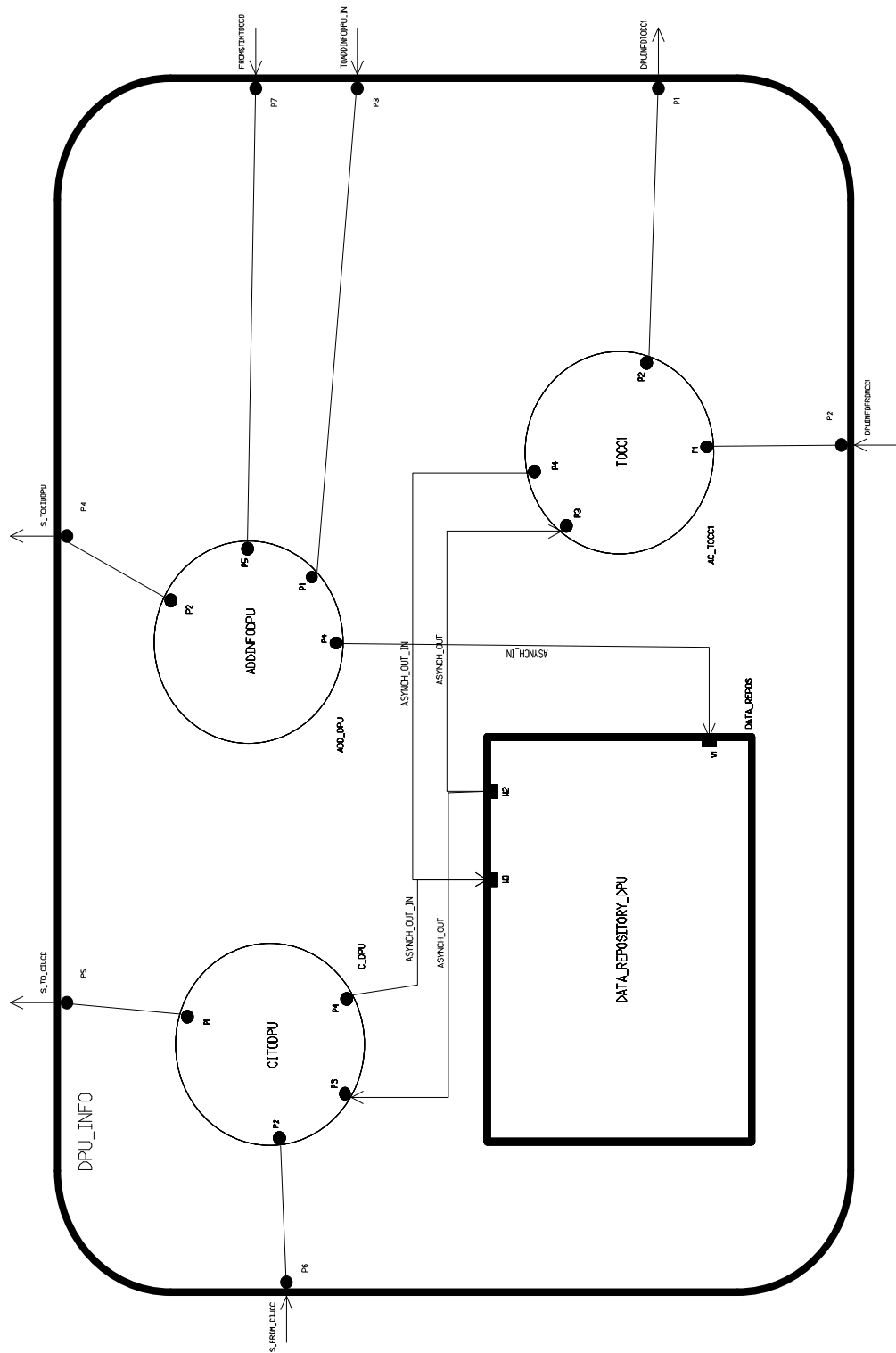


Figure D-27. DPU_INFO.

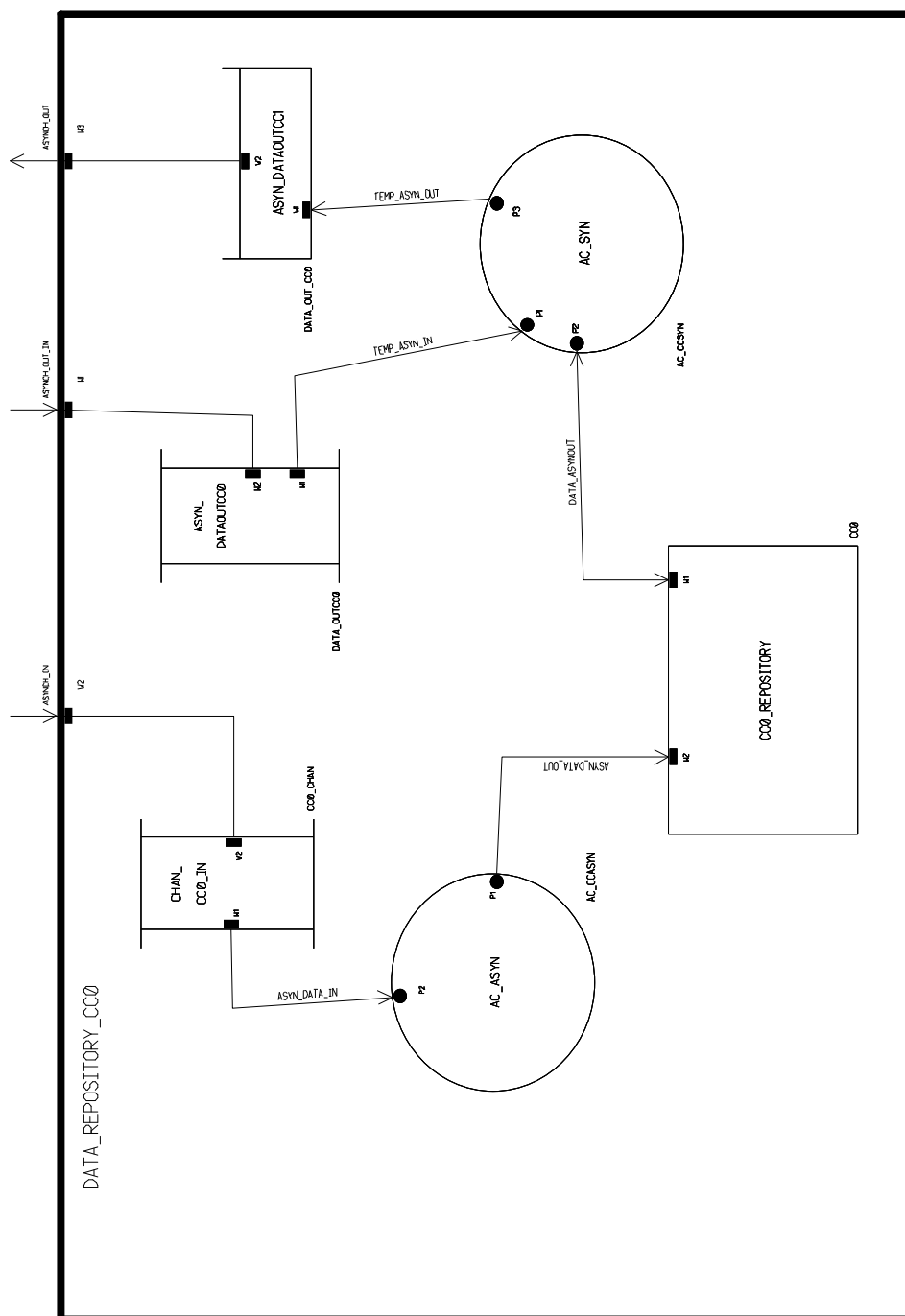


Figure D-28. DATA_REPOSITORY_CC0.

