4

An Implementation of High Availability in Networked Robotic Systems

Florin Daniel Anton, Theodor Borangiu and Silvia Anton
University Politehnica of Bucharest
Romania

1. Introduction

Nowadays production flows are modular, each module in the enterprise being specialized and used to achieve a particular task. In many cases the modules are interconnected and materials are sequentially processed in each module resulting a final, unique product or assembly. One typical such production module is a flexible cell/system using multiple robots. In such complex enterprise environments, providing continuous services for applications is a key component of a successful implementing of robotized manufacturing. *High availability* (HA) is one of the components contributing to continuous service provision for applications, by masking or eliminating both planned and unplanned downtime of systems and applications. This is achieved by eliminating hardware and software *single points of failure* (SPOF).
The systems configured for high availability are a combination of hardware and software components configured to work together to ensure automated recovery in case of failure with a minimal acceptable downtime.
A high availability solution will ensure that the failure of any component of the solution - either hardware, software or system management, will not cause the application and its data to become permanently unavailable. High availability solutions should eliminate single points of failure through appropriate design, planning, hardware selection, software configuring, application control, carefully environment control and change management discipline.
In short, one can define high availability as the process of ensuring an application is available for use by duplicating and/or sharing hardware resources managed by a specialized software component. A high availability solution in robotized manufacturing provides automated failure detection, diagnosis, application recovery, and node (robot controller) re integration.
The chapter discusses the implementing of a high availability solution in a robotized manufacturing structure (cell, line). The solution is based on a High Availability Linux Cluster which is responsible for data availability and preservation on a NFS (Network File System) file system, and a second cluster – Fabrication Cluster (FC) which runs on Adept robot controllers, developed under the V+ programming language.

2. High availability versus fault tolerance

Based on the response time and response action to system detected failures, clusters and systems can be generally classified as:

Source: Advances in Robotics, Automation and Control, Book edited by: Jesús Arámburo and Antonio Ramírez Treviño,
ISBN 78-953-7619-16-9, pp. 472, October 2008, I-Tech, Vienna, Austria

- Fault-tolerant
- High availability

2.1 Fault-tolerant systems

The systems provided with *fault tolerance* are designed to operate virtually without interruption, regardless of the failure that may occur (except perhaps for a complete site going down due to a natural disaster). In such systems all components are at least duplicated for both software and hardware.
This means that all components, CPUs, memory, Ethernet cards, serial lines and disks have a special design and provide continuous service, even if one sub-component fails. Only special software solutions will run on fault tolerant hardware.
Such systems are very expensive and extremely specialized. Implementing a fault tolerant solution requires a lot of effort and a high degree of customization for all system components.
For environments where *no* downtime is acceptable (life critical systems), fault-tolerant equipment and solutions are required.

2.2 High availability systems

The systems configured for *high availability* are a combination of hardware and software components configured to work together to ensure automated recovery in case of failure with a minimal acceptable downtime.
In such industrial systems, the software involved detects problems in the robotized environment (production line, flexible manufacturing cell), and manages application survivability by restarting it on the same or on another available robot controller.
Thus, it is very important to eliminate all single points of failure in the manufacturing environment. For example, if a robot controller has only one network interface (connection), a second network interface (connection) should be provided in the same node to take over in case the primary interface providing the service fails.
Another important issue is to protect the data by mirroring and placing it on shared disk areas accessible from any machine in the cluster, directly or using the local area network.

3. High availability terms and concepts

For the purpose of designing and implementing a high-availability solution for networked robotic stations integrated in a manufacturing environment, the following terminology and concepts are introduced:
RMC: The Resource Monitoring and Control (RMC) is a function giving one the ability to monitor the state of system resources and respond when predefined thresholds are crossed, so that many routine tasks can be automatically performed.
Cluster: Loosely-coupled collection of independent systems (nodes – in this case robot controllers) organized in a network for the purpose of sharing resources and communicating with each other. A cluster defines relationships among cooperating systems, where peer cluster nodes provide the services offered by a cluster node should that node be unable to do so. These individual nodes are together responsible for maintaining the functionality of one or more applications in case of a failure of any cluster component.
There are two types of high availability clusters:

- Peer domain
- Managed domain

The general difference between these types of clusters is the relationship between the nodes.

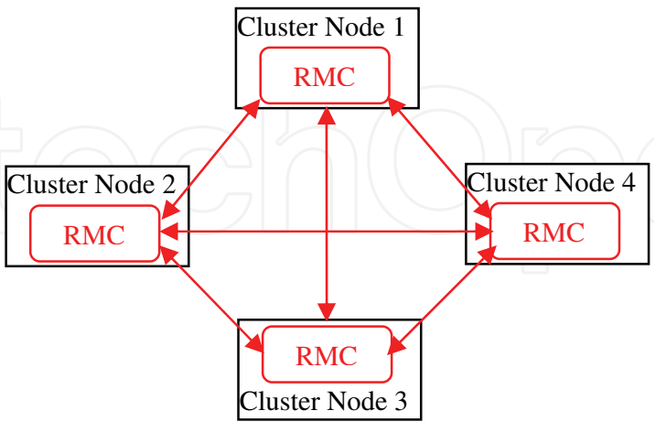

Fig. 1. Peer domain cluster topology.

In a *peer domain* (Fig. 1.), all nodes are considered equal and any node can monitor and control (or be monitored and controlled) by any other node (Harris et. al., 2004).
In a *management domain* (Fig. 2), a management node is aware of all nodes it is managing and all managed nodes are aware of their management server, but the nodes themselves know nothing about each other.

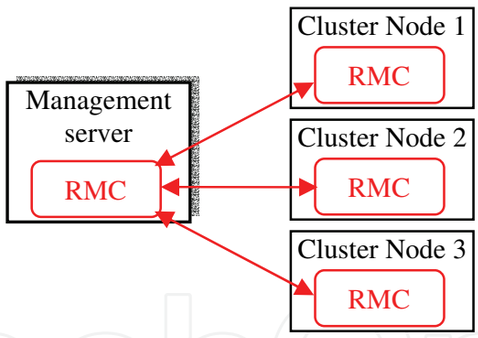

Fig. 2. Managed domain cluster topology.

Node: A robot controller that is defined as part of a cluster. Each node has a collection of resources (disks, file systems, IP addresses, and applications) that can be transferred to another node in the cluster in case the node or a component fails.
Clients: A client is a system that can access the application running on the cluster nodes over a local area network. Clients run a client application that connects to the server (node) where the application runs.
Topology: Contains basic cluster components nodes, networks, communication interfaces, communication devices, and communication adapters.
Resources: Logical components or entities that are being made highly available (for example, file systems, raw devices, service IP labels, and applications) by being moved from one node to another. All the resources that together form a highly available application or service are grouped in one resource group (RG). The cluster keeps the RG highly available

as a single entity that can be moved from node to node in the event of a component or node failure. Resource groups can be available from a single node or, in the case of concurrent applications, available simultaneously from multiple nodes. A cluster may host more than one resource group, thus allowing for efficient use of the cluster nodes.

Service IP label: A label matching to a service IP address and which is used for communications between clients and the node. A service IP label is part of a resource group, which means that the cluster will monitor it and keep it highly available.

IP address takeover: The process whereby an IP address is moved from one adapter to another adapter on the same logical network.

Resource takeover: This is the operation of transferring resources between nodes inside the cluster. If one component or node fails due to a hardware or operating system problem, its resource groups will be moved to the another node.

Failover: Represents the movement of a resource group from one active node to another node (backup node) in response to a failure on that active node.

Fallback: Represents the movement of a resource group back from the backup node to the previous node, when it becomes available. This movement is typically in response to the reintegration of the previously failed node.

Heartbeat packet: A packet sent between communication interfaces in the cluster, used to monitor the state of the cluster components - nodes, networks, adapters.

RSCT processes: (Reliable Scalable Cluster Technology). They consist of two processes (topology and group services) that monitor the state of the cluster and each node. The cluster manager receives event information generated by these processes and takes corresponding (response) actions in case of failure(s).

Group Leader (GL): The node with the highest IP as defined in one of the cluster networks (the first network available), that acts as the central repository for all topology and group data coming from the RSCT daemons concerning the state of the cluster.

Group leader backup: This is the node with the next highest IP address on the same arbitrarily chosen network, acting as a backup for the Group Leader; it will take over in the event that the Group Leader leaves the cluster.

Mayor: A node chosen by the RSCT Group Leader (the node with the next highest IP address after the GL Backup), if such exists; otherwise the Mayor node is the GL Backup itself. It is the Mayor's responsibility to inform other nodes of any changes in the cluster as determined by the Group Leader (GL).

Quorum: The notion of quorum is used to ensure that in case of loss of connectivity between two subsets of the peer domain only one subset considers itself as the peer domain. The quorum is defined as $n/2+1$, where n is the number of nodes defined in the peer domain. A peer domain cannot be turned online if less than the quorum of nodes can communicate.

SPOF: A single point of failure (SPOF) is any individual component integrated in a cluster which, in case of failure, renders the application unavailable for end users. Good design will remove single points of failure in the cluster - nodes, storage, networks. The implementation described here manages such single points of failure, as well as the resources required by the application.

4. RMC architecture and components design

The most important unit of a high availability cluster is the Resource Monitoring and Control (RMC) function, which monitors resources (selected by the user in accordance with the application) and performs actions in response to a defined condition.

The design of RMC architecture is presented for a multiple-resource production control system. The set of resources is represented by the command, control, communication, and operational components of networked robot controllers (station controllers) and robot terminals (station computers) integrated in the manufacturing cell.
The RMC subsystem to be defined is a generic cluster component that provides a scalable and reliable backbone to its clients with an interface to resources.

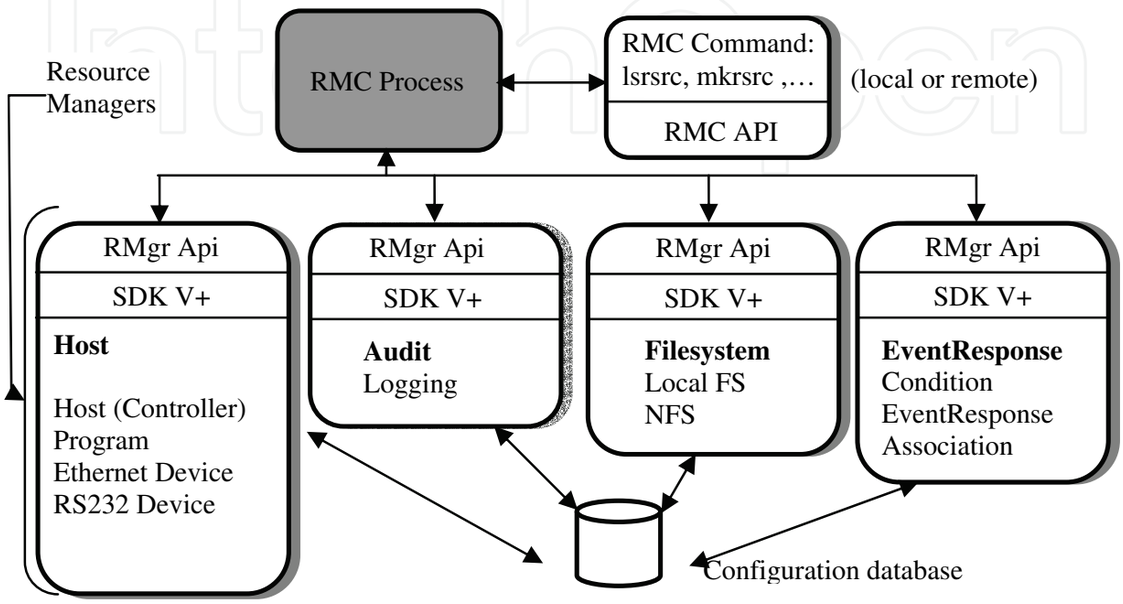

Fig. 3. The structure of the RMC subsystem.

The RMC has no knowledge of resource implementation, characteristics or features. The RMC subsystem therefore delegates to resource managers the actual execution of the actions the clients ask to perform. The architecture allows the following functions to be supported on clusters:

- Provide single monitoring and management infrastructure for clusters.
- Provide global access to subsystems and resources throughout the cluster.
- Support operations for configuring, monitoring, and controlling all cluster resources by RMC clients.
- Encapsulate all resource dependent operations.
- Provide a common access control mechanism across all resources.
- Support integration with other subsystems to achieve the highest levels of availability.

To support these functions, the structure of the RSCT Resource Monitoring and Control is not simple (see Fig. 3).
The RMC subsystem and RMC clients need not be in the same node; RMC provides a distributed service to its clients. The RMC clients can connect to the RMC process either locally or remotely using the RMC API i.e. Resource Monitoring and Control Application user Interface (Matsubara *et al*., 2002).
Similarly, the RMC subsystem interacting with Resource Managers need not be in the same node. If they are on different nodes, the RMC subsystem will interact with local RMC subsystems located on the same node as the resource managers; then the local RMC process will forward the request between them.

Each resource manager is instantiated as one process. To avoid the multiplication of processes, a resource manager can handle several resource classes.
The commands of the Command Line Interface are V+ programs (V+ is the robot programming environment); the end-user can check and use them as samples for writing his own commands.

4.1 The Command Line Interface (CLI)

A RMC command line client can access all the resources within a cluster locally (A) and remotely (B) located (Fig. 4). The RMC command line interface (Fig. 5) is comprised of more than 50 commands (V+ programs): some components, such as the *Audit* resource manager, have only two commands, while others, such as *Event Response* resource manager, have 15 commands.

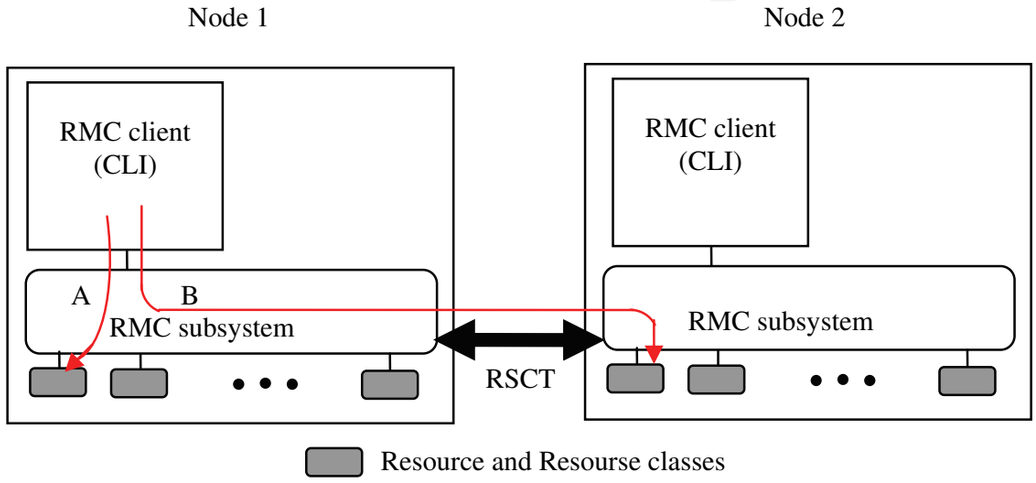

Fig. 4. The relationship between RMC Clients (CLI) and RMC subsystems.

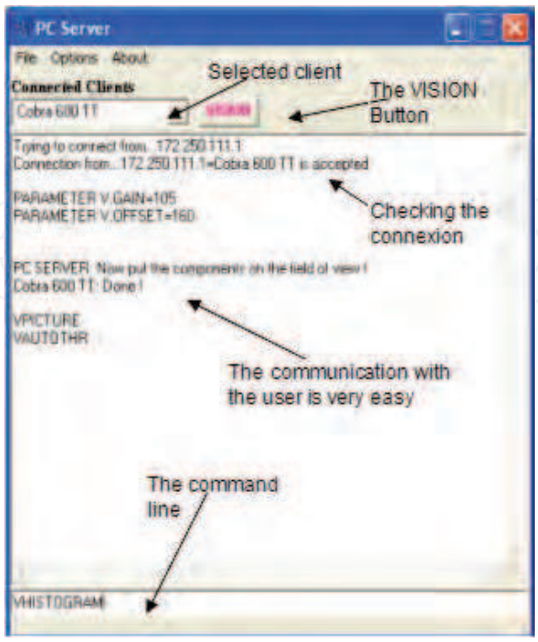

Fig. 5. The Command Line Interface.

Most RMC commands provide the user with a similar perception. Many commands have optional arguments to instruct the realization of the same functions. In addition to optional command arguments, RMC defines two environment variables:

- CT_CONTACT – define the communication target
- CT_MANAGEMENT_SCOPE – define the scope

These variables instruct most RMC commands to define their target and scope upon retrieving and setting values among cluster nodes.

Finally, RMC embeds the concept of *Resource Data Input*, which is a standardized file format for holding resource properties. Many RMC commands have a -f option that allows the end user to pass arguments to the command through a file instead of typing them on the command line. These files have the same Resource Data Input format for all commands so that the same file can be used for different commands. Two commands (**lsrsrcdef –** list resource definitions, and **lsactdef –** list actions definitions) can generate such a file so their output can be used as input for other commands.

4.2 The web-based Graphical User Interface (GUI)

The RMC graphical user interface is provided by the Web-based System Manager, and allow the user to configure the fabrication cluster (a HA cluster composed by industrial robots), and also acts as a teleoperation system.

The system allowing access to the web interface is composed by the following applications (Fig. 6):

The **Server Application (SA)**: Remote visual control and monitoring of multiple robot controllers from mobile and stationary matrix cameras.