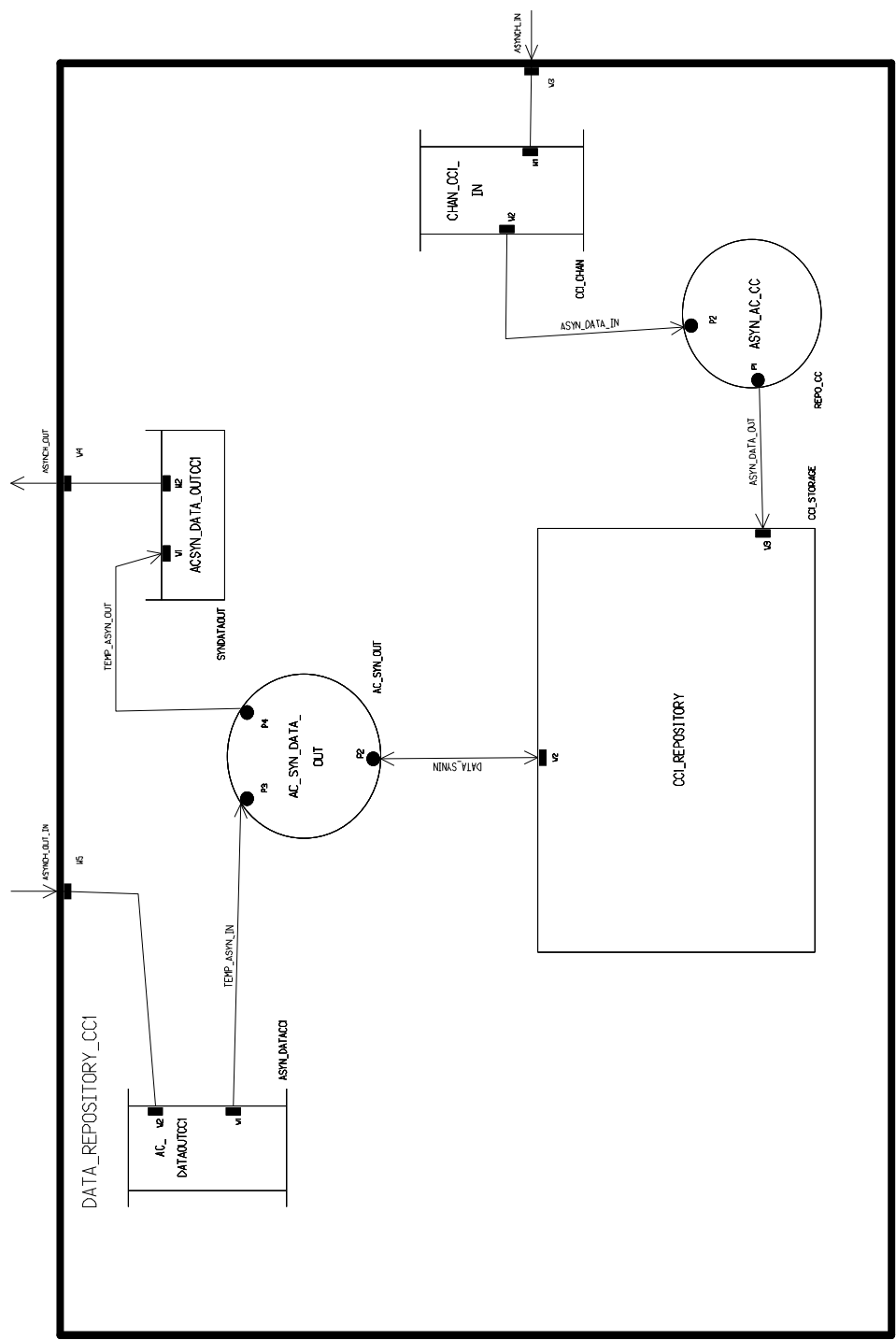


Figure D-29. DATA_REPOSITORY_CC1.

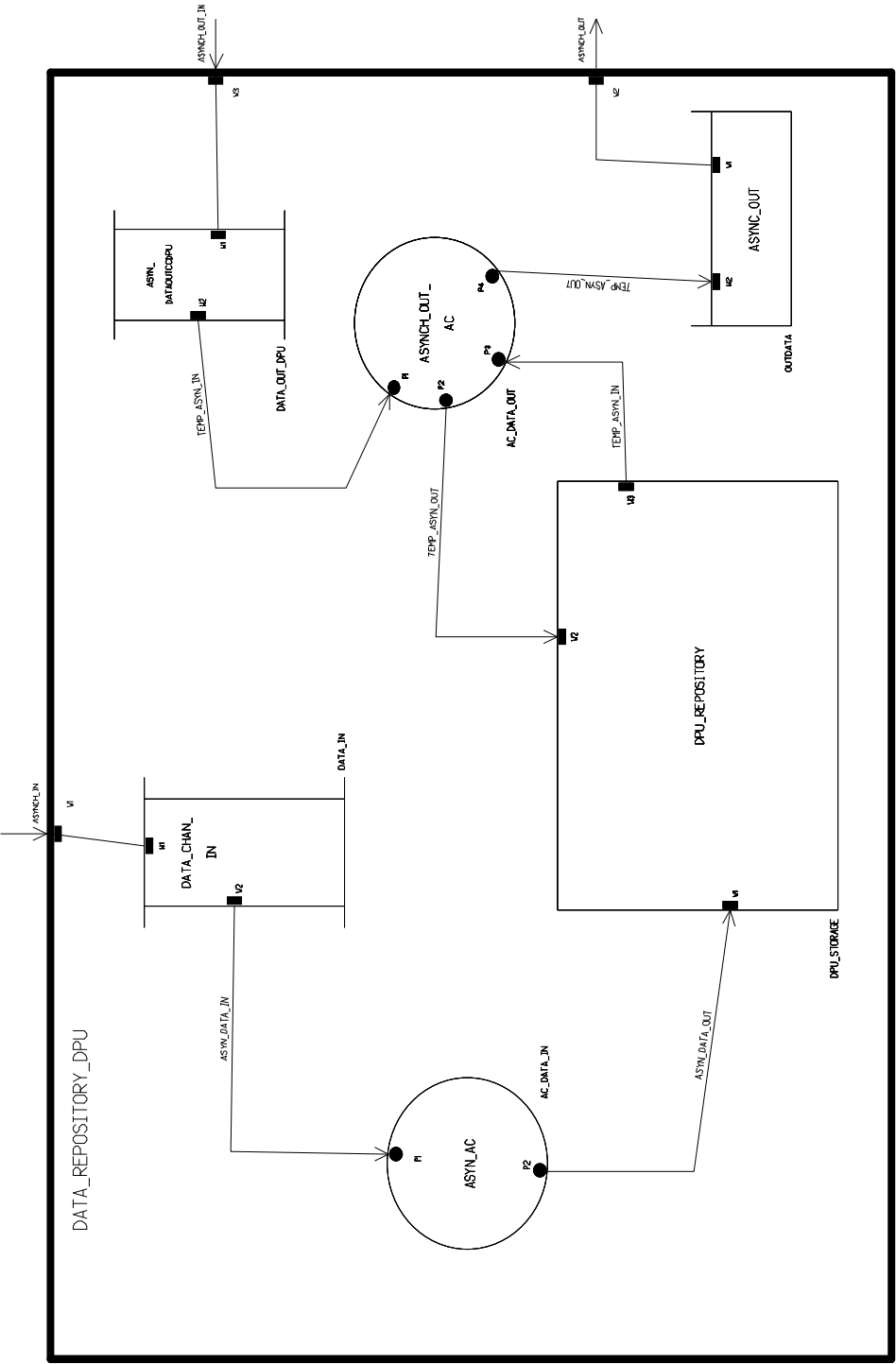


Figure D-30. DATA_REPOSITORY_DPU.