- *Visual control*: the Server Application supports almost all V+ and AdeptVision program instructions and monitor commands. Robot training and control are interactive – menu-driven and acknowledged by image display in a VISION window. Some of the main functions available in this window are: choice of the physical camera and virtual cameras (multiple information counterparts of a single physical camera) and of the image buffers; selecting the display mode and resolution; histogram and average curve contrast analysis; selection of switches and parameters for virtual camera construction; display of vision system status; training and planning multiple ObjectFinder models for recognition and locating (Automated Visual Inspection – AVI & Guiding Vision for Robots – GVR); learning fingerprint models for collision-free grasping; editing, saving, loading and running V+ programs.
- *Monitoring*: a Monitoring/Treatment scheme can be defined for each Client/Station (the latter can be selected from a drop-down list of robot controllers connected to the server, by adding/removing them from the Client window). For each client a list of events and controller variables to be monitored according to a user-definable timing and precedence, and reacted at by user-definable actions/sequences can be specified in an Automatic Treatment Window.
- *Communication management*: the Server Application manages the communication with the robot controllers and the observation cameras, transfers real-time images from the cameras observing the robot workplace and production environment, reports status information, stores in a database and displays images taken by the robot camera via its controller. Finally, the SA outputs commands which are received from the eClients or acknowledges task execution.

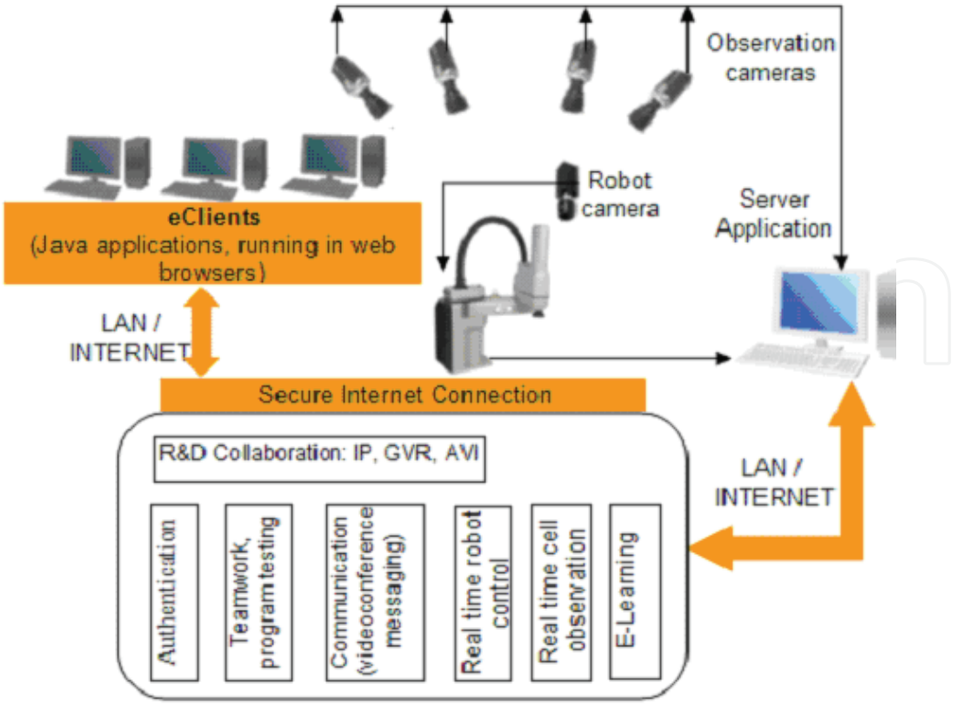

Fig. 6. The structure of the system which provides the web based interface.

The **eClients Applications (eCA)**: Java applications running in web browsers. They provide portal services and the connection of networked production agents: image data and RV program / report management; real-time robot control and cell / workplace observation. The eCA are composed by two applications:

- One application which has the function of retrieving images from the observation cameras (AXIS 214 PTZ), displaying them in real-time and also giving the user the possibility to change the orientation and zoom factor of the cameras.
- The second application is a VNC client.

The VNC viewer (Java client) is a web teleoperation application which can be executed into a web browser. The application connects to the Domino web server which makes a secure connection using a TCP/IP tunnel with a server having a private IP address, which cannot be accessed from internet but only using the Domino server. The private IP machine has a VNC server that exports the display, and also the teleoperation application (see Fig. 7). Using the exported display, the user can view and run the application as if the application runs on his own computer. The access is made using a username and a password, process managed by the Domino server.

For team training and application development, the system allows accessing related documents, presentations and multimedia materials, Web conferencing, instant messaging and teamwork support for efficient and security-oriented creation and use of group workspaces (students / trainers, researchers).

The Server and eClients Applications run on IBM PC workstations on which IBM Lotus software offerings are customized and integrated with other applications (error-free data and command transfer, V+ program editing, real time image display and refresh in multiple Client windows, replication functions, Client authentication, etc) in a virtual training and research laboratory across geographical boundaries.

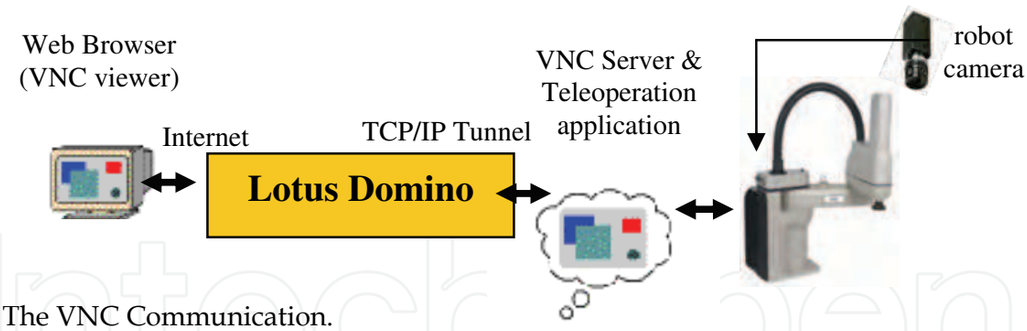

Fig. 7. The VNC Communication.

Lotus-oriented solutions have been considered for transferring messages, status reports, data and video camera images, interpreting them and accessing databases created by all partners. The final objective of the platform is to develop an E-Learning component allowing students to access and download technical documentation, create, test, debug and run RV and AVI programs, attend real-time laboratory demonstrations, and check their skills in proposed exercises.

Thus, IBM Lotus software unifies all Application modules, providing the necessary management and interconnecting tools for distributed industrial controllers, and the collaborative tools with back-end relational databases for team training and research.

To have access to the system, a user must have a username and a valid password to enter in the system. First the user must access the portal site using a java aware browser (like Internet Explorer, Opera, Firefox, with the JRE installed).

The portal which allows the web access is structured in two zones (Fig. 8):

- a public zone which contains all documentation, tutorials courses and others, needed by users to learn how to exploit the system; this part of the portal can be accessed by anyone;
- a private zone where the access is based on username and password. The private zone gives access to the eClients for teleoperation purposes.

After entering the correct username and password, the user is allowed in the system and has access to the teleoperation application which is a menu driven interface which allows him to interact with the system (see Fig. 9).

The web interface is also composed by two zones:

- An observation zone (right side in Fig. 9) where the user can monitor the jobs which are currently performed by the robots by using three networked cameras (due to the high resolution needed, only two cameras can be used at a time);
- A teleoperation zone where the user can interact with each robot.

The teleoperation application is composed by three windows (see Fig. 9):

- A camera control window, where the user can control all the camera parameters (tilt, pitch, zoom, focus and iris opening);
- A command window where the user can select the robot system which he wants to control, issue commands from the command line or activate the third window (Vision);
- A Vision window.

From the Vision window, vision commands can be issued by selecting the desired actions from the menus. The most important available functions are:

- selecting the physical and virtual cameras, and the virtual image buffers;
- selecting the display mode and the resolution;
- image acquisition;
- executing primary operations (histogram, thresholding, etc.);

- displaying the vision system status; training models;
- configuring switches and parameters for virtual camera set up.

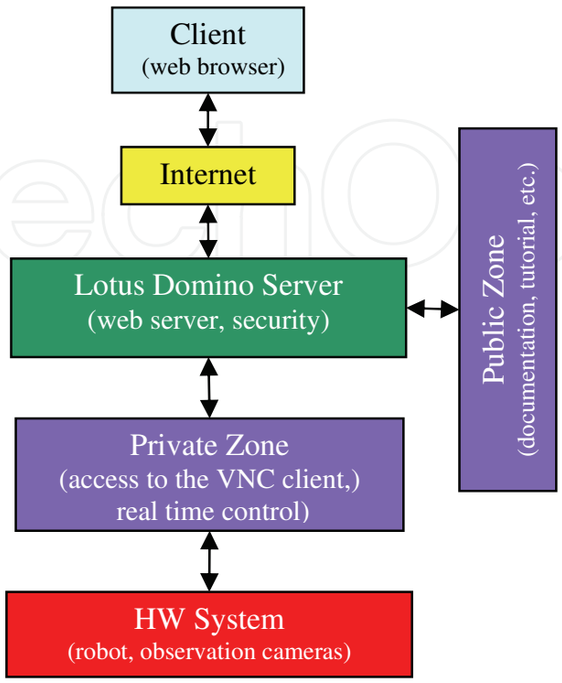

Fig. 8. The portal structure.

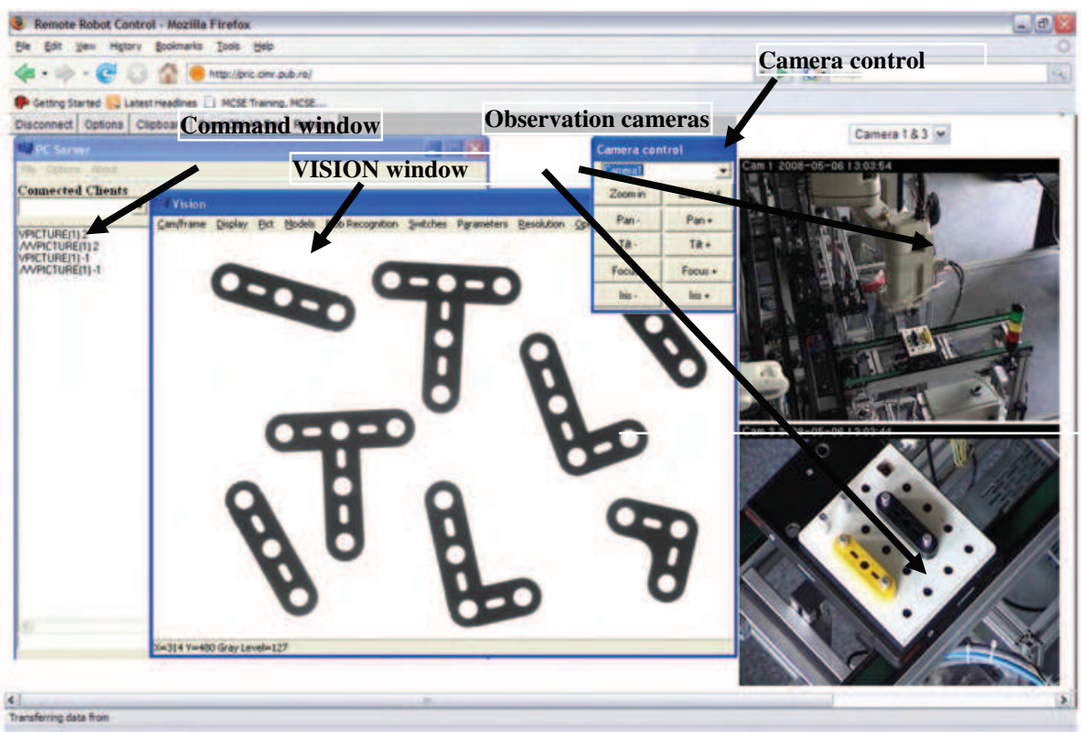

Fig. 9. The Web – based Graphical User Interface.

The advantage of the Vision window is that all commands can be issued using menus, but also the fact that the image acquired by the camera and sent to the server can now be accessed at pixel level. Another major advantage is that the training of part recognition and grasping models become a single-step process during which a unique window is used for parameters and constraints specification (Tomas Balibrea, *et al.*, 1997).
The client application can acquire full or partial images using the VGETPIC V+ operation and send them to the server (Adept Technology, 2001).
Captured image can be processed via the menus (filtering, binarization, convolution, morphing, etc.), saved in a common format and sent to a specialized image processing application. After processing, the image can be sent back to the client and integrated in the image processing module using the VPUTPIC V+ operation, for further use (an application using this mechanism was developed, and consists of a part identifying algorithm based on skeleton computation and matching.
In order to execute vision programs the user must set up the vision parameters such that the system can "see" the objects with the best image quality. The system has specialized functions to automatically establish these parameters or manually if some special settings are required.
After setting the parameters and loading the calibration camera-robot, the user can measure objects, apply filters, apply morphological operators, train and recognize objects.
The measurements include signature analysis; polar and linear offset signatures are computed and stored to be further used in applications. The skeleton computation is also included in the "Measurements" menu item (Borangiu, *et al.*, 2005).
An important feature of the system is the mechanism of training object recognition models and *multiple* grasping positions related to each model in order to provide alternative robot poses for collision-free object grasping at runtime.
Training of object models can be done in two ways:

- Creating a standard ObjectFinder model which is computed by a high level image processing module integrated in the system. The training procedure for this type of model is based on a sequence of steps which have been embedded in the application.
- Selecting a set of object features and storing them in a data structure which will further characterize each model (Borangiu, 2004).

After training and storing models, the user can write applications using these models for object search; the interface can be accessed from a great variety of places.
Therefore great care was taken to ensure optimal use in case of low bandwidth and poor quality hardware. High level of security and availability are key components; they were ensured by the selection of high quality technical devices and well-planned loading. As the portal must be prepared for an increasing number of users, the tool must be highly scalable. It needs to integrate contents and services and provide access for document repositories. User groups must be able to access personalised contents.
High-level availability must be also guaranteed. The system must be scalable according to load and requirements. The use of NLBS (*Network Load Balancing System*) provides a solution for an even load of the network. The portal user interface needs to be customized; templates must be created and uploaded. The authorisations and user groups must be defined.
The eClient application and Lotus Domino Server implement a security access policy to the virtual workspace. The access to the eClient application is granted based on the Domino defined ACL's (*Access Control Lists*), such that the user must specify a user name and a password in order to connect to the application. Two classes of privileges were defined:

1. A **user class** where the operator can observe images acquired from observation web cameras and images from the Vision system taken by multiple area cameras; he can also view the commands issued by the trainer and watch the results of the commands;
2. A **trainer class** where the operator is authorized to issue commands for every connected robot system, upload, download and modify programs. The trainer can also take pictures from an individual camera and use the specific vision tools to process that image. The observation cameras can also be moved and positioned in the desired location by the trainer. The trainer can give users full or partial permission to program testing.

The communication between users is realized by help of an integrated console (text mode) or using an Instant Messaging and Web Conferencing application (Sametime).
IBM Lotus Domino server software was used to combine enterprise-class messaging and calendar / scheduling capabilities with a robust platform for collaborative applications on a wide variety of operating systems. Designing the Lotus Domino Server made available three offerings: Domino Messaging Server (messaging only), Domino Utility Server (applications only), and Domino Enterprise Server (both messaging and applications) (Brooks, *et al.*, 2004). Some of the most important Lotus Domino features were used:

- Encryption, signing, and authentication using the RSA public-key technology, which allows marking a document such that the recipient of this document can decisively find out that the document was unmodified during transmission.
- Access Control Lists (ACLs) determining who can access each database (application) and to what extent.
- Usage of Domino's new features to reduce network utilization. Network compression reduced the number of bytes sent during transactions by up to 50 percent. Connections across heavily loaded links such as WANs and XPCs saw the most benefit.
- Availability for the Windows NT and XP platforms, automatic fault recovery after shutdown and server restart without administrator intervention after the occurrence of an exception. Fault recovery used operating system resources, like message queues.

Because, the application eClient is accessed over the Internet, security represented a critical element of the design. Access to different levels of the application is controlled by xACLs (extended ACLs) to allow or disallow access. The existing database Access Control Lists (ACLs) and the new ACL file feature ensure that application-private databases remain secure. In addition, file protection documents for the Domino Web server used to serve the eClient (Java application) provide additional access control for files accessed via HTTP.

4.3 Resource, resource class, and attribute

This section provides information about the three terms used in RMC: resource, resource class, and attribute, as illustrated in Fig. 10.

Resource

The *resource* is a fundamental concept in the RMC architecture. A resource is an abstraction of an instance of a physical or logical entity that provides services to applications or system components. A system or cluster is composed of numerous resources of various types.
Each resource has the following characteristics:

- An *operational interface* used by its clients. This interface is the reason for the resource's existence. For example, the operational interface of a data file is given by the standard open, close, read, and write system calls.
- A set of persistent data values called *persistent attributes*, describing the characteristics or configuration of the resource (e.g. the file system name, data file name, and so on).

- A set of dynamic data values called *dynamic attributes*, reflecting the current state or other measurement values of the resource (e.g. the disk block usage of a file system).
- A handle that uniquely identifies the resource within the cluster. This value is unique across time and space and is called a *resource handle*.
- A *set of operations* that manipulate the state or configuration of the resource.

Resource class

A *resource class* is a collection of resources having similar characteristics. The resource class provides descriptive information about the properties and characteristics that are common to any resource within the resource class (for example, all the file system resources, files such as prog1.v2 and prog2.v2 belong to the file system resource class in RMC).