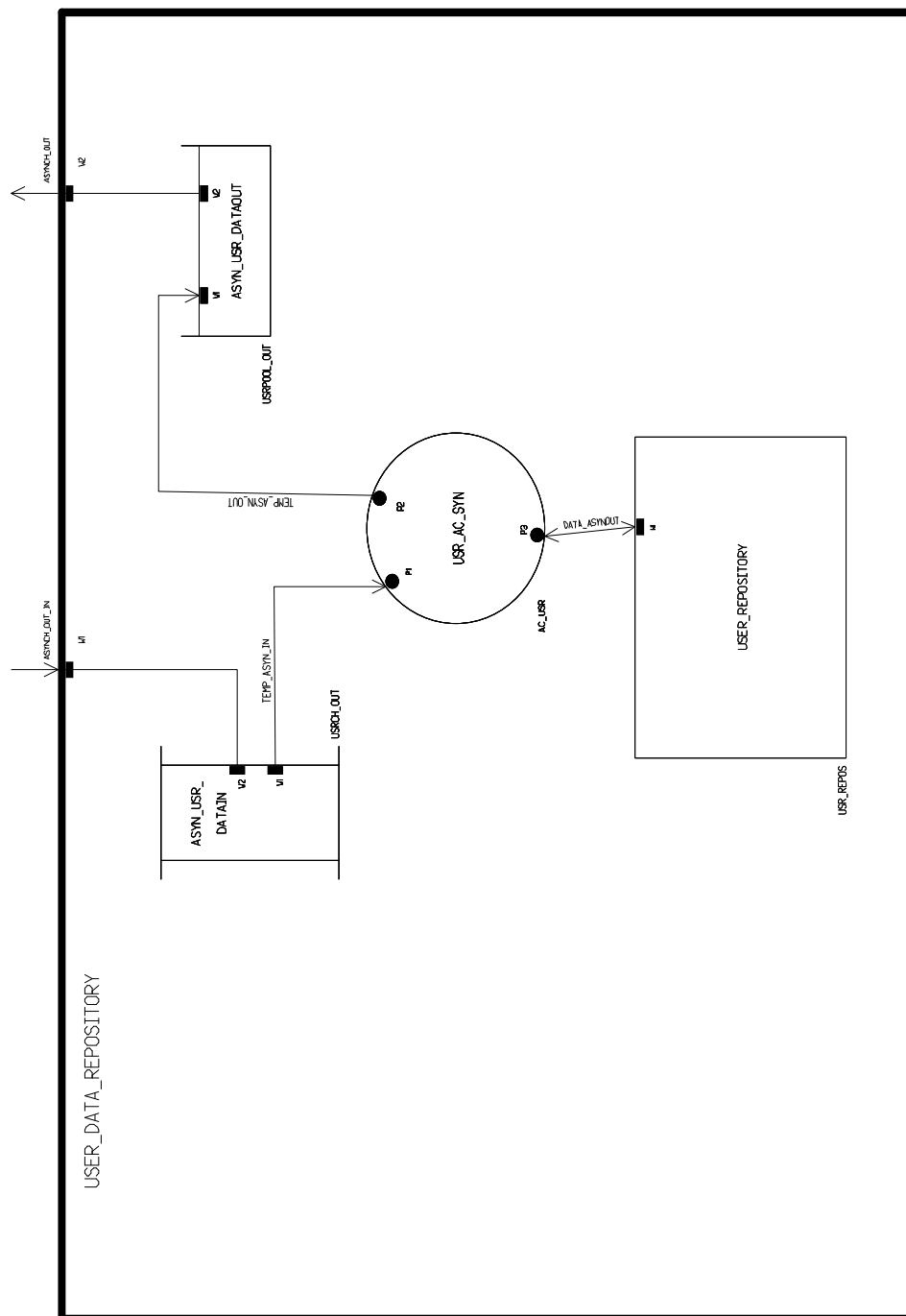


Figure D-31. USER_DATA_REPOSITORY.

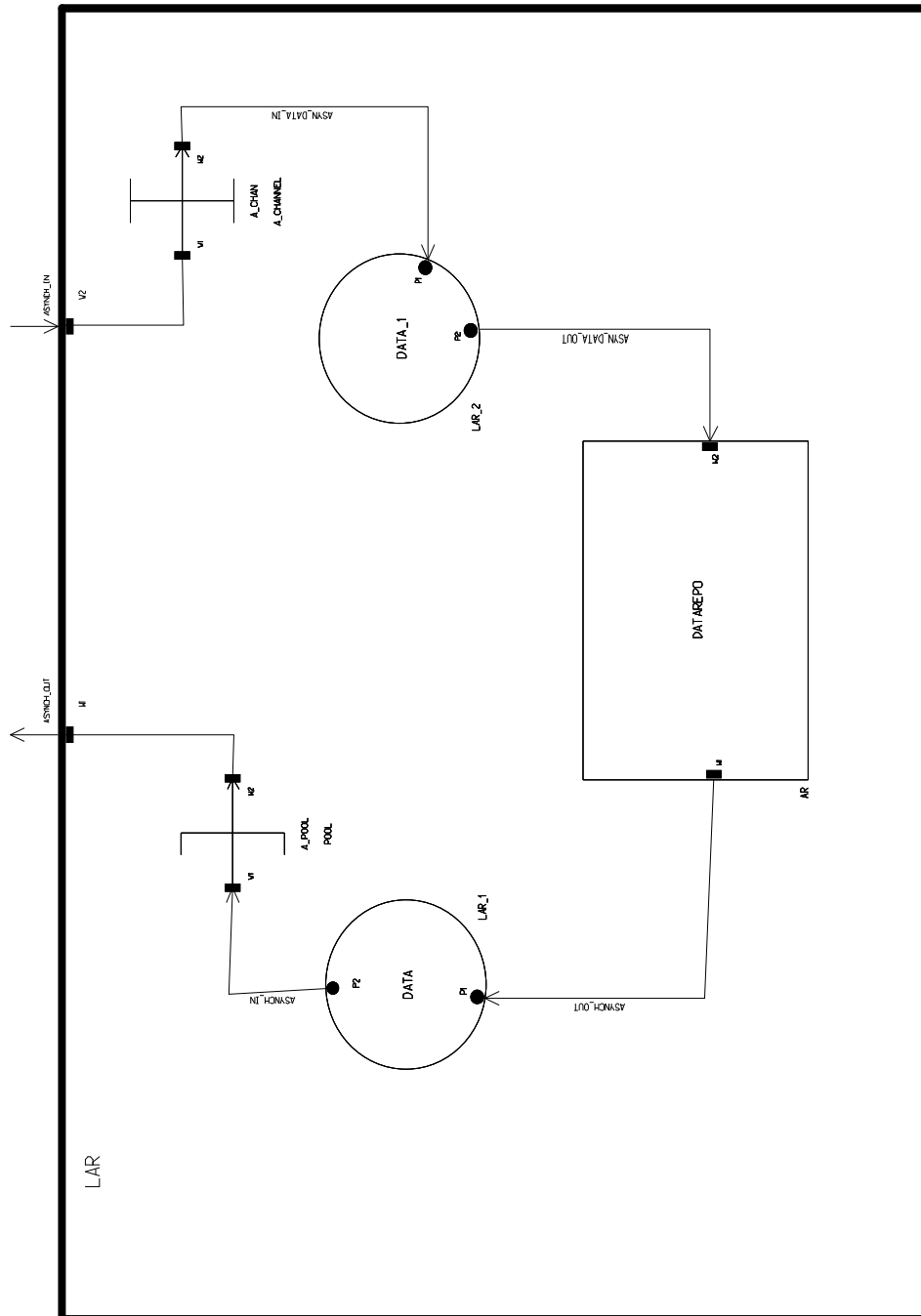


Figure D-32. LAR.