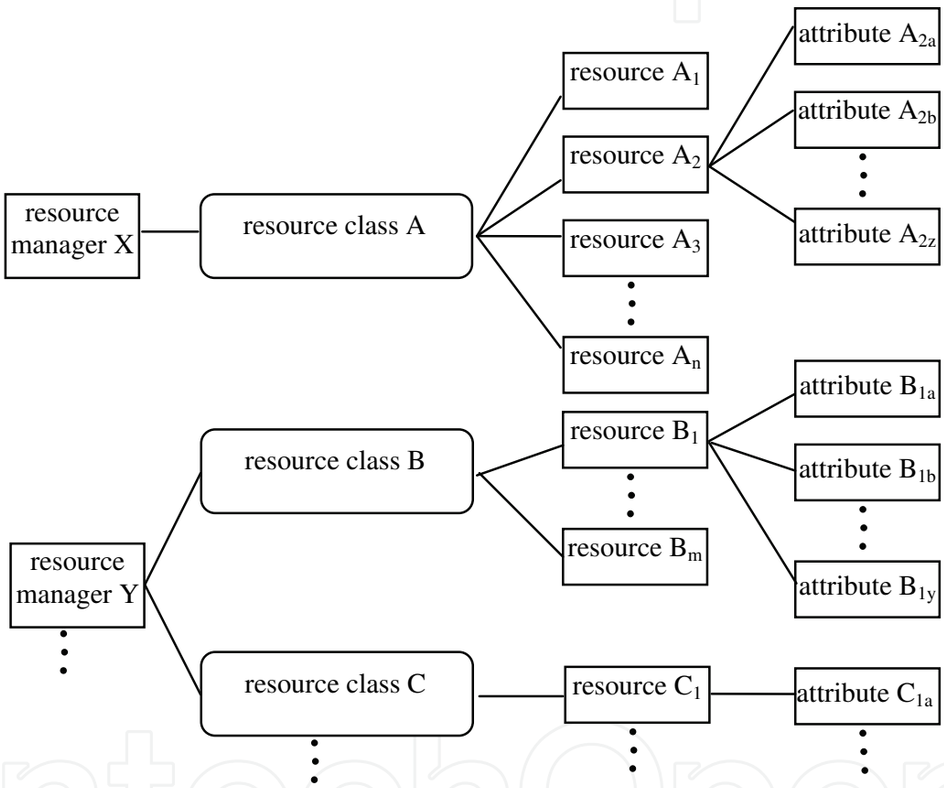

Fig. 10. Resource managers, resource classes, resources, and attributes.

Each resource class has the following characteristics:

- Descriptive information about the *class* and its *resources*.
- A list of the *resource handles* for all existing resources within the class.
- A set of persistent data values called *persistent attributes* that describe or control the operation of the resource class.
- A set of dynamic data values called *dynamic attributes* (e.g. every resource class supports a dynamic attribute that changes whenever the resource number in the class changes.
- An *access control list* (ACL) that defines permissions authorizing users to manipulate or query the resource class.
- A set of *operations* to *modify* or *query* the resource class.

Attribute

A resource class and a resource have several *attributes*. An attribute has a value and a unique name within the resource class or the resource.

A resource attribute is classified into either a *public* or a *private* property. The property is used as a hint for the RMC client for whom the attribute is to be presented to general users. Private attributes typically contain information which is not relevant to general users; therefore private attributes are hidden by default. However, one can display private attributes by specifying a flag in the command. Web-based System Manager does not support the display or modification of private resource attributes.

Attributes fall into two categories: *persistent* and *dynamic*. Each category has a different role, and is used differently by RMC clients:

- **Persistent attributes**: persistent attributes are configuration parameters for resources, and define the resource's characteristics (examples of persistent attributes: Device Name, IP Address, etc). For resource classes, persistent attributes describe or control the operations of the class. For the RMC client, the main usage of persistent resources is to specify which resources will be accessed. They are used as filters over the set of managed resources to select those on which the client wants to act.
- **Dynamic attributes**: Dynamic attributes reflect internal states or performance variables of resources and resource classes. One generally refers to dynamic attributes to define the monitoring condition of the resource to be monitored (for example, if you want to monitor a file system resource, you can monitor the disk space usage of the file system).

4.4 Resource managers

Each resource manager is the interface between the RMC subsystem and a specific aspect of the Adept Windows operating system instance it controls. All resource managers have the same architecture and interact with other RMC components. However, due to their specific nature they have different usage for the end user. Resource managers fall into four groups:

1. *Logging and debugging* (Audit resource manager) The Audit Log resource manager is used by other RMC components to log information about their actions, errors, and so on. The typical end user of Audit Log resource manager is a system administrator which scans these logs to see what is happening in the various RMC components. Unless a problem occurs, there is generally no need for the user to look at these logs.
2. *Configuration* (Configuration resource manager). The configuration resource manager is used by the system administrator to configure the system in a Peer Domain cluster. It is not used when RMC is configured in Standalone or Management Domain nodes.
3. *Reacting to events* (Event Response resource manager). The Event Response resource manager is the only resource manager directly used in normal operation conditions. The CLI allows the end user to define which events he wants to monitor in the nodes, and how the system should react to these events.
4. *Data monitoring* (Host resource manager, File system resource manager, Sensor resource manager). This group contains the file system resource manager and the Host resource manager. They can be seen by the end user as the containers of the objects and variables to monitor. However, it is important to note that these resource managers have no user interface. The end user cannot interact directly with them and has to pass through the Event Response resource manager to access the resources managed by these resource managers.

The only exception is the Sensor resource manager. It is the only resource manager that provides for expandability. Its user interface therefore contains four commands to let the end user define, modify or delete resources to be monitored. Once these resources are created, the normal interface to them is through the Event Response resource manager.

4.4.1 Event response resource manager

The Event Response resource manager (ERRM) plays the most important role to monitor systems using RMC. It uses several terms like *action*, *condition*, *expression*, *response* which will be discussed later.

The Event Response resource manager provides the system administrator with the ability to define a set of conditions to be monitored in the various nodes of the cluster, and to define actions to be taken in response to events (Lascu *et al*., 2002; Lascu, 2005). These conditions are applied to dynamic properties of any resources of any resource manager in the cluster.

The Event Response resource manager provides a simple automation mechanism for implementing event driven actions. Basically, one can do the following actions:

- Define a condition composed of a resource property to be monitored and an expression that is evaluated periodically.
- Define a response that is composed of zero or several actions that consist of a command to be run and controls, such as to when and how the command is to be run.
- Associate one or more responses with a condition and activate the association.

In addition:

- Different users can employ the same conditions and responses, mix and match them to build different monitoring scenarios.
- Many predefined conditions and responses are shipped with the system and can be used as templates for the user who wants to define own monitoring scenarios.
- Any events that occur and any responses that are run are logged in the audit log. One can look at the audit log to check that the defined conditions were triggered, how the system reacted, and what was the result of the actions.

The Event Response resource manager provides one with both a CLI and a GUI (Web-based System Manager). It can then be decided whether you want to use the Web-based System Manager or the CLI. In a stand-alone environment and for setting up limited monitoring of the resources, the Web-based System Manager is the easiest tool to use, unless the user is already familiar with the CLI commands and arguments. However, for managing a cluster or developing sophisticated monitoring scenarios, the recommended tool is the CLI where the user can write his own CLI commands.

Expression

The expression is the means of making computations with data stored in attributes and audit log record fields. An expression is composed of variables, constants and operators, and returns a Boolean value: *true* or *false*.

The same syntax is used regardless of the following three environments; however, the expressions handle different types of variable:

- **ERRM conditions**: trigger events based on dynamic attributes of resources.
- **GUI or CLI**: filter resources based on variables representing persistent attributes of the resources.
- **Audit log expressions**: select audit records with variables being the name of a record field.

Condition

A condition applies to a resource class. It may contain a selection string, which is used to define which resource in the resource class should be monitored. It also contains a Boolean expression that must be satisfied by one (and only one) dynamic attribute of the resource to fire the condition. A condition has a name, so the user can refer to it once defined. A condition can also contain an optional rearm expression.

Fig. 11 shows the relationship between the monitored resource(s), conditions, events, and rearm events.

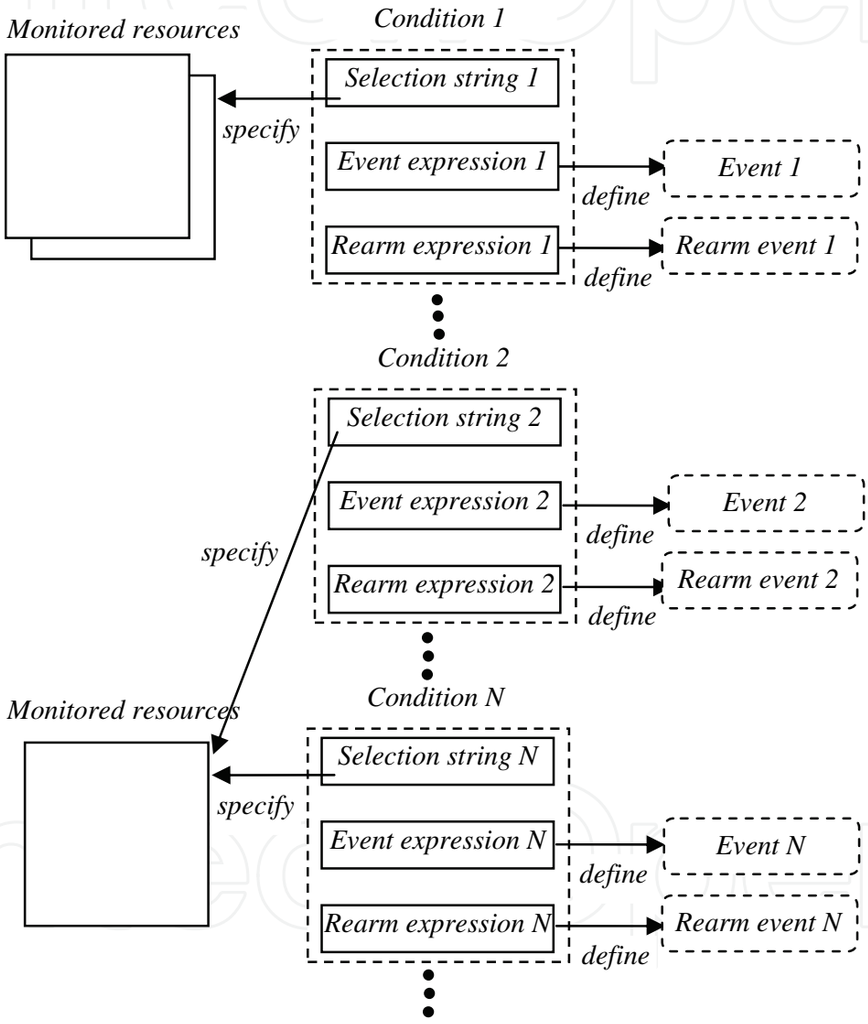

Fig. 11. Condition, expression, and events.

For example, to see if the file system uses more than 90 percent of its available disk space, a predefined condition called "disk space used" is provided with RMC. This condition uses the file system resource class and selects the local disk file system from all the file systems of the monitored nodes. It contains the expression "PercentTotUsed > 90" which applies to the PercentTotUsed attribute of the file system resource.

A condition may also contain a node group list that the event is monitored on. The default value for the node group list is NULL for all the nodes within the management scope. The default value for the management scope is local scope for the local node.
A condition can specify multiple monitored resources selected from a resource class by the selection string. A condition cannot select resources selected from multiple resource classes. Multiple conditions can specify the same resource(s); however, they must have different condition names. If multiple conditions that specify the same resource(s) have the same expression or the rearm expression, then they define the same event or the rearm event. However, you may want to specify different actions for them.
There are no objects defined as *event* and *rearm event*. If an event expression is evaluated as true, the result is called as an event. If a rearm expression is evaluated as true, the result is called as a rearm event.

Action

RMC comes with three *predefined actions*: logevent, notifyevent, and wallevent:

- The *logevent* action appends a formatted string to a user defined log file that can be read later on using the **alog** command.
- The *notifyevent* action sends the content of the event to the user.
- The *wallevent* event broadcasts a message to all users.

One can specify when an action should be executed from as follows:

- Only when the event defined by the associated condition occurs.
- Only when the rearm event defined by the associated condition occurs.
- Either the event or the rearm event defined by the associated condition occurs.

Finally, the user can also provide customized scripts as actions and the existing scripts can be used as examples or starting points for more complex operations. User defined scripts are executed in an environment where variables are set according to the event that would trigger the action. For example, the ERRM_RSRC_NAME variable would contain the name of the resource whose dynamic attribute change caused an event to fire.

Response

A *response* is a set of zero or more actions that will be performed by the system when an expression of a condition evaluates to true. A response contains the following information on how the actions are executed:

- Arguments for the actions.
- Time ranges (periods) to describe when the action can be run.
- Definitions of environment variables to set before executing the action.
- The expected return code of the action and whether the results of the action should be directed, for example, to the audit log.

A response can be defined with multiple actions. For example, the critical notification response calls the following three predefined actions:

- Logging an entry in the criticalEvents.log.
- Sending a message to the user.
- Warning all logged users with **wallevent**.

Association between a condition and a response

Conditions and responses can exist without being used and with nothing related to each other. Actions are part of responses and only defined to responses. Although it is possible that multiple responses have an action using the same name, these actions do not refer to the same object. One can associate a response with multiple actions.
To start observing the monitored resource, a condition must be associated with at least one response. A condition can be associated with multiple responses. However, each pair of a

condition and a response is considered a separate association. Once associated, the condition and response should be activated (started).
Fig. 12 illustrates the relationship between the conditions, the responses, and the actions. In this scheme, there are three associations (A, B, and C).

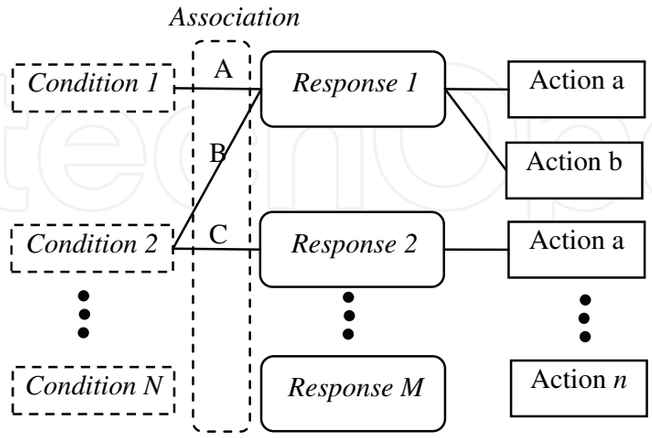

Fig. 12. Conditions, responses and actions.

An association has no name. The labels A, B, and C are for reference purposes. To refer to the specific association, one must specify the *condition name* and the *response name* that make the association. For example, one must specify the condition 1 and the response 1 to refer to the association A. Also, it must be clear that the same action name (in this example, action a) can be used in multiple responses, but these actions are different objects.
ERRM evaluates the defined conditions which are logical expressions based on the status of resources attributes; if the conditions are true a response is executed.

4.4.2 Audit log resource manager

The Audit Log resource manager provides the other RMC subsystems with a logging facility. It can be seen as a means to trace various actions taken by the cluster system.
Typically, these subsystems will add entries in the audit log corresponding to their normal behaviour or to error cases. Examples of information provided in the audit log are:

- Instances of starting and stopping monitoring
- Events
- Actions taken in response to events
- Results of these actions
- Subsystem errors and monitoring errors

Only subsystems can add records to the audit log; there is no user interface that allows the end user to add entries in the audit log.
Only two actions are offered by the Audit Log resource manager:

- **Lsaudrec**: List records in the audit log.
- **Rmaudrec**: Delete the specified records in the audit log.

4.4.3 Configuration resource manager

The configuration resource manager can create and manage a peer domain cluster. Thus, the configuration resource manager need not be used to use RMC on a stand-alone server or in a management domain.

4.4.4 File system resource manager

The File system resource manager manages only one resource class: the local file systems on the cluster node. It can be used to list the file systems, get their status, and retrieve values like the percentage of used space or *inodes*.

4.4.5 Host resource manager

The host resource manager provides the facility to monitor hardware devices and programs running on the cluster nodes. The monitored hardware includes disks, serial lines and network interface adapters. The program monitoring feature allows you to monitor when processes executing any arbitrary program are created or destroyed.

4.4.6 Sensor resource manager

The sensor resource manager allows the user to extend the functionality of RMC for monitoring parts of the Adept Windows robot controller operating system for which RMC does not provide a predefined resource class or attribute.

The sensor resource manager provides four commands to create, rename, or remove new resources called sensors. A *sensor* is a command that will be periodically executed, returning a value that can be monitored by defining a condition in the ERRM. By default, the period of execution is 60 seconds, but this period can be changed by the user to another value when creating the sensor.

If more than one returned value is needed, it is possible to define sensors returning multiple "property = value" pairs. This feature can be used, for example, to implement behaviours, such as notify user of the value of properties 2 and 3 when property 1 exceeds a threshold.

5. Solution implementing for networked robots

In order to implement the solution on a network of robot controllers, first a shared storage is needed, which must be reached by any controller from the cluster.

The file system from the storage is limited to NFS (*network file system*) by the operating system of the robot controllers (Adept Windows). Five Adept robot manipulators were considered, each one having its own multitasking controller.

For the proposed architecture, there is no option to use a directly connected shared storage, because Adept robot controllers do not support a Fiber Channel Host Bus Adapter (HBA). Also the storage must be high available, because it is a single point of failure for the Fabrication Cluster (FC), see Fig. 13.

Due to these constraints, the solution was to use a High Availability cluster to provide the shared storage option (NFS Cluster), and another cluster composed by Adept Controllers which will use the NFS service provided by the NFS Cluster (Fig. 13).

The NFS cluster is composed by two identical IBM xSeries 345 servers (2 processors at 2.4 GHz, 1GB RAM, and 75GB Disk space, two RSA 232 lines, two Network adapters, and two Fiber Channel HBA), and a DS4100 storage. The storage contains a Quorum volume used by the NFS cluster for communication between nodes, and a NFS volume which is exported by the NFS service which runs in the NFS cluster. The servers have each interface (network, serial, and HBA) duplicated to assure redundancy (Anton *et al*., 2006; Borangiu *et al*., 2006).

In order to detect the malfunctions of the NFS cluster, the servers communicate using heartbeat packets to ensure that the communication is established.