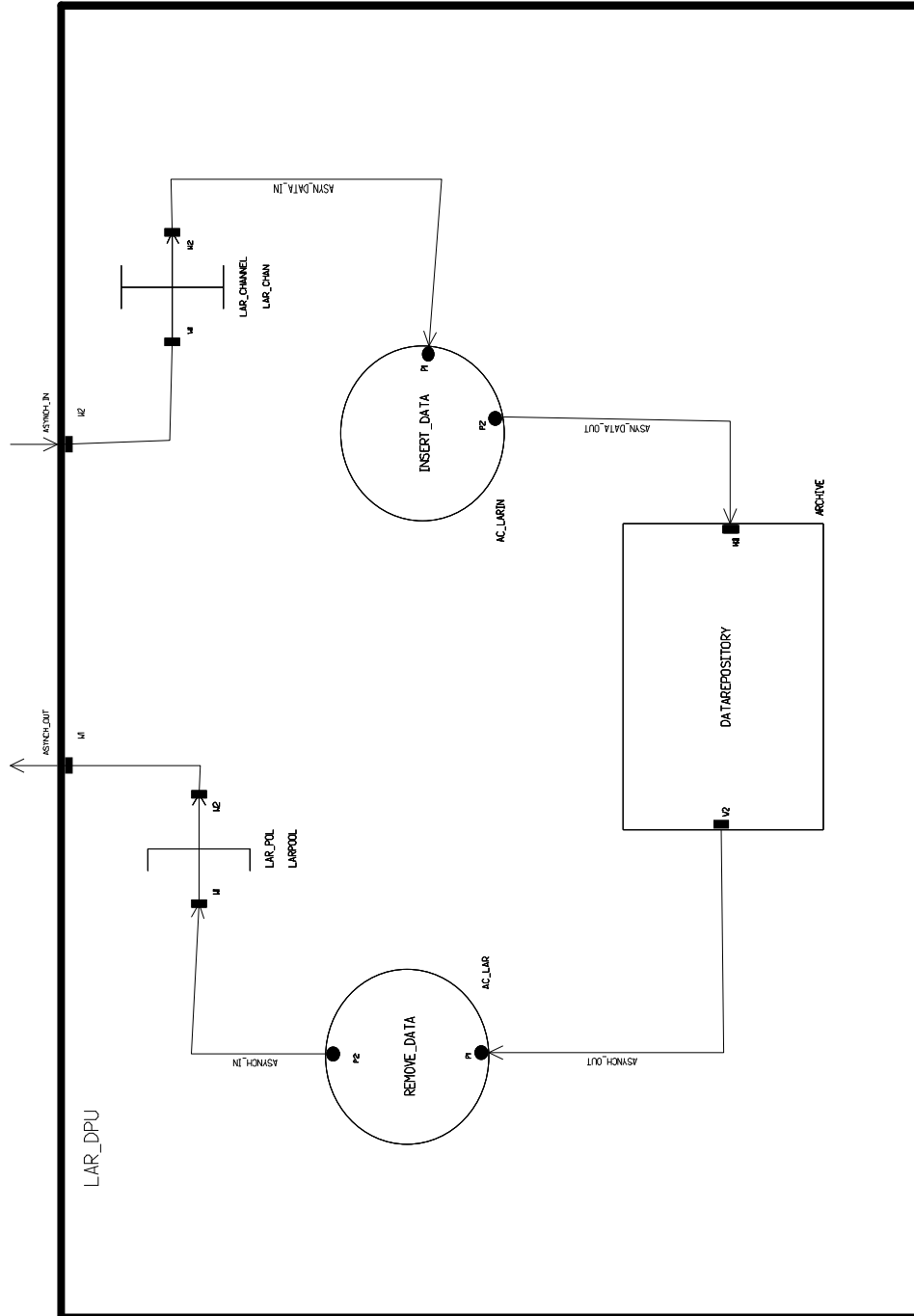


Figure D-33. LAR_DPU.

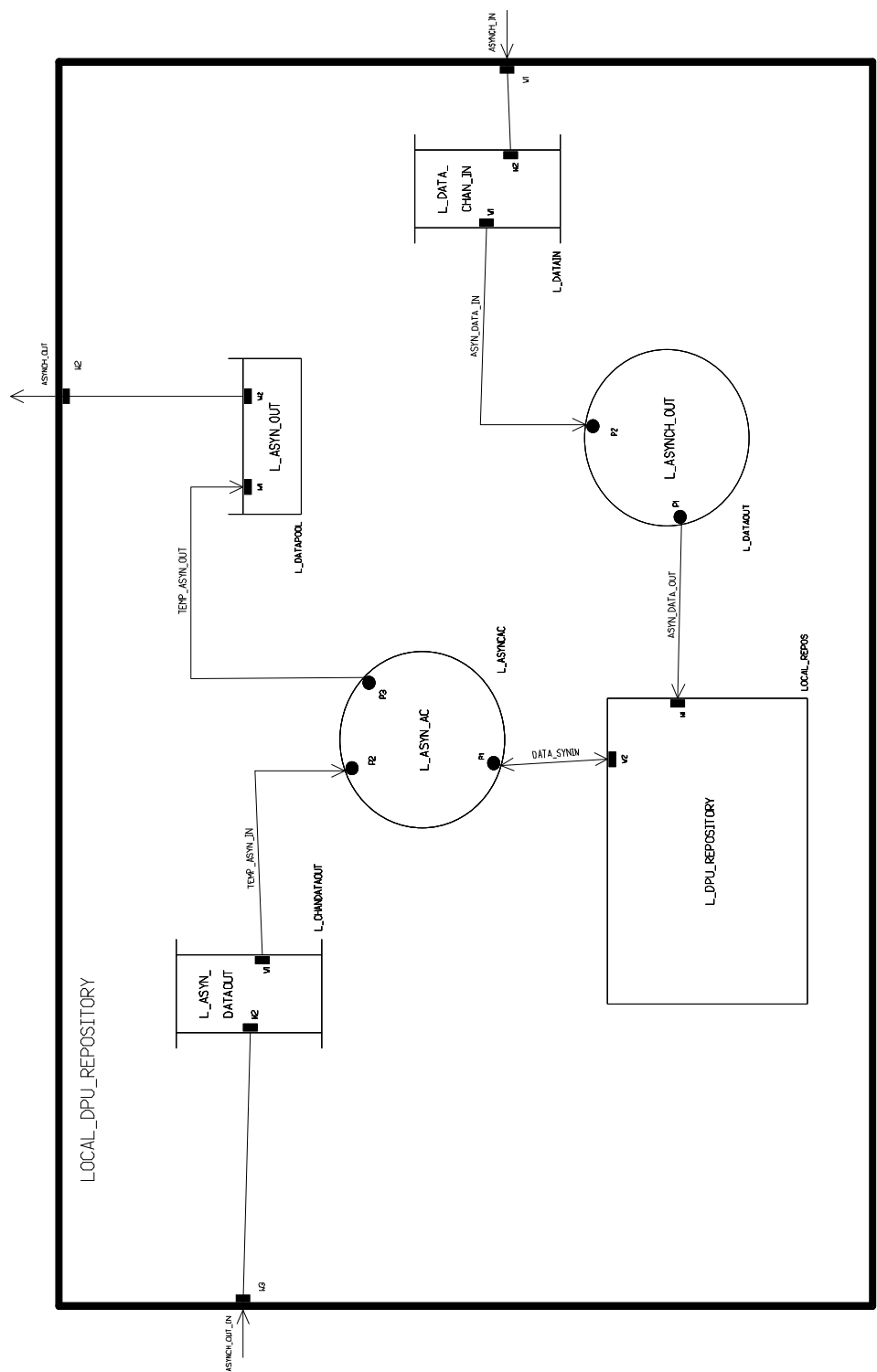


Figure D-34. LOCAL_DPU_REPOSITORY.

List of own publications

- 1) M. Valera, S.A. Velastin, (2005). Intelligent distributed surveillance systems: A Review. In *IEE Proceedings - Vision, Image and Signal Processing*, **152(2)**, pp. 192-204.
- 2) M. Valera, S.A. Velastin, (2004). Real-time architecture for large distributed surveillance systems. *IEE Workshop in Intelligent Distributed Surveillance Systems*, London, pp. 41-45.
- 3) M. Valera, S.A. Velastin, (2004). Real-Time Networks to design a distributed architecture for large real time surveillance systems. *8th World Multi-Conference on Systemics, Cybernetics and Informatics(SCI2004)*.USA, pp. 60-65.
- 4) M. Valera, (2004). Design method for real-time intelligent distributed wide-area surveillance system. *PREP2004, University of Hertfordshire, UK*.
- 5) M. Valera, S.A. Velastin (2003) An Approach For Designing A Real-Time Intelligent Distributed Surveillance System. *Symposium on Intelligent Distributed Surveillance Systems, IEE, UK*, pp. 6/1-6/5.