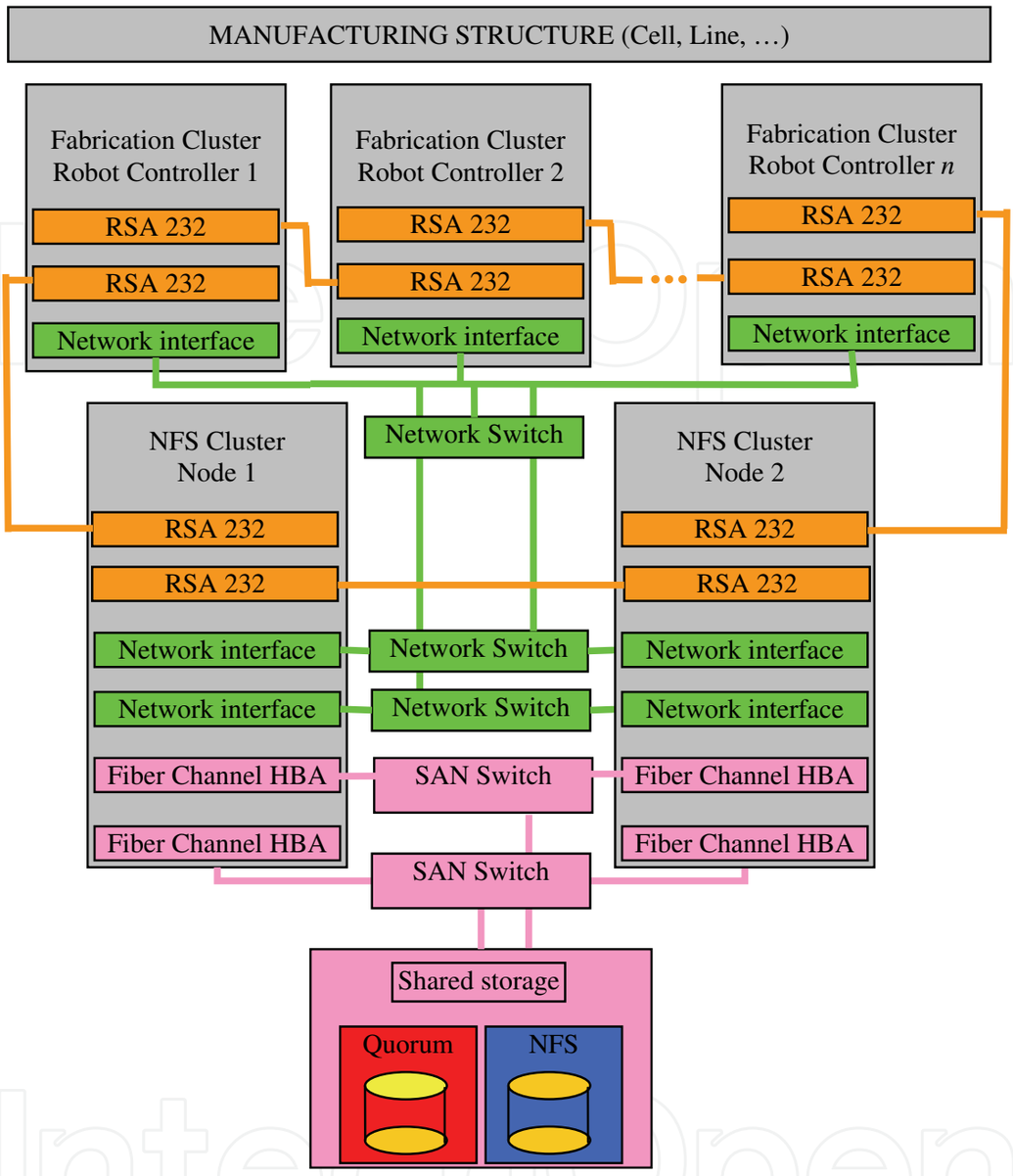

Fig. 13. Implementing the high availability solution for the networked robotic system.

There are three communication routes: the first route is the *Ethernet network*, the second is the *quorum volume* and the last communication route is the *serial line*. If the NFS cluster detects a malfunction of one of the nodes and if this node was the node which served the NFS service the cluster is reconfiguring as follows:

1. The server which is still running writes in the quorum volume which is taking the functions of the NFS server, then
2. Executes the resource takeover operation (the resource being the NFS volume) and mounts the NFS volume, then
3. Takes the IP of the other server (IP takeover) and
4. Starts the NFS service.

In this mode the Fabrication Cluster is not aware about the problems from the NFS cluster, because the NFS file system is further available.
The Fabrication Cluster can be composed by at least two robot controllers (nodes) – *group leader* and *group leader backup*. The nodes have resources like: robot manipulators (with attributes like: collision detection, current robot position, etc...), serial lines, Ethernet adapter, variables, programs, and NFS file system. The NFS file system is used to store programs, log files and status files. The programs are stored on NFS to make them available to all controllers, the log files are used to discover the causes of failure and the status files are used to know the last state of a controller.
In the event of a node failure, the production flow is interrupted. In this case, if there is a connection between the affected node and the group leader, the leader will be informed (if not, the heartbeat detects the failure) and the GL takes the necessary actions to remove the node from the cluster. The GL also reconfigures the cluster so the manufacturing process will continue. For example if one node cluster fails in a three-node cluster, the operations this node was doing will be reassigned to one of the remaining nodes.
The communication paths in the multiple-robot system are the *Ethernet network* and the *serial network*. The serial network is the last communication resort due to the low speed and also to the fact that it uses a set of Adept controllers to reach the destination (because it is a ring type connection). In this case the *ring network* will be down if more than one node fails.
Rebuilding the communication route in the network represents the last step required to restart the normal behave of the FC control system.
In case of malfunction of the communication network the following important cases can appear:

1. If the connection between the Switch and the Supervisor PC (the PC where the Server Application runs) is down the remote control will be lost, but the FMC will reconfigure as follows: the controller will use the Ethernet network for communication, and the controller with the first IP from the class will take the functions of the Supervisor PC. If the connexion is re-established the Supervisor PC makes a query, finds the replacing controller, transfers the databases and restarts the normal behaviour.
2. If the switch is not functioning, all the Ethernet connexions are lost, but the controllers will use the serial "network". The behaviour is like in the first case only that the web users can view the status from the Supervisor PC, including the images acquired by the observation cameras.
3. If a controller loses the Ethernet connexion, it will use one of the two serial lines to reach the Supervisor PC depending on the CPU time of the neighbours.

6. Conclusion

Fault-tolerance is provided to the cell communication system (Fig. 14), and therefore redundancy at both Station Controller level (a break down of a Robot Controller is detectable, the production tasks can be rescheduled to the remaining valid units for graceful degraded behaviour) and Station Computer level (replication of data bases for the IBM PC-type device terminals, reassignment of computers in case of break downs).
The fault tolerance solution presented in this paper is worth to be considered in environments where the production structure has the possibility to reconfigure, and where the manufacturing must assure a continuous production flow at batch level (job shop flow).
There are also some drawbacks in this solution, like the need of an additional NFS cluster. The spatial layout and configuring of robots must be done such that one robot will be able to

take the functions of another robot in case of failure. If this involves common workspaces, programming must be made with much care using robot synchronizations and monitoring continuously the current position of the manipulator.

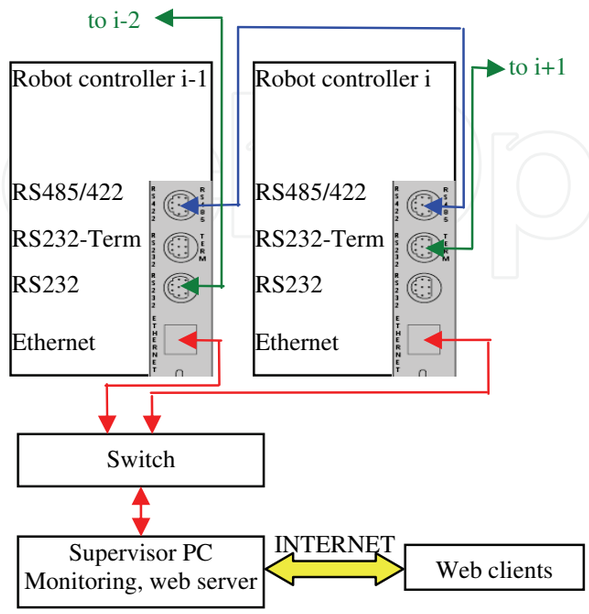

Fig. 14. Fault-tolerant communication architecture.

The advantage of the proposed solution is that the structure provides a continuous production flow with an insignificant downtime (during reconfiguration).

The solution was tested on a five-robot assembly cell located in the Robotics and IA Laboratory of the University Politehnica of Bucharest. The cell also includes two CNC milling machines and one Automatic Storage and Retrieval System, for raw material feeding and finite products storage.

During the tests, the robot network has detected a number of errors (end-effector collision with parts, communication errors, power failure, etc.) The robotic network has evaluated the particular situation, and the network was reconfigured and the abandoned applications were restarted in a time between 0.2 and 3 seconds. The network failure was also simulated during tests (Fig. 15).