References

1. [ADVISOR 2003]
ADVISOR (2003). *ADVISOR*. [Online]. Available at <<http://www-sop.inria.fr/orion/ADVISOR/>>. [Accessed 11th March 2006].
2. [Almeida et al. 2002]
Almeida, L., Pedreiras, P., Alberto, J., Fonseca, G. The FFT-CAN Protocol: Why and How. *IEEE Transactions on Industrial Electronics*, **49(6)**, pp.1189-1201.
3. [Arulampalam et al. 2002]
Arulampalam, S., Maskell, S., Gordon, N., Clapp, T. (2002). A Tutorial on Particle Filters for On-line Non-linear/Non-Gaussian Bayesian Tracking. *IEEE Transactions on Signal Processing*, **50(2)**, pp.174-188.
4. [Barni et al. 2000]
Barni, M., Bartolini, F., Cappellini, V., Piva, A. (2000). Digital Watermarking for the Authentication of AVS Video Sequences. in *Multimedia Video Based Surveillance Systems*, G. L. Foresti, G.L., Mahonen, P., Regazzoni, C.S. (eds.), Kluwer Academic Publishers, Boston, pp. 186-196.
5. [Bate 1986]
Bate, G. (1986). Mascot 3: an informal introductory tutorial. *Software Engineering Journal*, **1(3)**, 95-102.
6. [Bennewitz et al. 2002]
Bennewitz, M., Burgard, W., Thrun, S. (2002). Using EM to learn motion behaviours of persons with mobile robots. In *Proceedings of the Conference on Intelligent Robots and Systems (IROS)*, Switzerland, pp. 502 – 507.
7. [Berris et al. 2003]
Berris, W., Price, W.G., Bober, M.Z. (2003). The use of MEG-7 for intelligent analysis and retrieval in video surveillance. In *Proceedings of the IEE Workshop on Intelligent Distributed Surveillance Systems*, London, pp. 8/1-8/5.
8. [Beymer et al. 1997]
Beymer, D., McLauchlan, P., Coifman, B., Malik, J. (1997). A Real-Time Computer Vision System for Measuring Traffic Parameters. In *Proceedings of the*

- 1997 Conference on Computer Vision and Pattern Recognition*, IEEE Computer Society, pp. 495-502.
9. [Black et al. 2003]
Black, J., Ellis, T., Rosin, P. (2003). A Novel Method for Video Tracking Performance Evaluation. *The Joint IEEE International Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance*, pp. 125-132.
 10. [Boasson 2002]
Boasson, M. (2002). Embedded Systems Unsuitable for Object Orientation. In J. Blieberger, J., Strohmeier, A. (Eds.). *Lecture Notes in Computer Science, Reliable Software Technologies - Ada-Europe*. Chapter 1.
 11. [Booch 1991a]
Booch, G. (1991). *Object Oriented design: with applications*. Wokingham, Redwood City, California, USA. Benjamin and Cummings.
 12. [Booch et al. 2000b]
Booch, G., Jacobson, I., Rumbaugh, J. (2000). *UML Distilled Second Edition, A brief Guide to the Standard Object Modelling Language*. USA. Addison Wesley Longman, Inc.
 13. [Boult et al. 2001]
Boult, T.E., Micheals, R.J., Gao, X., Eckmann, M. (2001). Into the woods: visual surveillance of non-cooperative and camouflaged targets in complex outdoor settings. *Proceedings of the IEEE*, **89(1)**, pp. 1382-1401.
 14. [Brodsky et al. 2001]
Brodsky, T., Cohen, R., Cohen-Solal, E., Gutta, S., Lyons, D., Philomin, V., Trajkovic, M. (2001). Visual surveillance in retail stores and in the home. In *Advanced Video-based Surveillance Systems*, Kluwer Academic Publishers, Boston, Chapter 4, pp. 50-61.
 15. [Bui et al. 2001]
Bui, H. H., Venkatesh, S., West, G. A. W. (2001). Tracking and Surveillance in Wide-Area Spatial Environments Using the Abstract Hidden Markov Model. *IJPRAI*, **15(1)**, pp. 177-195.

16. [Burns and Wellings 1994]
Burns, A. and Wellings, A.J. (1994). HRT-HOOD: A Structured Design Method for Hard Real-Time Systems. *Real-Time Systems Journal*, **6**(1), pp.73-114.
17. [Cameron 1986]
Cameron, J.R (1986). An overview of JSD. *IEEE Transactions on Software Engineering*, **12**(2), pp.222-240.
18. [Carnegie Mellon Software Engineering Institute 2005]
Carnegie Mellon Software Engineering Institute (last modification 2005). *Middleware Software Technology Roadmap*. [Online]. Available at <<http://www.sei.cmu.edu/str/descriptions/middleware.html>>. [accessed 1st March 2006].
19. [Christensen and Alblas 2000]
Christensen, M. and R. Alblas, R. (2000). *V²- Design issues in distributed video surveillance systems*. Demark, 2000, pp. 1-86 [Online]. Available at <<http://www.cs.auc.dk/education/archive/1999/dat5-6/speciale-seminar.html>>. [Accessed 7th March].
20. [Clark 2000]
Clark, I.G. (2000). *A unified approach to the study of asynchronous communication mechanisms in real time systems*. Ph.D. Thesis, University, King's College, London.
21. [COHERENT 2005]
EPSRC Grant JOINT FINAL REPORT(2005). *Computational HEteRogeneously timed NeTworks(COHERENT)*. [Online]. Available at <<http://www.staff.ncl.ac.uk/i.g.clarck/async/coherent/Coherent-final-report.pdf>>. [accessed 1st March 2006].
22. [Collins et al. 2000a]
Collins, R.T., Lipton, A.J., Kanade, T., Fujiyoshi, H., Duggins, D., Tsin, Y., Tolliver, D., Enomoto, N., Hasegawa, O., Burt, P., Wixson, L. (2000). *A System for Video Surveillance and Monitoring*. Robotics Institute, Carnegie Mellon University, pp 1-68.
23. [Collins et al. 2001b]

- Collins, R. T., Lipton, A.J., Fujiyoshi, H., Kanade, T. (2001). Algorithms for cooperative Multisensor Surveillance. *Proceedings of the IEEE*, **89(10)**, pp. 1456-1475.
24. [Conti et al. 2002]
Conti, M., Donatiello, L., Furini, M. Design and Analysis of RT-Ring: A Protocol for supporting Real-Time Communications. *IEEE Transactions on Industrial Electronics*, **49(6)**, pp.1214-1226.
25. [CORBA 2005]
Catalog of OMG Specifications (2005). *Specialized CORBA specifications*. [Online]. Available at < http://www.omg.org/technology/documents/spec_catalog.htm >. [accessed 14th March 2006].
26. [Coulouris et al. 2001]
Coulouris, G., Dollimore, J., Kindberg, T. (2001). *Distributed Systems: Concepts and Design*, Third Edition. London, Pearson Education Limited.
27. [Cristian and Fetzer 1999]
Cristian, F. and Fetzer, C. (1999). The Timed Asynchronous Distributed SystemModel. *IEEE Transactions on Parallel and Distributed Systems*, **10(6)**, pp. 643-657.
28. [CROMATICA 1999]
Digital Imaging Research Centre- Vision Systems (1999). CROMATICA [Online]. Available at <<http://dilnxsrv.king.ac.uk/cromatica/>>. [Accessed 11th March 2006].
29. [Decleir et al. 1999]
Decleir, C., Hacid, M.S., Koulourndijan, J. (1999). A database Approach for Modelling and Querying Video Data. In *Proceedings of the 15th International Conference on Data Engineering*, Australia, pp.1-22.
30. [Diehl et al. 2000]
Diehl, S., Hartel, P., Sestoft, P. (2000). Abstract Machines for programming language implementation. *Future Generation Computer Systems*. **16(7)**. 739-751.
31. [Erik Wyke 2000]

- SNART Awards. *Investigation of models for real-time systems: AIDA through UML and ROOM by Erik Wyke*. [Online]. Available at <
http://www.snart.org/previous_prizes.shtml>. [accessed 4th March 2006].
32. [Ferryman et al. 2000]
 Ferryman, J.M., Maybank, S.J., Worrall, A.D. (2000). Visual Surveillance for Moving Vehicles. *International Journal of Computer Vision*, 37(2), Kluwer Academic Publishers, Netherlands, pp. 187-197.
 33. [François and Medioni 2001]
 François, A.R.J. and Medioni, G.G. (2001). A Modular Software Architecture for Real-Time Video Processing. *Lectures Notes in Computer Science, Proceedings of the Second International Workshop on Computer Vision Systems*. 2001. pp 35-49.
 34. [Garcia et al. 2000]
 Garcia, L. M., Grupen, R.A. (2000). Towards a Real-Time Framework for Visual Monitoring Tasks. *Third IEEE International Workshop on Visual Surveillance*, Ireland, pp. 47-56.
 35. [Geradts and Bijhold 2000]
 Geradts, Z. and Bijhold, J. (2000). Forensic Video Investigation. In *Multimedia Video Based Surveillance Systems*, G. L. Foresti, G.L., Mahonen, P., Regazzoni, C.S., (eds.). Kluwer Academic Publishers, Boston, pp.4-12.
 36. [Ghosh 2001]
 Ghosh, S. (2001). Understanding complex, real-world systems through asynchronous, distributed decision-making algorithms. *The Journal of Systems and Software*. **58(2)**, pp. 153-167.
 37. [Gomaa 1984a]
 Gomaa, H. (1984). A software Design Method for RT systems. *Communications of the ACM*, **27(9)**, 938-949.
 38. [Gomaa 1989b]
 Gomaa, H. (1989). Structuring Criteria for Real Time System Design. *Proc 11th International Conference on Software Engineering, ACM*, Pittsburgh, pp. 290-301.
 39. [Gomaa 1993c]

- Gomaa, H. (1993). *Software Design Methods for Concurrent and Real-Time systems*. USA. Addison & Wesley Professional.
40. [Gong and Xiang 2003]
S. Gong, T. Xiang, T. (2003). Recognition of Group Activities using Dynamic Probabilistic Networks. *Ninth IEEE International Conference on Computer Vision*, volume 2, France, pp.742-750.
41. [Graham 1994]
Graham, I. (1994). *Object Oriented methods*, 2nd ed. Harlow. Addison & Wesley.
42. [Haritaoglu et al. 2000]
Haritaoglu, I., Harwood, D., Davis, L.S. (2000). W⁴:Real-Time Surveillance of People and their activities. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **22(8)**, pp. 809-830.
43. [Haveman 1997]
Haveman, J. (1997). Transaction Decomposition: Refinement of Timing Constraints. In *Proceedings of the South Pacific Conference on Formal Methods*. [Online]. Available at <<http://citeseer.ist.psu.edu/haveman97transaction.html>>. [Accessed 19th July 2006].
44. [Hauck 1995]
Hauck, S. (1995). Asynchronous Design Methodologies: An overview. *Proceedings of IEEE*, **83(1)**, pp. 69-93.
45. [Heikkila and Silven 1999]
Heikkila, J. and Silven, O. (1999). A Real-Time System for Monitoring of Cyclists and Pedestrians. In *Second IEEE International Workshop on Visual Surveillance*, Colorado, 1999, pp. 74-81.
46. [Hemayed 2003]
Hemayed, E.E. (2003). A survey of self-camera calibration. In *Proceedings of the IEEE Conference on Advanced Video and Signal Based Surveillance*, Florida, pp 351-358.
47. [Henning and Vinoski 1999]
Henning, M. and Vinoski, S., (1999). *Advanced CORBA Programming with C++*. USA. Addison-Wesley.

48. [HOOD 1986a]
CISI Ingénieré, CRI A/S, Matra Space, Mitchell R.J. (ed.) (1988). *The Manual for the HOOD Design Methodology: HOOD Manual*. Netherlands. European Space Agency.
49. [HOOD 1989b]
Vielcanet, P. CISI Ingénieré (ed.) (1989). HOOD design method and control/command techniques for the development of real time software. *Proceedings of the sixth Washington Ada symposium on Ada*. pp. 213-219.
50. [HOOD 2004c]
HOOD: An Industrial Approach for Software Design (2004).Rosen, J-P. *Overview of HOOD chapter*. [Online]. Available at < <http://www.adalog.fr/hoodbook.htm>>. [Accessed 7th March 2006].
51. [Hull et al. 2004]
Hull, M.E.C., Ewart, S., Millar, R.J., Hanna, J.R.P. (2004). Modeling Complex Real- Time and Embedded Systems- The UML and DORIS combination. *Real Time Systems*, **26**, 135-159.
52. [ICSE 2004]
Invited talk (2004). In *Proceedings 26th International Conference on Software Engineering* (2004). Edinburgh International Conference Centre, Scotland, UK.
53. [IECCA and MUF 1978a]
Joint IECCA and MUF Committee (1978). *The Official Handbook of Mascot*.UK.
54. [IECCA and MUF 1983b]
Joint IECCA and MUF Committee (1983). *The Official Handbook of Mascot II*, Issue 1.UK.
55. [IECCA and MUF 1987c]
Joint IECCA and MUF Committee (1987). *The Official Handbook of Mascot*. version 3.1, Issue 1.UK.
56. [Ivanov et al. 1999]
Ivanov, Y., Stauffer, C., Bobick, A., Grimson, W.E.L. (1999). Video Surveillance of Interactions. In *Second IEEE International Workshop on Visual Surveillance*, Colorado, pp. 82-91.

57. [Ivanov and Bobick 2000]
Ivanov, Y. and Bobick, A. (2000). Recognition of visual activities and interaction by stochastic parsing. *IEEE Transactions of Pattern Recognition and Machine Intelligence*. **22(8)**, pp.852-872.
58. [Jackson 1983a]
Jackson, M. A. (1983). *System Development*. London. Prentice-Hall International.
59. [Jackson 1994b]
Jackson, M. (1994). Jackson Development Methods: JSP and JSD. *Encyclopedia of Software Engineering*. Vol I, John Wiley & Sons. pp 585-593.
60. [Jackson and Rouskas 2002]
Jackson, L.E. and Rouskas, G.N. (2002). Deterministic Preemptive Scheduling of Real-Time Tasks. *Computer, IEEE*, **35(5)**, pp. 72-79.
61. [JavaBeans 2006]
Sun Developer Network (SDN) (2006). *Desktop Java, JavaBeans*. [Online]. Available at <<http://java.sun.com/products/javabeans>>. [Accessed 19th July 2006].
62. [Jaynes 1999]
C. Jaynes, C. (1999). Multi-view Calibration from Planar Motion for Video Surveillance. *Second IEEE International Workshop on Visual Surveillance*, Colorado, 1999, pp. 59-67.
63. [Jian-Guang et al. 2003]
Jian-Guang, L., Qi-Feing, L., Tie-Niu, T., Wei-Ming, H. (2003). 3-D Model Based Visual Traffic Surveillance. *Acta Automatica Sinica*, **29(3)**, pp. 434-449.
64. [Kalinsky and Barr 2002]
Kalinsky, D. and Barr, M. (2002). Priority Inversion. *Embedded Systems Programming*. April, pp. 55-56.
65. [Krumm et al. 2000]
Krumm, J., Harris, S., Meyers, B., Brumit, B., Hale, M., Shafer, S. (2000). Multi-Camera Multi-Person Tracking for Easy Living. *Third IEEE International Workshop on Visual Surveillance*, Ireland, pp. 8-11.
66. [London Underground Ltd (n.d.)]

- London Underground Limited (n.d.). *Standard for CCTV systems*. S&CSE-ST0015-A2 London, Signal and Control Systems engineering.
67. [Macmillan Technical Publishing and Cisco Systems 1998]
Macmillan Technical Publishing and Cisco Systems. (1998). *Cisco CCIE Fundamentals: Network Design and Case Studies*. USA. Cisco Press.
68. [Mähönen and Saaranen 2000]
Mähönen, P. and Saaranen, M. (2000). Broadband Multimedia Transmission for Surveillance Applications. In *Multimedia Video Based Surveillance Systems*, G. L. Foresti, G.L., Mahonen, P., Regazzoni, C.S., (eds.). Kluwer Academic Publishers, Boston, pp. 173-185.
69. [Makris et al. 2004]
Makris, D., Ellis, T., Black, J. (2004). Bridging the gaps between cameras. In *International Conference on Multimedia and Expo*, Taiwan, pp. 205-210.
70. [Marchesotti et al. 2003]
Marchesotti, L., Messina, A., Marcenaro, L., Regazzoni, C. S. (2003). A cooperative Multisensor System for Face Detection in Video Surveillance Applications. *Acta Automatica Sinica*, **29(3)**, pp. 423-433.
71. [Marcenaro et al. 2001]
Marcenaro, L., Oberti, F., Foresti, G.L., Regazzoni, C.S. (2001). Distributed architectures and logical-task decomposition in Multimedia surveillance systems. *Proceedings of the IEEE*, **89(10)**, pp. 1419-1438.
72. [Marsden and Fabre 2001]
Marsden, E. and Fabre, J.C. (2001). *Failure analysis of an ORB in presence of faults. Dependable Systems of Systems (DSoS)*. IST-1999-11585. Project founded by the European Community under the Information Society Technology.
73. [Medvidovic and Taylor 2000]
Medvidovic, N. and Taylor, R.N. (2000). A Classification and Comparison Framework for Software Architecture Description Languages. *Software Engineering*, **26(1)**, 70-93.
74. [Micheloni et al. 2003]

- Micheloni, C., Foresti, G.L., Snidaro, L. (2003). A co-operative multi-camera system for video-surveillance of parking lots. *Intelligent Distributed Surveillance Systems Symposium by the IEE*, London, pp. 21-24.
75. [Microsoft .Net 2006]
.NET: Driving Business Value with the Microsoft Platform (2006). *What is .NET?*. [Online]. Available at <<http://www.microsoft.com/net/default.mspx>>. [Accessed 19th July 2006].
76. [Middleware 2005]
IEEE distributed systems online (2005). *Middleware areas*. [Online]. Available at <<http://dsonline.computer.org/middleware>>. [Accessed 12th March 2006].
77. [Mowbray and Zahavi 1995]
Mowbray, J.T. and Zahavi, R. (1995). *The Essential CORBA, System Integration Using Distributed Objects*. John Wiley & Sons, Inc.
78. [MPI 2003a]
SP Parallel Programing Workshop (2003). *Message Passing Interface (MPI)*. [Online]. Available at <<http://www.mhpc.edu/training/workshop/mipi/MAIN.html>>. [Accessed 19th July 2006].
79. [MPI 2006b]
Cornell University, Cornell Theory Center (2006). *Basis of MPI Programming*. [Online]. Available at <<http://www.tc.cornell.edu/Services/Education/Topics/MPI/Basics/Introduction.html>>
80. [Mullery 1979]
Mullery, G.P. (1979). CORE-A method for Controlled Requirement Specification. In *Proceedings of the 4th International Conference on Software Engineering, IEEE*, Munich, September, [n.d.], pp. 126-135.
81. [Muñoz 2002]
Muñoz, R. (2002). *Using LOTOS for the Analysis of MASCOT*. Ph.D. Thesis. King's College London, UK.
82. [Mustafa 2000]
Mustafa, A.J. (2000). *Petri Nets approach for the analysis of MASCOT interprocess communications*. Ph.D. Thesis. King's College London. London.

83. [Naedele 2001]
Naedele, M. (2001). An approach to modeling and evaluation of functional and timing specification of real-time systems. *The Journal of Systems and Software*, **57(2)**, pp. 155-174.
84. [Ng et al. 1999]
K.C. Ng, K.C., Ishiguro, H., Trivedi, M., Sogo, T. (1999). Monitoring Dynamically Changing Environments by Ubiquitous Vision System. *Second IEEE Workshop on Visual Surveillance*, pp. 67-74.
85. [Nguyen et al. 2003a]
Nguyen, N.T., Venkatesh, S., West, G., Bui, H.H. (2003). Multiple Camera Coordination in a Surveillance System. *Acta Automatica Sinica*, **29(3)**, pp. 408-421.
86. [Nguyen et al. 2003b]
Nguyen, N.T., Bui, H.H., Venkatesh, S., West, G. (2003). Recognising and Monitoring High-Level Behaviour in Complex Spatial Environments. *IEEE International Conference on Computer Vision and Pattern Recognition*, Wisconsin, pp.1-6.
87. [Norhashimah et al. 2003]
Norhashimah, P., Fang, H., Jiang, J. Video Extraction in Compressed Domain. *IEEE Conference on Advanced Video and Signal Based Surveillance*, Florida, pp. 321-327.
88. [Nwagboso 1998]
Nwagboso, C. (1998). User focused Surveillance Systems Integration for Intelligent Transport Systems. In *Advanced Video-based Surveillance Systems*. C.S. Regazzoni, G. Fabri, G. Vernazza (eds.), Kluwer Academic Publishers, Boston, Chapter 1.1, 1998, pp. 8-12.
89. [Oates et al. 2000]
Oates, T., Schmill, M.D., Cohen, P.R. (2000). A method for clustering the experiences of a mobile robot with human judgements. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, AAAI Press, pp. 846-851.

90. [O'Donoghue and Hull 1996]
O'Donoghue, P.G. and Hull, M.E.C. (1996). Using Timed CSP during object oriented design of real-time systems. *Information and Software Technology*, **38(2)**, pp. 89-102.
91. [Oren et al. 1997]
Oren, M., Papageorgiou, C., Sinham, P., Osuna, E., Poggio, T. (1997). Pedestrian detection using wavelet templates. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, Puerto Rico, pp.193-199.
92. [Paquet and Teare 2001]
Paquet, C. and Teare, D. (2001). *Building Scalable Cisco Networks*. USA, Cisco Press.
93. [Parnas 1972]
Parnas, D.L, Morris, R. (ed.) (1972). On the Criteria To Be Used in Decomposing Systems into Modules. *Communications of the ACM*. **15(12)**. 1053-1058.
94. [Pavlidis et al. 2001]
Pavlidis, I., Morellas, V., Tsiamyrtzis, P., Harp, S. (2001). Urban Surveillance Systems: From the laboratory to the commercial world. *Proceedings of the IEEE*, **89(10)**, pp. 1478-1495.
95. [Paulidis and Morellas 2002]
Paulidis, I. and Morellas, V. (2002). Two examples of Indoor and Outdoor Surveillance Systems. In *Video-based Surveillance Systems*, Remagnino, P., Jones, G.A., Paragios, N., Regazzoni, C.S. (eds.). Kluwer Academic Publishers, Boston, pp. 39-51.
96. [Paynter 2000]
Paynter, S., Armstrong, J., Haveman, J., (2000). ADL: The Activity Description Language for Real-Time Networks. *Formal Aspects of Computing*. 12(2), 120-144.
97. [Pellegrini and Tonani 1998]
Pellegrini, M. and P. Tonani, P. (1998). Highway traffic monitoring. In *Advanced Video-based Surveillance Systems*. Regazzoni, C.S., Fabri, G., Vernazza, G., (eds.). Kluwer Academic Publishers, Boston, pp. 27-33.
98. [Peters and Pedrycz 2000]

- Peters, J.F., Pedrycz, W. (2000). *Software Engineering: An Engineering Approach*. New York, John & Sons.
99. [Ping Lai Lo et al. 2003]
Ping Lai Lo, B., Sun, J., Velastin, S.A. (2003). Fusing Visual and Audio Information in a Distributed Intelligent Surveillance System for Public transport Systems. *Acta Automatica Sinica*, **29(3)**, pp 393-407.
100. [Phillip 1996]
Phillip, A. L. (1996). *Real Time Systems Design and Analysis an Engineer's Handbook*. 2nd Edition IEEE PRESS. John Wiley & Sons.
101. [Phillips 1967]
Phillips, C.S.E. (1967). Networks for Real-Time Programming. *Computer Journal*. **10(1)**. 46-52.
102. [Pozzobon et al. 1998]
Pozzobon, A., Sciutto, G., Recagno, V. (1998). Security in Ports: the User Requirements for Surveillance System. In *Advanced Video-based Surveillance Systems*, Regazzoni, C.S., Fabri, G., Vernazza, G. (eds.). Kluwer Academic Publishers, Boston, pp 18-26.
103. [Rath and Manmatha 2003]
Rath, T. M. and Manmatha, R. (2003). Features for word spotting in historical manuscripts. In *Proceedings of the Seventh International Conference on Document Analysis and Recognition*, pp.512-527.
104. [Regazzoni et al. 2001]
Regazzoni, C. S., Ramesh, V., Foresti, G. L. (2001). Special Issue on Video Communications, Processing, and Understanding for Third Generation Surveillance Systems. *Proceedings of the IEEE*. **89(10)**, pp.1355-1365.
105. [Remagnino et al. 1997]
Remagnino, P., Baumberg, A., Grove, T., Hogg, D., Tan, T., Worral, A., Baker, K. (1997). An Integrated Traffic and Pedestrian Model-Based Vision System. In *BMVC97 Proceedings*, Israel, pp. 380-389.
106. [Ronetti and Dambra 2000]

- Ronetti, N. and Dambra, C. (2000). Railway Station Surveillance: The Italian Case. In *Multimedia Video Based Surveillance Systems*, G. L. Foresti, G.L., Mahonen, P., Regazzoni, C.S., (eds.). Kluwer Academic Publishers, Boston, pp. 13-20.
107. [Rumbaugh et al. 1991]
Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W. (1991). *Object-Oriented Modeling and Design*, Englewood Cliffs, Prentice Hall.
108. [Rybski et al. 2002]
Rybski, P. E., Stoeter, S.A., Gini, M., Hougen, D.F., Papanikolopoulos, N.P. (2002). Performance of a Distributed Robotic System Using Shared Communications Channels. *IEEE Transactions on Robotics and Automation*, **18(5)**, pp.713-727.
109. [Saad and Smith 2003]
Saad, A. and Smith, D. (2003). An IEEE 1394-Firewire-based embedded video system for surveillance applications. *IEEE Conference on Advanced Video and Signal Based Surveillance*, Florida, pp. 213-219.
110. [Seidewitz and Stark 1986]
Seidewitz, E. and Stark, M. (1986). *Towards a General Object Oriented Software Development Methodology*. MD 20771. NASA Goddard Space Flight Center. Greenbelt.
111. [Simpson 1986a]
Simpson, H.R. (1986). The MASCOT method. *Software Engineering Journal*, **1(3)**, pp. 103-120.
112. [Simpson 1990b]
Simpson, H.R. (1990). *A Data Interaction Architecture for Real Time Embedded Multi Processor Systems*. Proceedings of a RAeS Conference on Computing Techniques in Guided Flight. Royal Aeronautical Society Conference. Boscombe Down, pp. 1-10.
113. [Simpson 1990c]
Simpson, H.R. (1990). Four-slot fully asynchronous communication mechanism. *IEE Proceedings on Computers and Digital Techniques*.**137(1)**. 17-30.
114. [Simpson 1992d]

- Simpson, H.R. (1992). *Real Time Networks in Configurable Distributed Systems*. Processing of the IEE International Workshop on Configurable Distributed Systems. Imperial College London, UK, pp 1-14.
115. [Simpson 1994e]
Simpson, H.R. (1994). Architecture for Computer Based Systems. *IEEE Computer Society Press Reprint*. Los Alamitos. USA, pp. 70-82.
116. [Simpson 2003f]
Simpson, H.R. (2003). Protocols for process interaction. *IEE Proceedings on Computers and Digital Techniques*. **150(3)**. 157-182.
117. [Stauffer et al. 2000]
Stauffer, C., Eric, W., Grimson, L. (2000). Learning Patterns of Activity using Real-Time tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **22(8)**, pp. 747-757.
118. [Stankovic and Ramamritham 1990]
Stankovic, J.A. and Ramamritham, K. (1990). What is Predictability for Real-Time Systems?. *Real-Time Systems*, **2(4)**, pp. 247-254.
119. [Stringa and Regazzoni 1998]
Stringa, E. and Regazzoni, C.S. (1998). Content-based Retrieval and real-Time Detection from video sequences acquired by surveillance systems. *International Conference on Image Processing*, Chicago, pp. 138-142.
120. [Summer School 2004]
Summer School in Computer Vision (2004). Software tools for Computer Vision. *Summer School in Computer Vision in Surrey University*, UK.
121. [Thonnat and Rota 2000]
Thonnat, M. and Rota, N. (2000). Video sequence Interpretation for Visual Surveillance. *Third IEEE International Workshop on Visual Surveillance*, Dublin, pp.59-68.
122. [Tierney et al. 2000]
Tierney B., Johnston W., Lee J., Thompson M. (2000). A Data Intensive Distributed Computing Architecture for “Grid” applications. *Future Generation System*, **16(5)**,473-481.

123. [Turner 1993]
Turner, K.J. (1993). *Using formal description techniques: an introduction to ESTELLE, LOTOS and SDL*, John Wiley & Sons Ltd.
124. [UML 2003]
Fowler, M. (2003). *UML Distilled Third Edition, A Brief Guide To The Standard Object Modeling Language*. Pearson Education, Inc.
125. [UML 2 Metamodel 2005]
Catalogue of OMG Modelling And Metadata Specifications (2005). *UML 2 Metamodel*. [Online]. Available at < <http://www.omg.org/cgi-bin/doc?formal/05-04-01> >. [accessed 6th March 2006].
126. [UML 2006]
UML TM Resource Page (2006). *OMG's List Of UML 2.0 Tools*. [Online]. Available at <<http://www.uml.org>>. [accessed 6th March 2006].
127. [Valera and Velastin 2003a]
Valera, M. and Velastin, S A. (2003). An approach for designing a real-time intelligent distributed surveillance system. *In Proceedings of the IEE Workshop on Intelligent Distributed Surveillance Systems*, London, pp. 42-48.
128. [Valera and Velastin 2005b]
Valera M. and Velastin, S.A. (2005). Intelligent distributed surveillance systems: a review. *IEE Proceedings of Visual Image Signal Process*, **152(2)**, pp. 192-204.
129. [Velastin 2003]
Velastin, S A. (2003). Getting the Best Use out of CCTV in the Railways. London, Rail Safety and Standards Board, pp 1-17.
130. [Wikipedia 2001]
Wikipedia (2001). *System Engineering*. [Online]. Available at < http://en.wikipedia.org/wiki/Systems_engineering > [accessed 2nd March 2006].
131. [Wren et al. 1997]
Wren, C., Azarbayejani, A., Darrell, T., Pentland, A. (1997). Pfinder:Real-Time Tracking of the Human Body. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **19(7)**, pp. 780-785.
132. [Wu et al. 2001]

- Wu, C.H., Irwin, J.D., Dai, F. F. (2001). Enabling Multimedia Applications for Factory Automation. *IEEE Transactions on Industrial Electronics*, **48(5)**, pp.913-919.
133. [Xia 2000]
Xia, F. (2000). *Concurrent Real Time Systems with such techniques as MASCOT and Petri nets*. Ph.D. Thesis. King's College, London.
134. [Xu et al. 2004]
Xu, M., Lowey, L., Orwell, J. (2004) .Architecture and algorithms for tracking football players with multiple cameras. In *Proceedings of the IEE Workshop on Intelligent Distributed Surveillance Systems*, London, 2004, pp. 51-56.
135. [Ye et al. 2001]
Ye, H., Walsh, G.C., Bushnell, L.G. (2001). Real-Time Mixed-Traffic Wireless Networks. *IEEE Transactions on Industrial Electronics*, **48(5)**, pp.883-890.
136. [Yourdon and Constantine 1979]
Yourdon, E. and Constantine, L.L., (1979). *Structured design*. Prentice-Hall.
137. [Yuancai 2002]
Yuancai, Ye C.(2002), *Use of free SocketPro package for creating super client and server applications*. [Online]. Available at < <http://www.codeproject.com/internet/yesocketpro.asp>> [accessed 1st March 2006].
138. [Yuan et al. 2003]
Yuan, X., Sun, Z., Varol, Y., Bebis, G. (2003). A Distributed Visual Surveillance System. *IEEE Conference on Advanced Video and Signal Based Surveillance*, Florida, pp.199-205.
139. [Yun et al. 1993]
Yun, K.Y., Dill, D. L., Nowick, S. M. (1993). Practical generalizations of asynchronous state machines. *The European Conference on Design Automation with The European Event in ASIC Design*. pp. 525-582.
140. [Zhi-Hong 2003]
Zhi-Hong, Z. (2003). Lane Detection and Car Tracking on the Highway. *Acta Automatica Sinica*, **29(3)**, pp. 450-456.