One robot (R2) was disconnected from the Ethernet network, the heartbeat packet sent by the robot to the other cluster members has detected the malfunction and the robot has switched the communication using the serial line; this was done in 0.3 seconds after the Ethernet cable was removed. The communication between the affected robot and its neighbours was done using the serial lines, and the communication with other robots was done by routing the communication using the Ethernet line of the neighbours (R1 and R3). In this way the communication latency was reduced.

After the communication was re-established, the serial lines of the robot were disconnected. The robot has detected the communication failure, stopped the manipulation program and retracted the manipulator in a home position in the exterior of the working area in 0.8s.

The neighbours have sent the heartbeat packets using the serial lines and detected that they do not have any connection with the robot and announced the group leader (GL) which has

removed the robot from the cluster and reconfigured the cell (see Fig. 15). The neighbour R1 having the same working area as the affected robot R2 has loaded the values of the variables and the production program from the shared storage, and started production, continuing from the point where R2 has been stopped. The cell reconfiguration from the point where the serial lines were disconnected has taken 2.2 seconds.

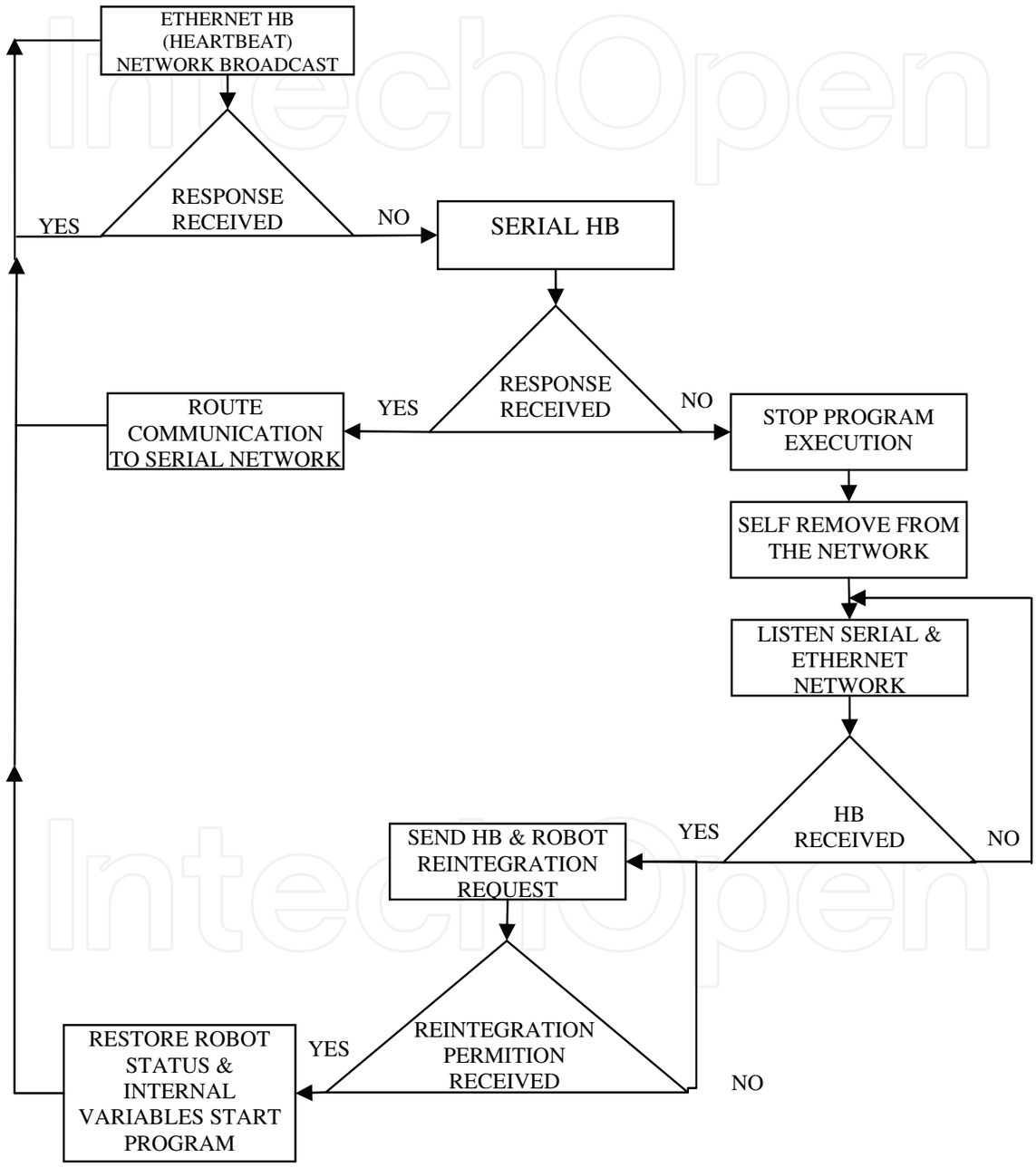

Fig. 15. Algorithm for network failure detection and reconfiguration.

Another communication test consisted in disconnecting both the serial lines and the Ethernet line at the same time, in this case the cluster tested the communication sequentially

and the cluster reconfiguration took 2.5 seconds. When configuring the cluster in order to test in parallel the communication lines, the reconfiguration took 2.3 seconds but the controllers processed a higher communication load.
The most unfavourable situation occurred when a robot manipulator was down; in this case the down time was greater because the application which was executed on that controller had to be transferred, reconfigured and restarted on another controller. Also if the controller still runs properly it will become group leader to facilitate the job of the previous GL (that has also a manipulation task to do).
The proposed solution is not entirely fault tolerant; however, in some situations it could be considered as a fault tolerant system due to the fact that even if a robot controller failed, the production continued in normal conditions.
The project is in the final stage of development and will be finished at the end of 2008. The research project will provide a portal solution for linking the existing fault tolerant pilot platform with multiple V+ industrial robot-vision controllers from Adept Technology located in different labs.

7. References

Adept Technology Inc., (2001). *AdeptVision Reference Guide Version 14.0 Part Number 00964-03000,* San Jose, Technical Publications.

Anton F., D., Borangiu, Th., Tunaru, S., Dogar, A., and S. Gheorghiu, 2006. Remote Monitoring and Control of a Robotized Fault Tolerant Workcell, *Proc. of the 12th IFAC Sympos. on Information Control Problems in Manufacturing INCOM'06*, Elsevier.

Borangiu, Th., (2004). Intelligent Image Processing in Robotics and Manufacturing, Romanian Academy Press, Bucharest.

Borangiu, Th., F.D. Anton, S. Tunaru and N.-A. Ivanescu, (2005). Integrated Vision Tools And Signature Analysis For Part Measurement And Recognition In Robotic Tasks, IFAC World Congress, Prague.

Borangiu, Th., Anton F., D., Tunaru, S., and A. Dogar, 2006. A Holonic Fault Tolerant Manufacturing Platform with Multiple Robots, Proc. of *15th Int. Workshop on Robotics in Alpe-Adria-Danube Region RAAD 2006*.

Brooks, K., R. Dhaliwal, K. O'Donnell, E. Stanton, K. Sumner and S. Van Herzele, (2004). *Lotus Domino 6.5.1 and Extended Products Integration Guide*, IBM RedBooks.

Harris, N., Armingaud, F., Belardi, M., Hunt, C., Lima, M., Malchisky Jr., W., Ruibal, J., R. and J. Taylor, 2004. *Linux Handbook: A guide to IBM Linux Solutions and Resources,* IBM Int. Technical Support Organization, 2nd Edition.

Lascu, O. et al, 2005. *Implementing High Availability Cluster Multi-Processing (HACMP) Cookbook,* IBM Int. Technical Support Organization, 1st Edition.

Lascu, O., Sayed, R., Carroll, S., Coleman, T., Haehnel, M., Klabenes, P., Quintero, D., Reyes, R., Jr., Tanaka, M., Truong, D., D., 2002. *An Introduction to Security in a CSM 1.3 for AIX 5L Environment,* IBM Int. Technical Support Organization, 1st Edition.

Matsubara, K., Blanchard, B., Nutt, P., Tokuyama, M., and T. Niijima, 2002. *A practical guide for Resource Monitoring and Control (RMC),* IBM Int. Technical Support Organization, 1st Edition.

Tomas Balibrea, L.M., L.A. Gonzales Contreras and M. Manu (1997). *Object Oriented Model of an Open Communication Architecture for Flexible Manufacturing Control*, Computer Science 1333 - Computer Aided System Theory, pp. 292-300, EUROCAST '97, Berlin.

Advances in Robotics, Automation and Control

Edited by Jesus Aramburo and Antonio Ramirez Trevino

ISBN 978-953-7619-16-9
Hard cover, 472 pages
Publisher InTech
Published online 01, October, 2008
Published in print edition October, 2008

The book presents an excellent overview of the recent developments in the different areas of Robotics, Automation and Control. Through its 24 chapters, this book presents topics related to control and robot design; it also introduces new mathematical tools and techniques devoted to improve the system modeling and control. An important point is the use of rational agents and heuristic techniques to cope with the computational complexity required for controlling complex systems. Through this book, we also find navigation and vision algorithms, automatic handwritten comprehension and speech recognition systems that will be included in the next generation of productive systems developed by man.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen