MASTER

# Towards one interface standard for process monitoring applications the design of an OPC server for systems controlled by OS-9

Krieckaart, J.

*Award date:*
2002

TU/e technische universiteit eindhoven

Aan BOC Edwards
Pharmaceutical Systems
t.a.v. de Hr. Corver

POBox 111
5100 AC DONGEN

Den Dolech 2
Postbus 513
5600 MB Eindhoven
Nederland
Telefoon 040 - 247 3195
Telefax 040 - 244 8375
E-mail j.c.m.d.valk.roulaux@tue.nl
E-hoog 1.09

| Uw kenmerk | Ons kenmerk | Datum |
|---|---|---|
| - | X 2002.360/WB/av | 25 juni 2002 |

Onderwerp
Embargo op verslag

Geahte heer Corver,

Naar aanleiding van Uw brief van 3 juni 2002 en ons telefonisch onderhoud op 24 juni j.l. kan ik U berichten dat wij een embargo op het afstudeerverslag van Jos Krieckaart zullen leggen voor de duur van een jaar, tot 15 juni 2003.

Een verdere verlenging van dit embargo is in principe niet mogelijk. Aangezien het verslag een onderdeel vormt van de voorwaarden voor het verlenen van het Ir-examen, dient het wettelijk ook openbaar en toegankelijk te zijn. Derhalve moet het na het verstrijken van het embargo in onze bibliotheek publiek ter beschikking staan. Wij kunnen daarom niet voldoen aan Uw verzoek om het verslag naar Uw bedrijf te retourneren.

Met vriendelijke groeten, namens de examencommisie van de faculteit Elektrotechniek,

Prof.dr.ir. W.M.G. van Bokhoven, voorzitter

# TOWARDS ONE INTERFACE STANDARD FOR PROCESS MONITORING APPLICATIONS

## The Design of an OPC Server for Systems Controlled by OS-9

**graduation report**

**AUTHOR:** J. Krieckaart

Eindhoven / Dongen: - 06 – 2002

**Supervisors BOC Edwards PS:** ir. J. Corver / ing. A. Schaepman
**Supervisors TUE:** Prof. Dr. ir. M.P.J Stevens / Dr. ir.P. v.d. Putten

## SUMMARY

*** *tbd*

## TABLE OF CONTENTS

## ABBREVIATIONS & TERMS

| | |
|---|---|
| BOC Edwards | BOC Edwards, Pharmaceutical Systems |
| cPCI | Compact Peripheral Component Interconnect |
| HMI | Human Machine Interface |
| ILF | In Line Filler |
| LAT | Loading Accumulation Table |
| OPC | OLE for Process Control |
| OS-9 | A real-time operating system by Microware |
| PLC | Programmable Logic Controller |
| NCCW | Non Contact Check Weigher |
| NMR | Nuclear Magnetic Resonance |
| TCAR | Transporter |
| UAB | Unloading Accumulation Buffer |
| UAT | Unloading Accumulation Table |
| WFA | Weight Filling Automation |
| SCADA | Supervisory Control and Data Acquisition |
| VME | Versa Module Europa |

## LIST OF FIGURES

## LIST OF TABLES

## LIST OF TEXTS

## 1. INTRODUCTION

BOC Edwards Pharmaceutical Systems is a company that produces machines for the pharmaceutical industry. Their product range varies from washing machines and sterilisation tunnels to fluid fillers to loading systems. All these machines have their own control system. Together with the variance of the complexity of the different machines the complexity of the control systems also varies a lot.

BOC Edwards Pharmaceutical Systems manufacturers their machines on a project-oriented base. Customers have great influence on design decisions that have to be made. This also involves the decisions that have to be made for the control systems of the machines. This brings about that for every new machine that is being build new problems can rise and have to be solved.

I started my graduation project in September 2001. My task was to take a look at the current situation concerning the control systems that are used to control the machines, find identify the problem areas and propose some improvements. One of the possible improvements could be tested in the form of an experiment. This graduation report describes this various stages of my graduation project at BOC Edwards Pharmaceutical Systems.

The experiment that I performed during my graduation project is the design of an OPC Server for OS-9 Controlled Systems. Such a server enables BOC Edwards Pharmaceutical Systems to integrate their most complex machines with SCADA systems.

The chapters 2 through 4 describe the different machines BOC Edwards Pharmaceutical Systems produces, identify the problem areas and describe which experiment is implemented. The chapters 5 through 8 describe the design of the OPC Server for OS-9 Controlled Systems. The chapters 9 through 11 describe the results of the tests that were performed to test the design and give conclusions and recommendations.

## 2. INVENTORY OF THE CURRENT SITUATION

The complexity of the different machines varies greatly. Some machines can be controlled by relays, if they are to be used in a stand-alone configuration. Other, more complex, machines are equipped with a control system that is based on a VME rack, which accommodates multiple CPU's.

### 2.1 General

The least complex machines, such as a Capper or a Washer can be controlled by relays only. However, when these machines are equipped with a more advanced HMI or are coupled with a SCADA system they are controlled by a PLC.

Most machines are controlled by a PLC. The customer often determines the brand of PLC that is being used on a machine. As a result of this, it happens that the control system for the same type of machine must be developed more than once. Each PLC manufacturer has its own preferred field bus and communication network between different PLC's and SCADA systems. As a result, the electrical design of a machine changes also, when another brand of PLC is being used. In order to be able to use the same sensors and actuators with different PLC's these devices are connected to the field bus with adapter blocks. When another brand of PLC is used, and together with the brand of PLC another field bus is being used, only the adapter blocks need to be changed and not all sensors and actuators.

There are four brands of PLC's that are being used. These are Allen-Bradley, Siemens, Modicon and Mitsubishi. Depending on the complexity of the machine, different types of PLC's may be used per manufacturer. The programming of the PLC's is always done with the programming tools that are provided by the manufacturers. Development is done on Windows PC's. The PLC's are programmed using ladder logic. In Table 2-1 you can see a survey of the different brands of PLC's some of their PLC types, programming software, preferred field bus and network.

**Table 2-1: Survey of PLC's used**

| Brand | Type | Programming Software | Field bus | Network |
|---|---|---|---|---|
| Allen-Bradley | SLC 500<br>PLC-5 | RSLogix 500<br>RSLogix 5 | Device Net | Ethernet<br>Data Highway Plus |
| Siemens | Simatic S7-200<br>Simatic S7-300<br>Simatic S7-400 | Step 7 | Profibus | Ethernet |
| Modicon | Quantum | ProWorx | Profibus | Modbus+ |
| Mitsubishi | Melsec AnSH | GppWin | SSC-Net<br>CC-Link | Melsecnet B<br>Melsecnet 10 |

More complex machines, particularly filling machines, are not controlled by a PLC, but by a rack based computer system. Designers have more flexibility when designing control systems based on computer systems then with PLC's. Complex algorithms are very difficult to implement on PLC's, whereas computer systems are very well suited to do that. Besides, computer based systems can have a much higher performance then PLC's when they have to calculate complex algorithms. Two different computer based systems are used on different filling machines. One of them is based on a gespac rack, and the other on a VME rack. In both systems the main controller is a 68k series processor from Motorola. The processor on then main controller board of the gespac system is a 68030 and in the VME system a 68040. Both control system run the operating System OS-9. The gespac system uses version 2.9 and the VME system version 3.0.3. Programming of the two different systems varies, too. The software on the gespac system is developed

on the target system, whereas the software of the VME system is developed on a Windows and then uploaded to the VME control system's main controller board.

Software development is done using the same method for all machines. Software development is carried out independent of the control system of the machine. The software development method used is the method of Hatley and Pirbhai. This method is a so-called structural design method. By using this method, the processes the control system has to perform are identified as well as the communications between these processes.

## 2.2 Washer

### 2.2.1 Introduction

The Washer is a machine that is used to wash empty vials before they are sterilised and filled. The washing is executed in stages. In the first stage the Washer picks up some vials and turns them upside down. During the next stages the vials are held above the washing stations. At the washing stations, water is spout into the vials. When the vials have passed all washing stations, the vials are turned upright and shifted out of the Washer. The movements from the pick up point to the washing stations, between the washing stations and to the outfeed are mechanically coupled. In Figure 2-1 you can see a sketch of the Washer. The arms of the turning, inner part of the washer move from one station to another and reside at a station for an amount of time. Each swerve one arm is placed at the infeed and one at the outfeed. All other arms that carry vials are placed above a washing station. When the turnable inner part has stopped, vials are fed into the arm at the infeed and moved out of the arm at the outfeed.

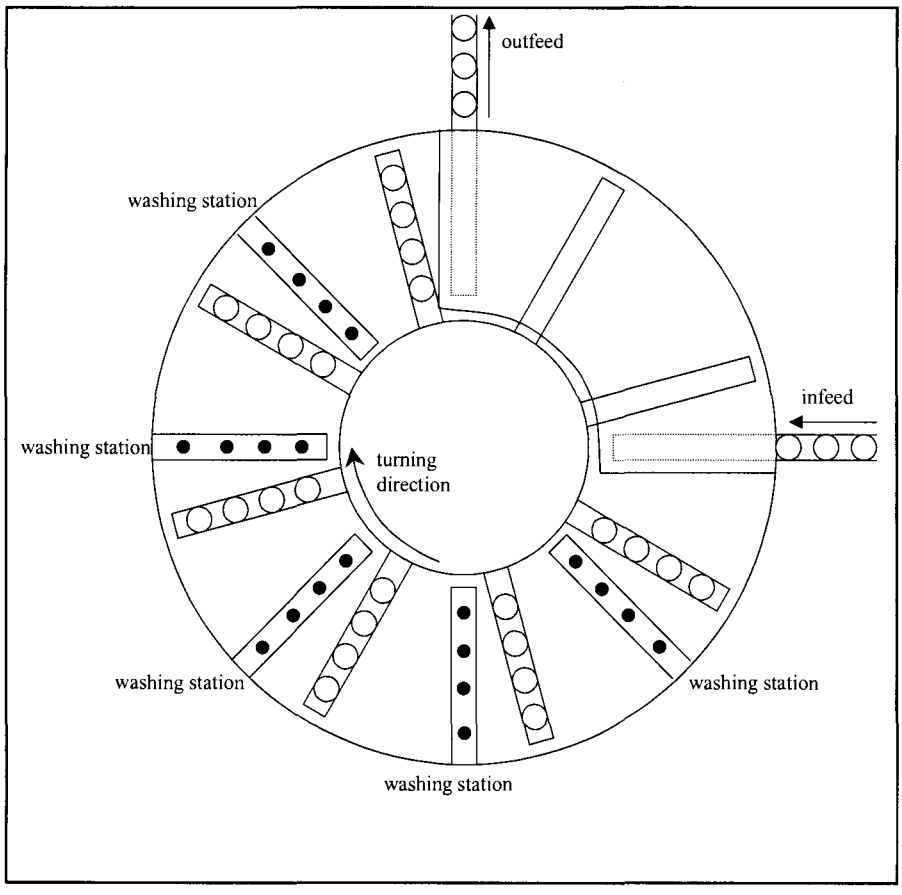


**Figure 2-1: Sketch of the Washer**

### 2.2.2 The Control System

The Washer is quite simple when compared to some other machines. It is controlled by relays or by a PLC. It can be controlled by relays. However, it is often equipped with a PLC in order to be able to integrate the machine in a SCADA system or to use a more advanced HMI.

No field busses are being used on the Washer. All inputs and outputs are directly connected to the relays or the PLC. The control system of the Washer uses about 20 digital inputs, about 25 digital outputs, 4 analog outputs and 2 serial communication ports. In Figure 2-2 you can see the components of the control system.

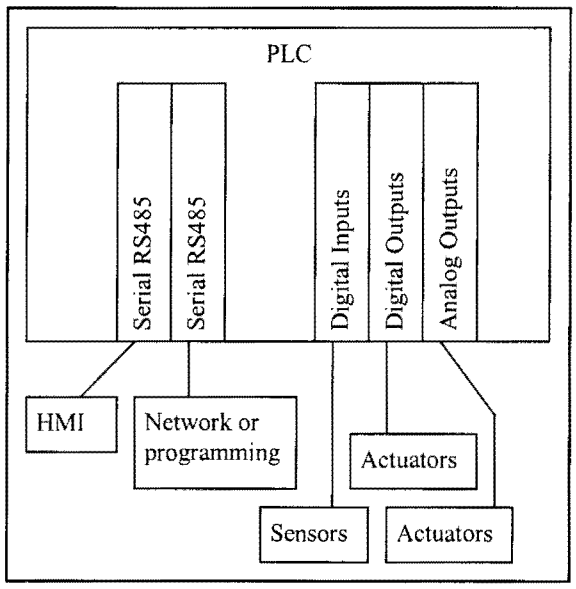


**Figure 2-2: The Control System of the Washer**

### 2.2.3 Software

The software design is done using the method of Hatley and Pirbhai. This method is used to identify the processes the control system has to execute and the communication between these processes.

None of the sensors or actuators of the Washer have a high rate of change. The control system is basically only monitoring the machine status ones it is started. Only during start-up, shutdown or when entering or recovering from an alarm state the rate of change of some sensors can be a bit higher. This causes the control system software to be quite simple and it doesn't have many very time critical parts.

In Figure 2-3 and Figure 2-4 you can see the first the first two diagrams that were created during the software design process.

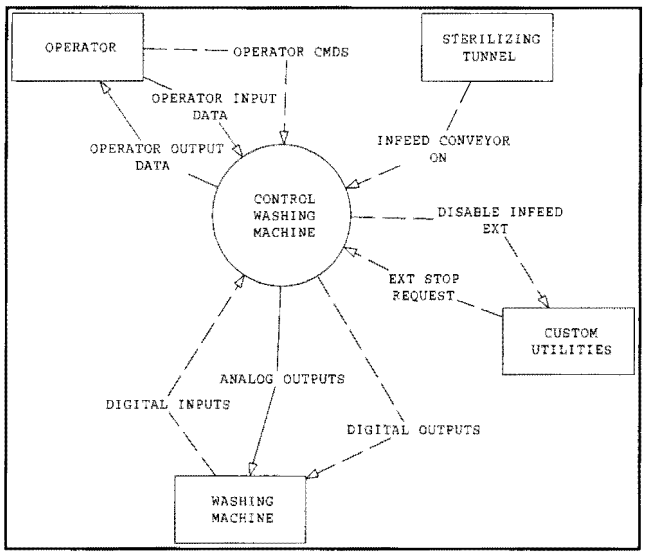


**Figure 2-3: Context Diagram of the Washer**

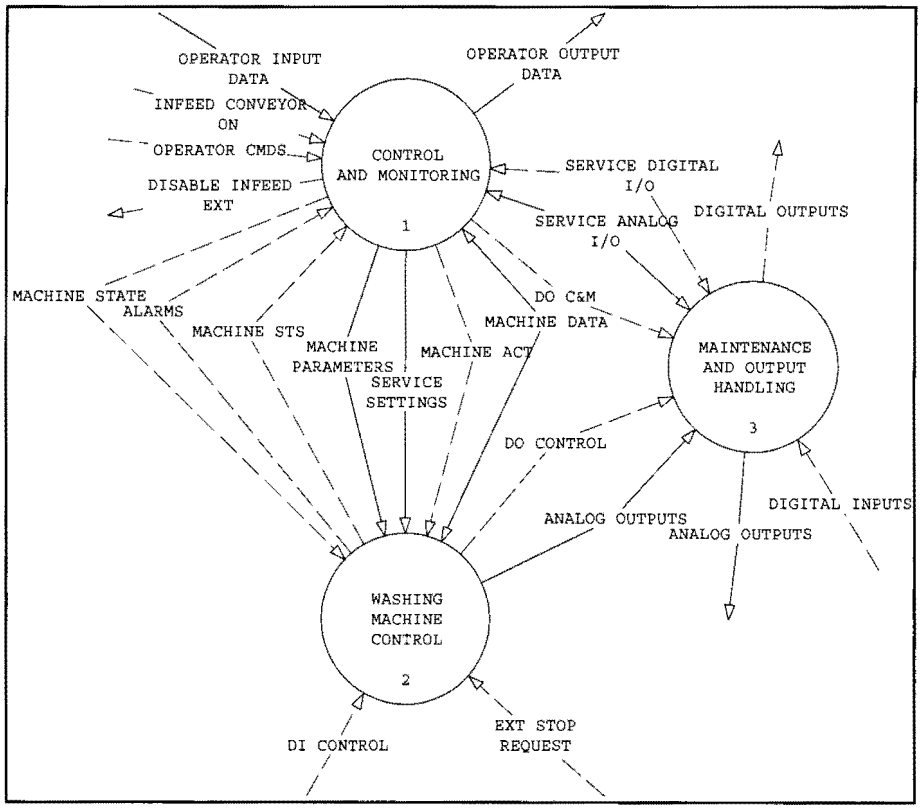


**Figure 2-4: Data Flow Diagram 0 of the Washer**

If the control system is based on a PLC, it is programmed with ladder logic. The programming is done with the use of the specific tool that is supplied by the manufacturer of the PLC.

## 2.3 In Line Filler (ILF)

### 2.3.1 Introduction

The ILF is a filling machine. It is used to fill different sizes of vials. There are two versions of the ILF. In one version the filling volume is being calculated by a time-pressure process. The other version uses so-called rotary pumps to do that.

The infeed unit of the ILF places the vials in a single line with no empty positions. Subsequently the vials are placed on the main transport at a fixed pitch. The vials are then transported underneath the filling section. The vials are filled while moving. They are not stopped. The filling unit moves with the vials, fills them and moves back to start filling the next vials. When the vials are filled a stopper is placed on it.

The ILF contains two weighing units. They are used to check the filling process. A vial is weighed before it is filled and the same vial is weighed after it is filled. The difference in weigh indicates if the filling is appropriate. If necessary the parameters of the filling process are adjusted.

The vials are taken from the main transport by grasping arms and placed on the weighing units. At the same time another grasping arm puts the vial that is weighed before back into the main transport, so no empty pockets in the main transport will occur. The grasping arms are driven by stepper motors.

After the stopper is placed on a vial, it arrives at the rejection unit. When the check weighing indicated that a vial is out of tolerance, it will be moved to the rejection tray. Otherwise it will continue to the regular outfeed. In Figure 2-5 you can see a sketch of the ILF.

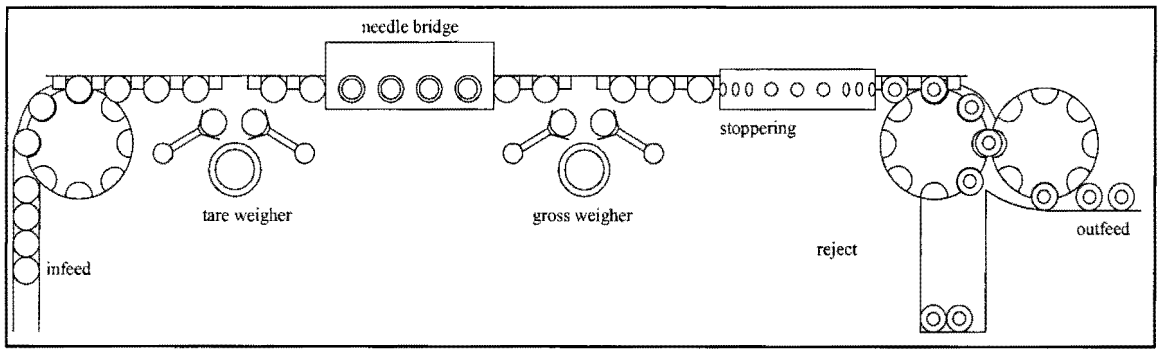


**Figure 2-5: Sketch of the ILF**

### 2.3.2 Control System

The base of the control system of the ILF is a VME rack. It is a so-called 6U rack with 14 slots. The main controller in this rack is based on a 68040 processor from Motorola. The control system of the two versions of the ILF is basically the same. The following table and pictures are based on an ILF with the time-pressure filling algorithm. In Figure 2-6 and Table 2-2 you can see how the control system of the ILF is build together.

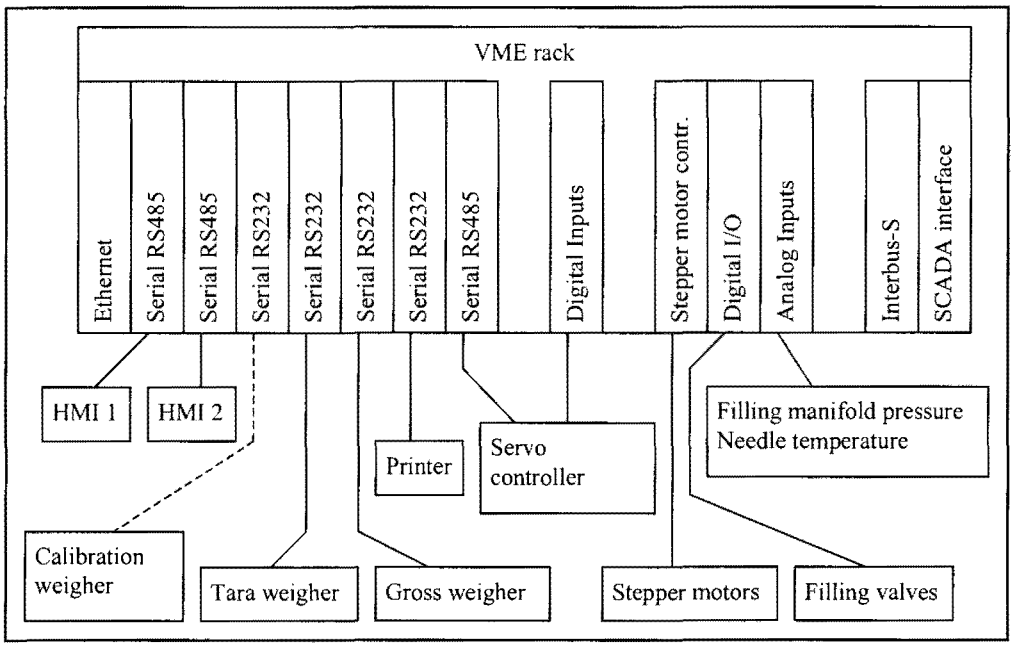


**Figure 2-6: The Control System of the ILF**

**Table 2-2: The Parts of the Control System of the ILF**

| Part | Function | Location | Manufacturer |
|---|---|---|---|
| PG1226/00 | Rack | | Philips / Nyquist |
| PG2056/60 | Main Processor 68040 | Rack | Philips / Nyquist |
| M363 | Ethernet (for service) | PG2056/60 | Inducom |
| PG2274 | SRAM disk | Rack | Philips / Nyquist |
| VB60 Ace | DSP for Time-Pressure | Rack | Perimos |
| M66 | Digital In/Out for Time-Pressure | VB60 Ace | Men |
| M59 | Analog In for Time-Pressure | VB60 Ace | Men |
| PG3984 | M-Carrier | Rack | Philips / Nyquist |
| M802 | Digital In | PG3984 | Philips / Nyquist |
| M806 | Serial In/Out | PG3984 | Philips / Nyquist |
| M806 | Serial In/Out | PG3984 | Philips / Nyquist |
| M806 | Serial In/Out | PG3984 | Philips / Nyquist |
| PG3984 | M-Carrier | Rack | Philips / Nyquist |
| M822 | Stepper motor controller | PG3984 | Philips / Nyquist |
| M822 | Stepper motor controller | PG3984 | Philips / Nyquist |
| M822 | Stepper motor controller | PG3984 | Philips / Nyquist |
| M822 | Stepper motor controller | PG3984 | Philips / Nyquist |
| PLC5/V30B | PLC (used for SCADA coupling) | Rack | Allen-Bradley |
| IBS VME6H SC/I-T | Interbus-S master controller | Rack | Phoenix Contact |

The heart of the control system is a Motorola 68040 processor, which is located at the main controller board. Most of the sensors and actuators on the ILF are connected to the control system with an Interbus-S field bus. Some critical devices, which have hard real-time constraints and need a high accuracy, are connected to directly to the control system. These devices include the filling valves and the stepper motors for the weighing units' grasping arms.

The field bus used on the ILF is Interbus-S. Interbus-S is an open standard that is not lead by a leading PLC manufacturer. This in contradiction with Profibus, Devicenet and CC-link which are the preferred field buses for Siemens, Allen-Bradley and Mitsubishi respectively.

An Interbus-S field bus consists of a single master and a maximum of 256 nodes. The network is carried out as a token ring, where each node is a receiver and a sender in the ring. The hardware implementation of Interbus-S is an RS422 implementation.

The control system can also contain a PLC. This PLC is placed in the VME rack. The PLC is not used to perform any control tasks, but only for coupling to a SCADA system.

The ILF has about 120 digital inputs, 110 digital outputs, 10 analog inputs and 20 analog outputs. Most of these inputs and outputs are connected to the field bus. Some are directly connected to the control system. About half of the inputs and outputs have a rate of change that is in the range of the number of vials per minute that are filled. The operation speed is given in vials per minute. The other inputs and outputs are connected to devices that monitor the machine's status or indicate the current state of the system. During startup or shutdown these devices can have a higher rate of change.

Besides the digital and analog inputs and outputs the control system of the ILF uses 7 serial communication ports to communicate with some parts on the machine. These are the weighing units, the HMI's, a printer and the servo controller. All these devices have some intelligence inside them. The HMI's have build in functionality for steering its display and keyboard. The weighing units are capable of doing measurements without much interference of the control system.

Besides the main controller, there is an other cpu in the control system. It is located on a separate board in the VME rack. It is a DSP that is used to control the actual filling process. It calculates the time that the filling valves must be opened for achieving the right filling volume. It uses the pressure and temperature of the liquid as inputs. For each needle of the filling unit there are some adjustable parameters that the DSP uses to calculate the filling time. These parameters are adjusted as a result of the check weighing that is performed by the weighing units. The DSP also takes care of the actual opening and closing of the filling valves. The DSP board carries its own input and output extensions for reading and steering the devices used in the time-pressure filling process. In the latest version of the ILF the DSP board is replaced with another 68k board. This board is the A10 from Men. It is equipped with a 68060 processor. This new board carries the same input and output mezzanine extensions as the original DSP board.

### 2.3.3 Software

The software design is done using the method of Hatley and Pirbhai. This method is used to identify the processes the control system has to execute and the communication between these processes. In Figure 2-7 and Figure 2-8 you can see the first two diagrams that were created during the software design process. These are the context diagram and data flow diagram 0. During the software design process these diagrams were further detailed. The result is that the processes of the system are identified along with the communication between these processes. The operating system used on the main controller of the control system of the ILF is OS-9 version 3.0.3 from Microware. It is compiled at a host PC running Microsoft Windows. The used compiler is a cross-compiler called Ultra C/C++, also from Microware. The ILF is programmed mostly using standard C. In addition some objects are used for interaction with some of the devices on the ILF. These objects are written in C++. The editor that is used for writing the code is Codewright, version 5.0c.

The scheduler in OS-9 is set to sequential scheduling. The created processes are enabled one after the other in a fixed order. There are only a few processes that are enabled on a priority base. These few processes have hard real-time constraints and require a fast response.

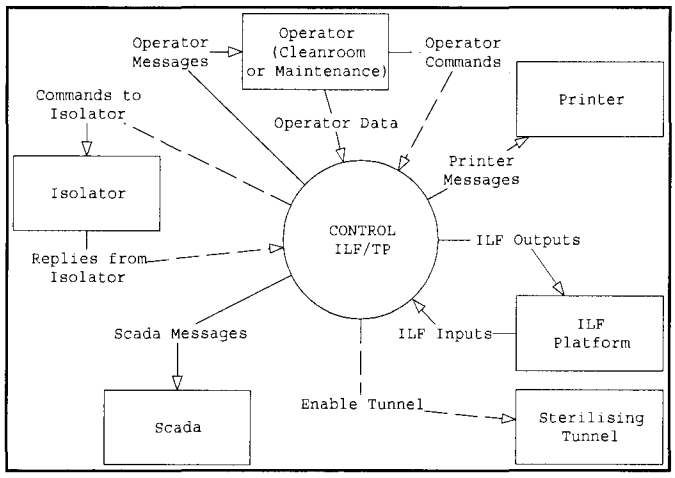


**Figure 2-7: Context Diagram of the ILF**

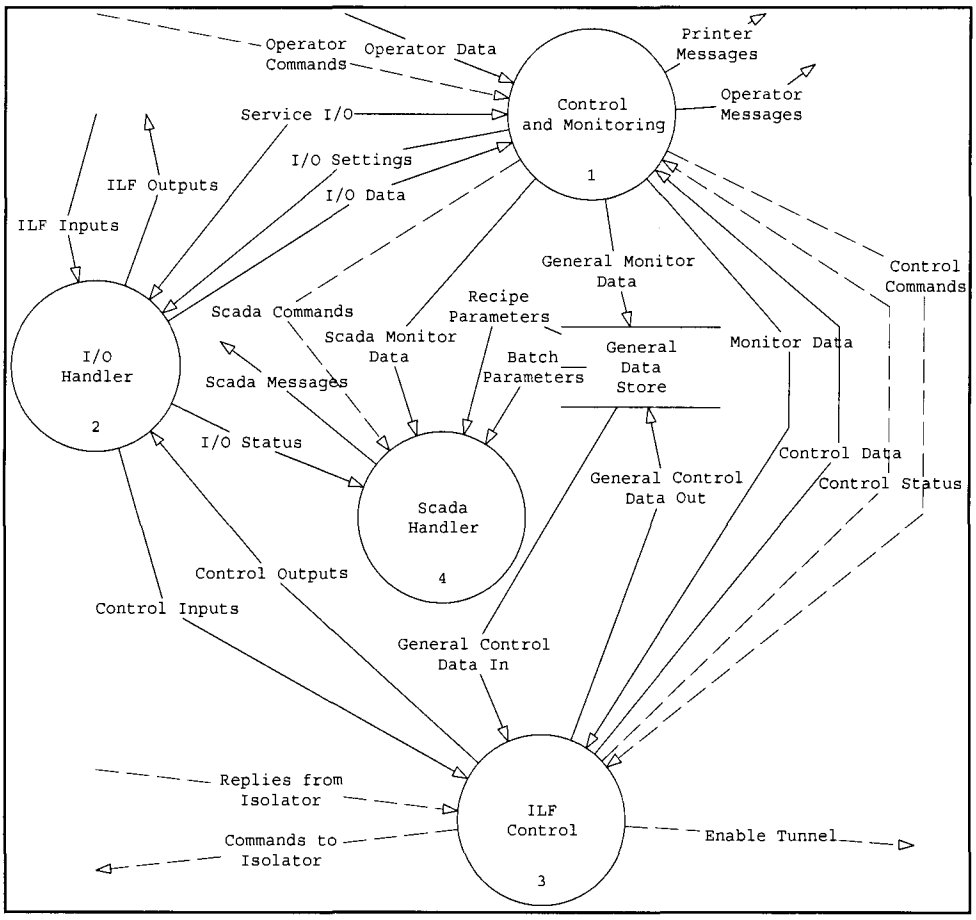


**Figure 2-8: Data Flow Diagram 0 of the ILF**

The processes running on the main controller can be divided into two groups. The first group contains the machine processes. These processes refer to the machine hardware. Each process takes care of controlling its corresponding part of the machine. The second group contains general processes. They are not directly participating in the machine control. These processes take care of interfacing with operators and communication between the different machine processes.

The machine processes are:

- General unit: the general unit takes care of general machine functions like cabin temperature, air pressure, power supply, safety control, etc. This process also controls the stack light, which indicates the current machine state.
- Infeed unit: the infeed unit is responsible for the control of incoming vials.
- Star unit: the star unit is responsible for the control and synchronization of the main transport and the vertical movement of the needle bridge in the filling section.
- Filler unit: the filler unit is responsible for the filling of the vials. It sets the parameters of the time-pressure process that is executed by the DSP. The DSP takes care of the actual filling process.
- Closing unit: the closing unit is responsible for the placement of the rubber stoppers onto the vials. It controls the height of the closing station and checks the presence of stoppers on the vials after the closing unit.
- Outfeed unit: the outfeed unit is responsible for the control of outgoing vials.
- Reject unit: the reject unit is responsible for rejecting of off specification vials.
- Tare unit: the tare unit handles the weighing of empty vials before they are filled. This includes taking the vials from and placing the vials back into the main transport.
- Gross unit: the gross unit handles the weighing of filled vials. This includes taking the vials from and placing vials back into the main transport.
- Isolator unit: the isolator unit controls the doors and glove ports of the isolator and communicates with the isolator.

The general units are:

- Main unit: the main program does not perform any actual control. Its purpose is to coordinate the other machine tasks, using mailboxes. It starts all other processes and takes care of communication between these tasks.
- Hmidisp unit: the hmidisp unit controls all actions from and to the HMI's. It uses the mmidrv unit for the actual communication with the HMI's.
- Mmidrv: the takes care of the communication with the HMI's.
- Alarm unit: the alarm unit takes care of situations in which the machine is not able to continue its normal operation. Alarm messages are sent to the HMI's and to the printer.
- Aifilter unit: the aifilter unit reads the analog inputs of the control system and filters the input if necessary. The values are placed in mailboxes, so that all other units can use them.
- Encoder unit: the encoder unit handles the vial tracking for all modules. It makes sure that the vial data mailbox contains the correct data for all pocket positions. It also checks if other processes process the data in the mailbox in time. If they don't, an alarm is generated.
- Language unit: the language unit is responsible for the generation of the correct text on the HMI's and on the printer.
- Printer unit: the printer unit takes care of sending messages to the printer.
- Sewcom unit: the sewcom unit is responsible for the communication with the SEW servo drivers.
- IBSman unit: the IBSman unit controls the Interbus-S field bus. All inputs and outputs that are connected to the field bus can be accessed via this unit.
- Hserver unit: the hserver unit is used to boot the DSP board.

The software on the DSP is written in C. There is no operating system running on the DSP. The M59 mezzanine card that is located on the DSP board is also equipped with a DSP. This DSP takes care of

reading the analog inputs for the time-pressure process and does some filtering on these inputs. In the newest version of the ILF, where the DSP board is replaced with a 68k board, the M59 mezzanine card still performs the same function. This 68k board is running the OS-9 operating system. Only one process runs on this board. This process calculates the filling profiles for each needle. The main controller board and the mezzanine cards on the board trigger the process.

## 2.4 Nett Weight Filler (WFA)

### 2.4.1 Introduction

The WFA is a filling machine that checks the weight of all vials that are being filled during the filling process. It consists of an infeed section, a filling section, closing section and an outfeed. The infeed section takes care of placing the vials in the main star wheel. Each filling cycle new empty vials are placed in the main star wheel and at the same time vials that were filled during the previous filling cycle are moved out of the main star wheel, past the closing unit to the outfeed. When the new vials are in place, an elevation system places them on so-called loadcells. These loadcells perform the weighing during the filling process. When the vials are standing on the loadcells a digital signal is sent to the loadcells to tare the loadcells. During the transportation phase the cutoff weights are sent to the loadcells via serial communication. Next the needle bridge is lowered and the filling is started. When the weight of the liquid in a vial on a loadcell reaches it cutoff value, it sends a digital signal to the control system and the filling is stopped for the corresponding needle. When the filling is stopped a check weigh is performed by the loadcell to check the actual weight of the liquid in the vial. The result of the check weighing is normally not equal to the cutoff value, because of the spouting of liquid into the vial during filling.

Once the vials are filled, they are moved past the closing section, where a rubber stopper is placed onto the vials. When they have passed the closing section, the vials are moved to the reject station. Vials that are out of tolerance or if the closing stopper is missing are moved to the reject tray. The other vials are moved to the outfeed. In Figure 2-9 you can see a sketch of the WFA.

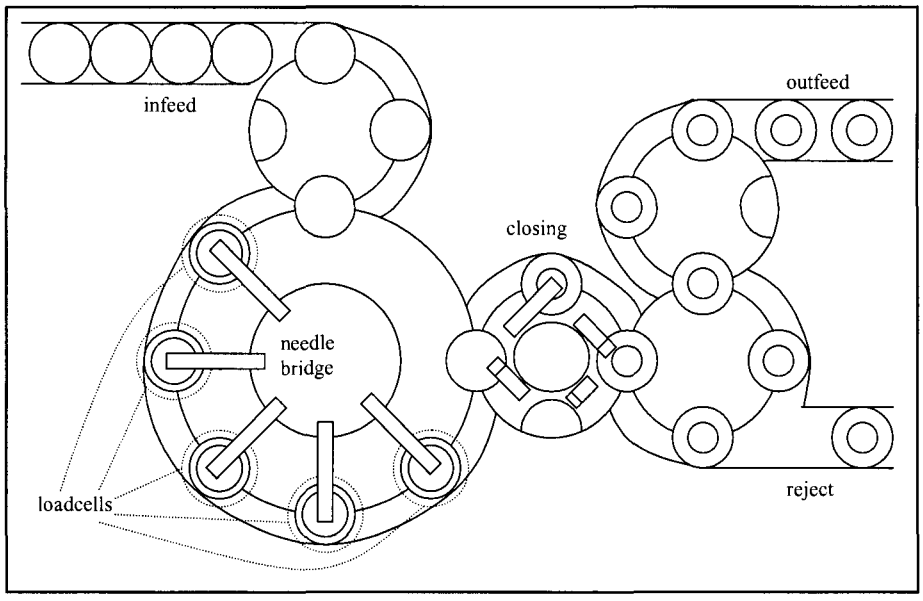


**Figure 2-9: Sketch of the WFA**

### 2.4.2 Control System

The control system of the WFA is based on a gespac system. This consists of a 19 inch rack with a gespac-bus back plane. Several single eurocard form factor cards can be placed in the rack. The main controller of this system is a 68030 from Motorola.

The WFA is not equipped with a field bus. All devices are connected directly to the gespac rack. The control system uses about 50 digital inputs, 70 digital outputs, 7 analog inputs, 4 analog outputs and 5 serial ports to communicate with all the devices on the WFA.

About half of the inputs of the control system have a rate of change that is in the range of the number of vials per minute that are filled. All other inputs have a lower rate of change. They are mainly used to monitor some process parameters or the status of parts of the machine. Changes on these inputs often result in an alarm. During startup and shutdown these inputs have a higher rate of change, but at the same time the other inputs will have a lower rate of change.

All loadcells are connected to an RS485 network, which in controlled by a serial port in the gespac rack. The RS485 network is used to write the cutoff values to and read the check measurements from loadcells. Two servo controllers are also connected to a serial port. This serial links are used to control the movement of the main transport and the movement of the needle bridge. Another serial port is used to control the HMI of the WFA. One serial port can be connected to an external modem. This modem can be used for maintenance. Once connected to the WFA and properly configured, it is possible to dial-in to the WFA. If there is a problem with the WFA an engineer from BOC Edwards can dial-in to the WFA and perform some checks without having to travel to the customer's site. A printer is connected to another serial port. The printer is used to print batch reports.

Each loadcell of the WFA is equipped with a DSP. This DSP continuously checks the weight during filling and tells the main control system to stop filling when the cutoff value is reached.

### 2.4.3 Software

The software design is done using the method of Hatley and Pirbhai. This method is used to identify the processes the control system has to execute and the communication between these processes.

In Figure 2-10 and Figure 2-11 you can see the first two diagrams that were created during the software design process. These are the context diagram and data flow diagram 0. During the software design process these diagrams were further detailed. The result is that the processes of the system are identified along with the communication between these processes.

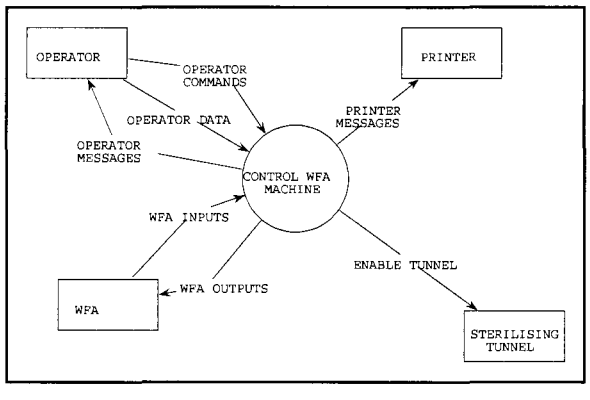


**Figure 2-10: Context Diagram of the WFA**

The operating system of the main controller of the WFA is OS-9 version 2.9 from Microware. Development is done on the target platform. The software development is done on the target. The compiler used is C compiler executive, also from Microware. The software is written in C, according to the Kernigan & Ritchie standard. Because development is done on the target, an editor that is running on the target is used. This editor is Umacs.

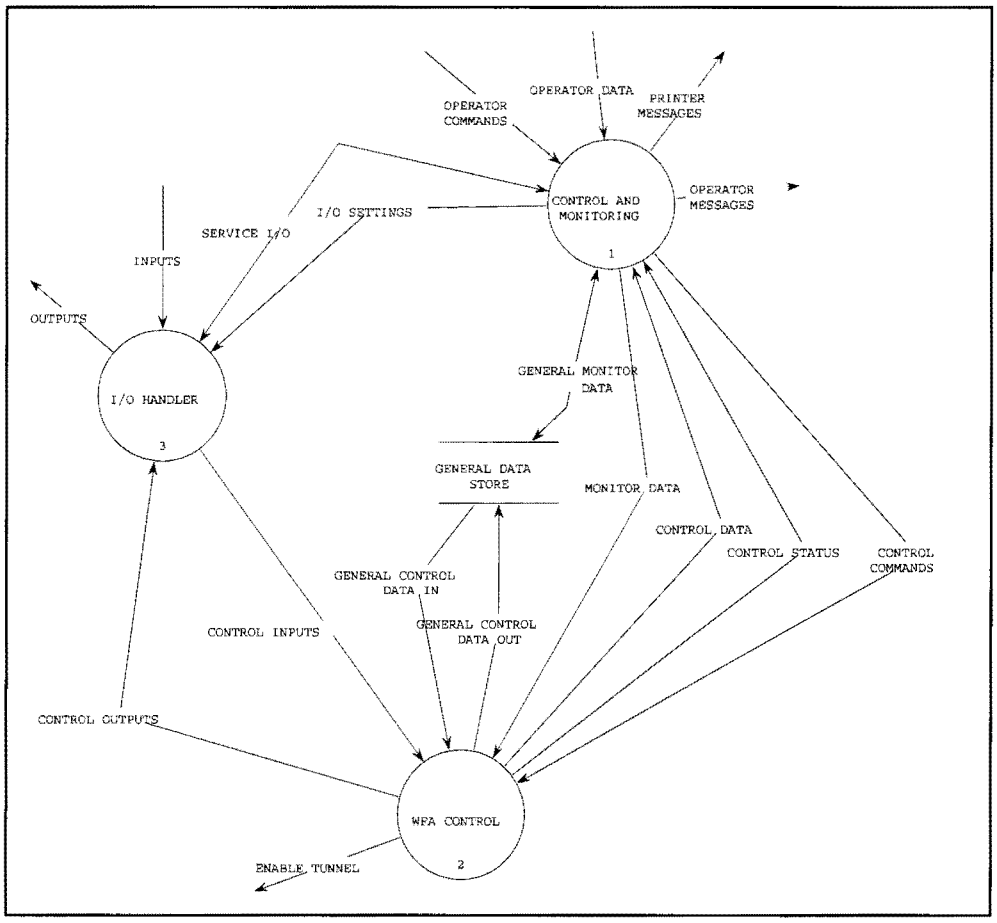


**Figure 2-11: Data Flow Diagram 0 of the WFA**

The scheduler in OS-9 is set to sequential scheduling. The created processes are enabled one after the other in a fixed order. There are only a few processes that are enabled on a priority base. These few processes have hard real-time constraints and require a fast response.

The processes running on the main controller can be divided into two groups. The first group contains the machine processes. These processes refer to the machine hardware. Each process takes care of controlling its corresponding part of the machine. The second group contains general processes. They are not directly participating in the machine control. These processes take care of interfacing with operators and communication between the different machine processes.

The machine processes are:

- General unit: the general unit takes care of general machine functions like cabin temperature, air pressure, power supply, safety control, etc. This process also controls the stack light, which indicates the current machine state.
- Infeed unit: the infeed unit is responsible for the control of incoming vials.
- Star unit: the star unit is responsible for the control and synchronization of the main transport and the vertical movement of the needle bridge in the filling section.
- Filler unit: the filler unit is responsible for the filling of the vials. It controls the vial lift for placing the vials on the loadcells and it controls the loadcells. It sets the cutoff values and reads the check weights.
- Prodtank unit: the prodtank unit is responsible for the product supply control. It controls the liquid level and pressure inside the product tank.

- Closing unit: the closing unit is responsible for the placement of the rubber stoppers onto the vials. It controls the height of the closing station and checks the presence of stoppers on the vials after the closing unit.
- Outfeed unit: the outfeed unit is responsible for the control of outgoing vials and for rejecting of off specification vials.

The general units are:

- Main unit: the main program does not perform any actual control. Its purpose is to coordinate the other machine tasks, using mailboxes. It starts all other processes and takes care of communication between these tasks.
- Hmidisp unit: the hmidisp unit controls all actions from and to the HMI's.
- Alarm unit: the alarm unit takes care of situations in which the machine is not able to continue its normal operation. Alarm messages are sent to the HMI's and to the printer.
- Aimuxada unit: the aimuxada unit reads the analog inputs of the control system and filters the input if necessary. The values are placed in mailboxes, so that all other units can use them.
- Encoder unit: the encoder unit handles the vial tracking for all modules. It makes sure that the vial data mailbox contains the correct data for all pocket positions. It also checks if other processes process the data in the mailbox in time. If they don't, an alarm is generated.
- Language unit: the language unit is responsible for the generation of the correct text on the HMI's and on the printer.
- Printer unit: the printer unit takes care of sending messages to the printer.

## 2.5 Non Contact Check Weigher (NCCW)

### 2.5.1 Introduction

At the moment that this inventory is made, the NCCW is still being developed. The purpose of the NCCW is to be able to use high speed filling machines like the ILF and still be able to check the filling of all vials. The weighing principal of the NCCW is based on Nuclear Magnetic Resonance (NMR). The basic principals of NMR are as follows. The substance that is being measured is placed in a strong magnetic field. During a short amount of time an RF pulse is transmitted to the substance. The result is that the vibration of the hydrogen atoms in the substance is affected. When the RF pulse is removed, the hydrogen atoms start to spin back to the state they had before the RF pulse. As a result the substance radiates at a given frequency. The amplitude of the signal is normative for the amount of the substance and thus for the weight of the substance. For each substance the system has different parameter values concerning measurements using the method of NMR.

Two versions of the NCCW are being developed. One is a stand-alone version that can check the weight of all passing vials and reject them if they are out of tolerance. The other version will be closely connected to the ILF. It runs at the same speed as the ILF and checks the weight of all filled vials. This version of the ILF does not have the check weighers that are on the standard version of the ILF. Based on the weigh of the substance in the vials, the NCCW sends correction data to the ILF. In this configuration the NCCW functions as a feedback for the ILF's filling process.

Besides the magnet and measurement probe the NCCW has an infeed, outfeed and reject tray. In Figure 2-12 you can see a sketch of the NCCW.

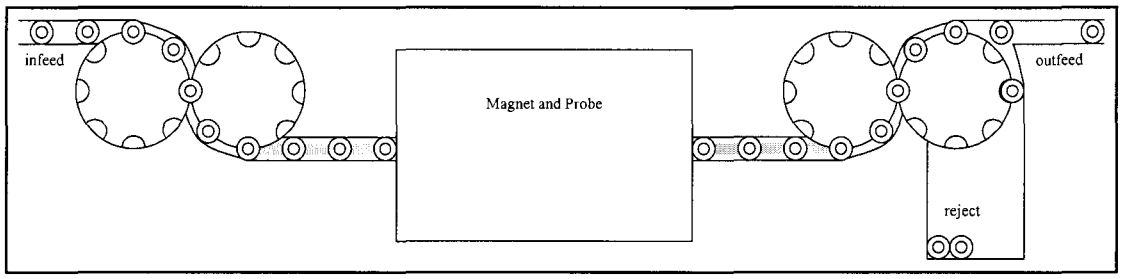


**Figure 2-12: Sketch of the NCCW**

### 2.5.2 Control system

The control system of the NCCW is, like the control system of the ILF, based on a VME rack. It looks much like the control system of the ILF. The same processor board is used as well as most of the interface cards are the same. When the NCCW is used together with the ILF, an Interbus-S slave interface is added to the NCCW's control system to communicate with the ILF. In addition to the Interbus-S connection, a pulse signal from the servos of the ILF is fed to the NCCW's servos. This signal is used for synchronization of the main transports of the two machines.

The functionality of the NMR equipment is not integrated with the control system of the NCCW. A separate acquisition computer is build into the electrical cabinet. This acquisition computer communicates with the control system via a serial connection. The actual measurement hardware, which is connected to the NMR probe, is outside the acquisition computer in a separate box.

The NCCW has about 25 digital inputs, 15 digital outputs, 3 analog inputs and 2 analog outputs. These inputs and outputs are connected to the control system via the Interbus-S. The control system is also equipped with some digital inputs and a motion counter, which are used for tracking of the vials on the NCCW and to get some information from the acquisition computer. The motion counter is connected to a digital output of the servo controller.

The NMR part of the NCCW, the magnet, probe and acquisition computer, are not designed by BOC Edwards itself, but by another company that is specialized in NMR measurements.

In Figure 2-13 and Table 2-3 you can see how the control system of the NCCW is configured. The configuration shown is the configuration of the NCCW that can be used in combination with the ILF.

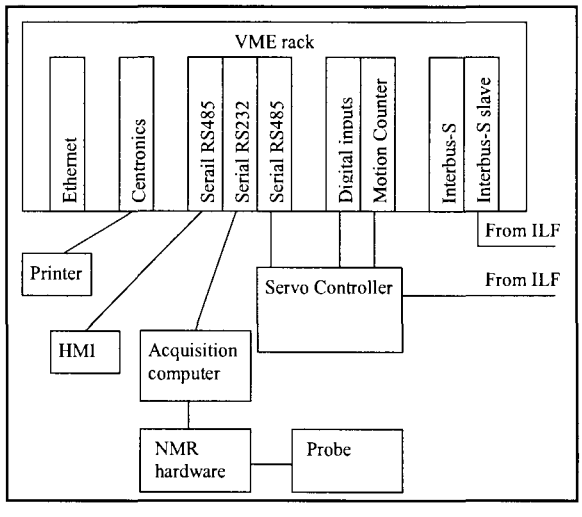


**Figure 2-13: Control System of the NCCW**

Table 2-3: The Parts of the Control System of the NCCW

| Part | Function | Location | Manufacturer |
|---|---|---|---|
| PG1226/00 | Rack | | Philips / Nyquist |
| PG2056/60 | Main Processor 68040 | Rack | Philips / Nyquist |
| M363 | Ethernet (for service) | PG2056/60 | Inducom |
| M52 | Centronics Interface | PG2056/60 | Men |
| PG2274 | SRAM disk | Rack | Philips / Nyquist |
| PG3984 | M-Carrier | Rack | Philips / Nyquist |
| M802 | Digital In | PG3984 | Philips / Nyquist |
| M806 | Serial In/Out | PG3984 | Philips / Nyquist |
| M39 | Interbus-S Slave | PG3984 | Men |
| M72 | Motion Counter | PG3984 | Men |
| IBS VME6H SC/I-T | Interbus-S master controller | Rack | Phoenix Contact |

### 2.5.3 Software

The software design is done using the method of Hatley and Pirbhai. This method is used to identify the processes the control system has to execute and the communication between these processes. In Figure 2-14 and Figure 2-15 you can see the first two diagrams that were created during the software design process. These are the context diagram and data flow diagram 0. During the software design process these diagrams were further detailed. The result is that the processes of the system are identified along with the communication between these processes. The operating system used on the main controller of the control system of the NCCW is OS-9 version 3.0.3 from Microware. It is compiled at a host PC running Microsoft Windows. The used compiler is a cross-compiler called Ultra C/C++, also from Microware. The NCCW is programmed mostly using standard C. In addition some objects are used for interaction with some of the devices on the NCCW. These objects are written in C++. The editor that is used for writing the code is Codewright, version 5.0c.

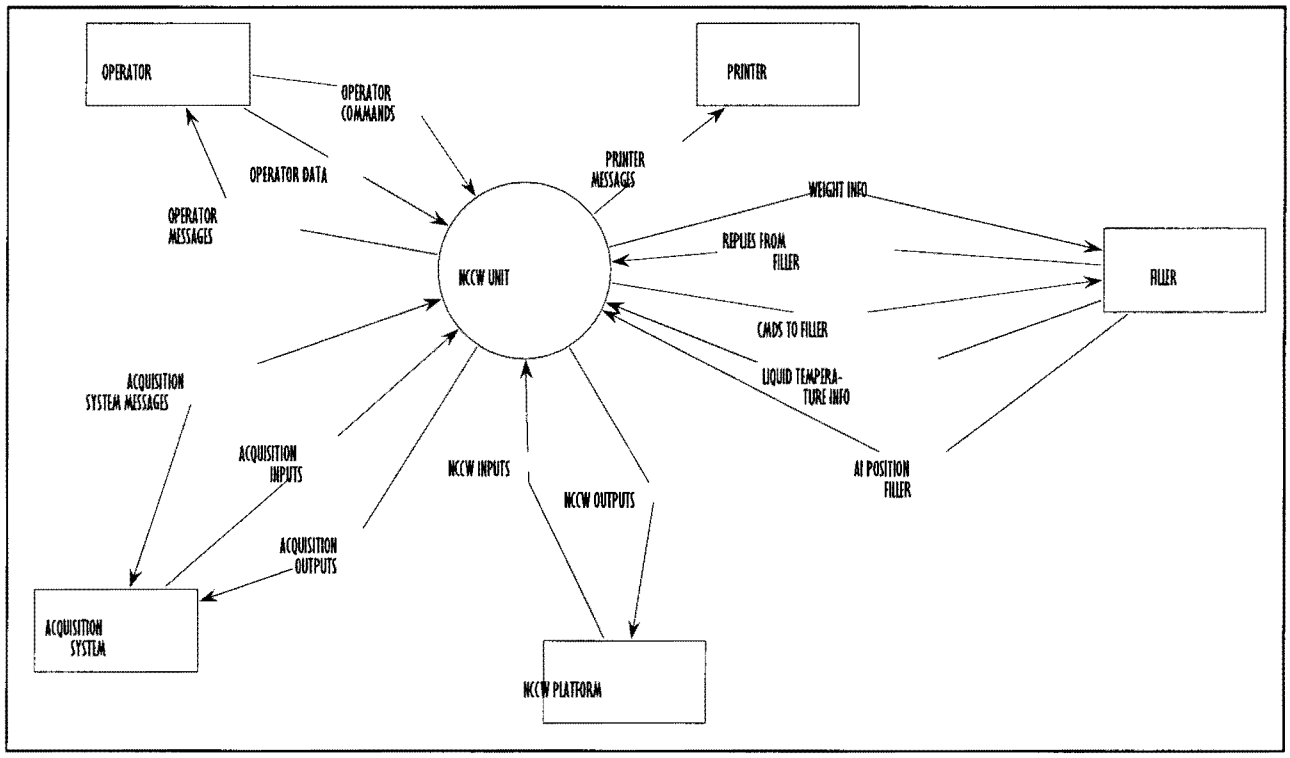


**Figure 2-14: Context Diagram of the NCCW**

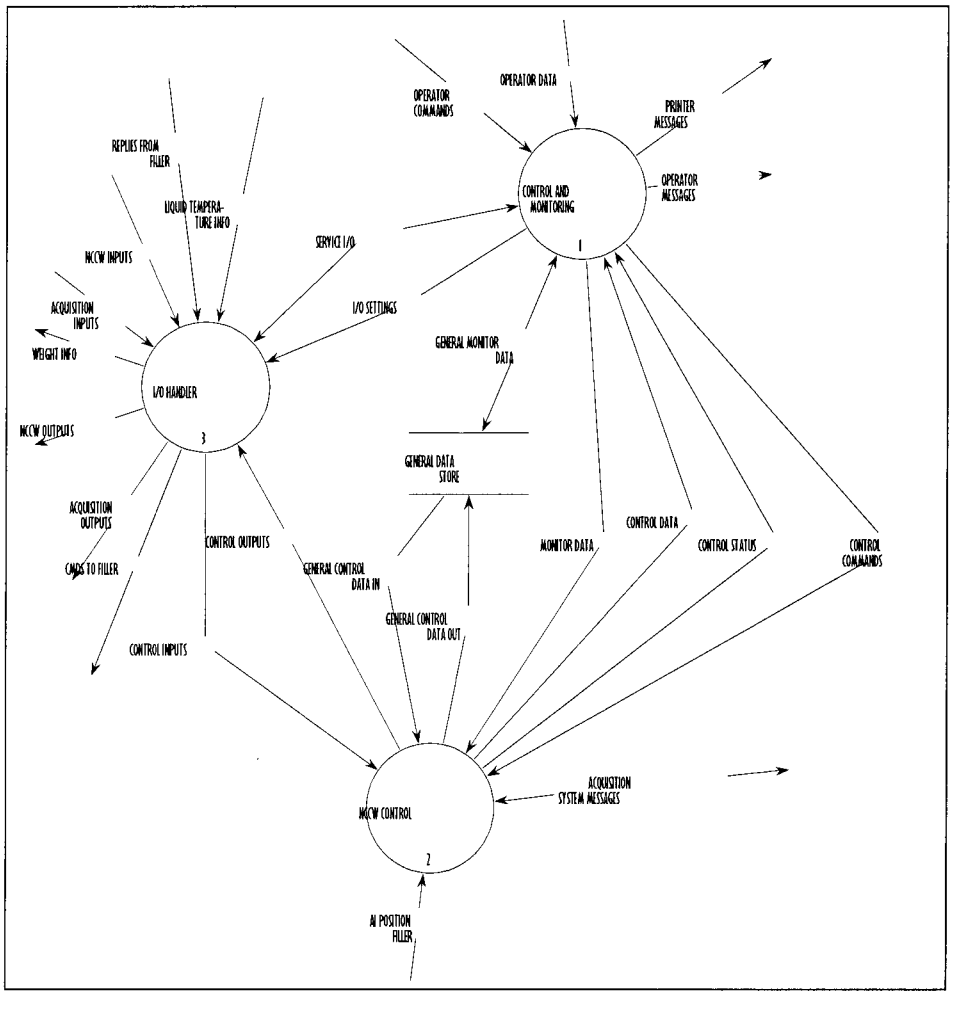


**Figure 2-15: Data Flow Diagram 0 of the NCCW**

The scheduler in OS-9 is set to sequential scheduling. The created processes are enabled one after the other in a fixed order. There are only a few processes that are enabled on a priority base. These few processes have hard real-time constraints and require a fast response.

The processes running on the main controller can be divided into two groups. The first group contains the machine processes. These processes refer to the machine hardware. Each process takes care of controlling its corresponding part of the machine. The second group contains general processes. They are not directly participating in the machine control. These processes take care of interfacing with operators and communication between the different machine processes.

The machine processes are:

- General unit: the general unit takes care of general machine functions like cabin temperature, air pressure, power supply, safety control, etc. This process also controls the stack light, which indicates the current machine state.
- Infeed unit: the infeed unit is responsible for the control of incoming vials.
- Transport unit: the transport unit is responsible for the movement of the main transport and synchronization with the preceding ILF.
- Interface unit: the interface unit is responsible for the communication with the preceding ILF. It also determines the liquid temperature based on the liquid temperature, glass temperature and environment temperature at the moment of filling.

- Outfeed unit: the outfeed unit is responsible for the control of outgoing vials.
- Reject unit: the reject unit is responsible for rejecting of off specification vials.
- IBSman unit: the IBSman unit controls the Interbus-S field bus. All sensors and actuators that are connected to the Interbus-S can be accessed through this unit.
- Measure unit: The measure unit handles the communication with the acquisition computer. It receives raw measurement values and calculates the nett weight based on this raw weight and some other environment parameters like temperature.

The general units are:

- Main unit: the main program does not perform any actual control. Its purpose is to coordinate the other machine tasks, using mailboxes. It starts all other processes and takes care of communication between these tasks.
- Hmidisp unit: the hmidisp unit controls all actions from and to the HMI's.
- Alarm unit: the alarm unit takes care of situations in which the machine is not able to continue its normal operation. Alarm messages are sent to the HMI's and to the printer.
- Encoder unit: the encoder unit handles the vial tracking for all modules. It makes sure that the vial data mailbox contains the correct data for all pocket positions. It also checks if other processes process the data in the mailbox in time. If they don't, an alarm is generated.
- Language unit: the language unit is responsible for the generation of the correct text on the HMI's and on the printer.
- Printer unit: the printer unit takes care of sending messages to the printer.
- Sewcom unit: the sewcom unit is responsible for the communication with the SEW servo drivers.

## 2.6 Loading and Unloading Systems

### 2.6.1 Introduction

When vials need to undergo a freeze-dry process after they are filled, they need to be placed into a Freeze Dryer for some time. The filling of vials is a continuous or semi-continuous process. Freeze drying on the other hand is a batch process. A freeze drying process can be as long as about one week. Of course the time a batch lasts depends on the type of product and volume in the vials. In order to be able to effectively switch between a continues and a batch process there needs to be the capability to buffer vials. This is done by a Loading Accumulation Table (LAT).

After the vials have been in a Freeze Dryer, a cap needs to be mounted onto them. This, again, is a continuous process. So, after the vials have been in a Freeze Dryer, they need to be buffered again and placed in a single line in order to go to a capper. When the vials are taken out of a Freeze Dryer, they can be placed in an Unloading Accumulation Buffer (UAB) or on an Unloading Accumulation Table (UAT).

A loading and unloading system normally consists of one or more Loading Accumulation Tables, One or more Freeze Dryers, an Unloading Accumulation Buffer or an Unloading Accumulation Table or a combination of an Unloading Accumulation Buffer and an Unloading Accumulation Table. It is also possible to combine a Loading Accumulation Table and an Unloading Accumulation Table in one Loading and Unloading Accumulation Table (LUAT). Other combinations can also be made on customer request. No two loading and unloading systems are the same. Every customers has his specific wishes for the configuration.

Transportation between the loading and unloading tables and buffers and the Freeze Dryers is performed with one or two Tansporters (TCAR). A TCAR can load or unload a complete shelf of vials at a time. It moves on two guide rails. Communication with fixed systems is done via an optical link. In Figure 2-16 you can see an example of a possible configuration of a loading and unloading system.

Vials arrive at a Loading Accumulation Table in a single line. They are fed onto the table, until the length of the line on the table is equal to the width of the table. Then infeed is stopped and a bar pushes the vials onto the table. When the bar is returned, the next line of vials is fed onto the table. This is repeated until the table is full. Then a TCAR will pick up all vials that are on the table and place them on a shelf in a Freeze Dryer.

When a Freeze Dryer is ready with a freeze dry batch, a TCAR picks up the vials from one shelf at a time. The TCAR delivers the vials to an Unloading Accumulation Table, an Unloading Accumulation Buffer or a Loading and Unloading Accumulation Table.

When a packed of vials is placed on an Unloading Accumulation Table or a Loading and Unloading Accumulation table, the table starts to move the vials to a single liner that is attached to the table. The vials are then transported to a Capper in a single line.

When the vials are placed in an Unloading Accumulation Buffer and all vial are in the buffer, the buffer starts to empty itself. This is done one shelf at a time. A shelf is placed at the elevation of the outfeed. The outfeed consists of a single liner. The vials are moved from the shelf to the single liner. When the shelf is empty, the next shelf is moved to the outfeed level and emptied via the single liner.

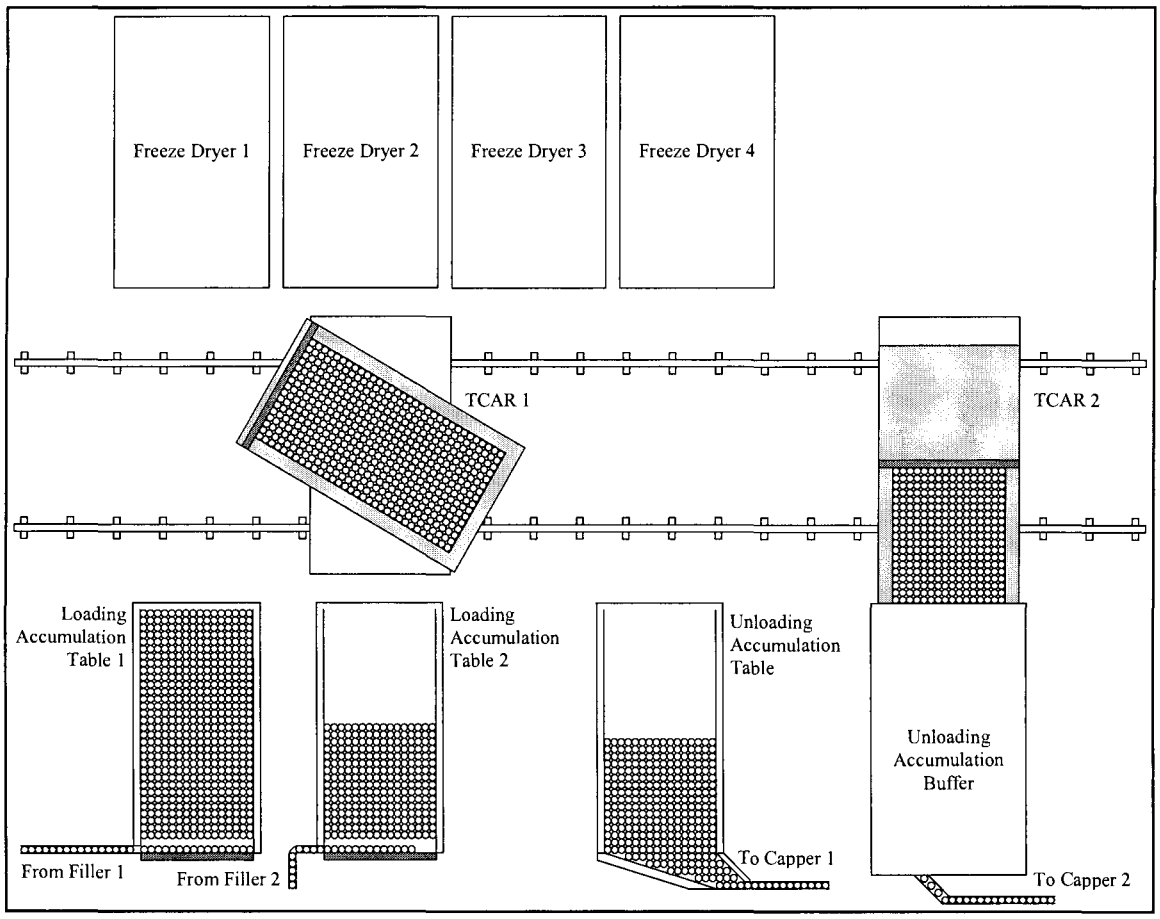


**Figure 2-16: Example of a Loading and Unloading System**

### 2.6.2 Control System

All machines in a loading and unloading system have their own control system. Every LAT, LUAT, TCAR, UAB, UAT and Freeze Dryer has a separate control system. They are all equipped with a PLC. The brand of PLC is customer dependent. These systems have been equipped with Allen-Bradley, Siemens, Modicon or Mitsubishi PLC's. The field buses used on the loading and unloading systems are related to the PLC that is being used. This is also the case with the network that is used for communication between the PLC's of the separate systems and with the used SCADA system. In Table 2-4 you can see the different combinations,

depending on the brand of PLC.

Table 2-4: Control System Configurations on Loading and Unloading Systems

| PLC manufacturer | Type of PLC | Programming software | Field bus | Network |
|---|---|---|---|---|
| Allen-Bradley | SLC5/03 | RSLogix 500 | Device Net | Ethernet |
| Siemens | S7-300 | Step 7 | Profibus | Ethernet |
| Mitsubishi | Melsec ans | GppWin | SSC-NET* | Melsecnet |
| Modicon | Quantum CPU 43412 | Pro Worx NxT | Profibus | Modbus |

* SSC-NET is not really a field bus. It is only used to communicate with the servo-controllers. CC-Link is the preferred field bus for Mitsubishi PLC's.

Almost all of the loading and unloading machines are equipped with a field bus and digital and analog inputs and outputs in the PLC rack. The field bus is used to communicate with the servo controllers and some of the digital and analog inputs and outputs of the machine. All other sensors and actuators are directly connected to the PLC's inputs and outputs. In Figure 2-17 you can see a typical configuration of a loading or unloading machine's control system.

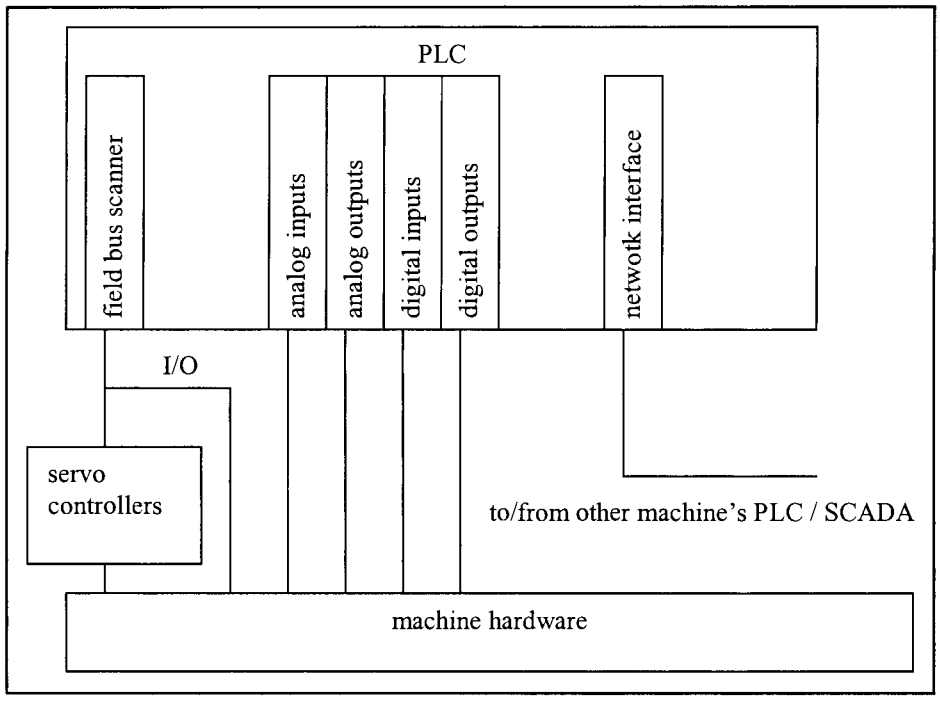


Figure 2-17: Typical Loading or Unloading Machine's Control System

shows the amount of digital and analog inputs and outputs that are used on the different loading and unloading machines. These numbers are indications, not exact counts. The number of inputs and outputs differs slightly from one project to the other.

Table 2-5: Number of Inputs and Outputs on Loading and Unloading Systems

| Machine | Digital Inputs | Digital Outputs | Analog Inputs | Analog Outputs |
|---|---|---|---|---|
| LAT | 25 | 20 | x | 2 |
| UAB | 25 | 35 | 1 | 10 |
| LUAT | 40 | 35 | 1 | 15 |
| TCAR | 20 | 15 | 1 | x |
| Master | 5 | 1 | x | x |
| UAT | 15 | 20 | 1 | 6 |

### 2.6.3 Software

The software design of all loading and unloading systems is done using the method of Hatley and Pirbhai. This method is used to identify the processes the control system has to execute and the communication between these processes. All systems are developed separately. Besides the loading and unloading machines, there is a master controller that takes care of overall system parameters like the temperature in the electrical cabinet. The separate machines are all controlled from a central SCADA system. The machines can communicate with each other. This is normally the case between the loading or unloading tables, buffers or Freeze Dryers at one side and the TCAR at the other side. The TCAR has to communicate with the machine it is docking with.

In Figure 2-18 to Figure 2-29 you can see the Context Diagrams and Data Flow Diagram 0's of the separate loading and unloading machines. During the software design process these Diagrams are further refined in order to identify all processes and the communication between these processes. The different diagrams may be inconsistent with each other, because they are taken from different projects.

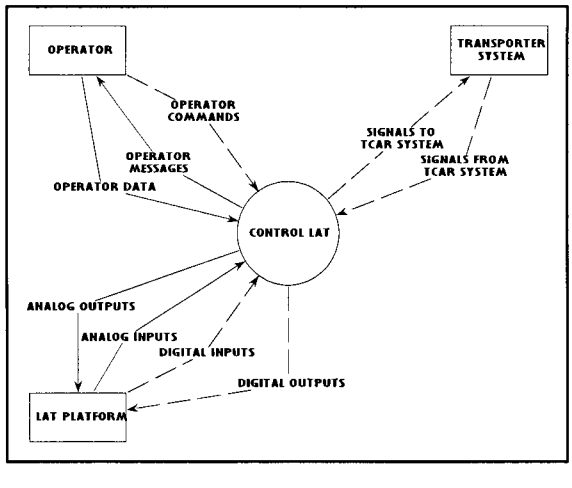


Figure 2-18: Context Diagram of the LAT

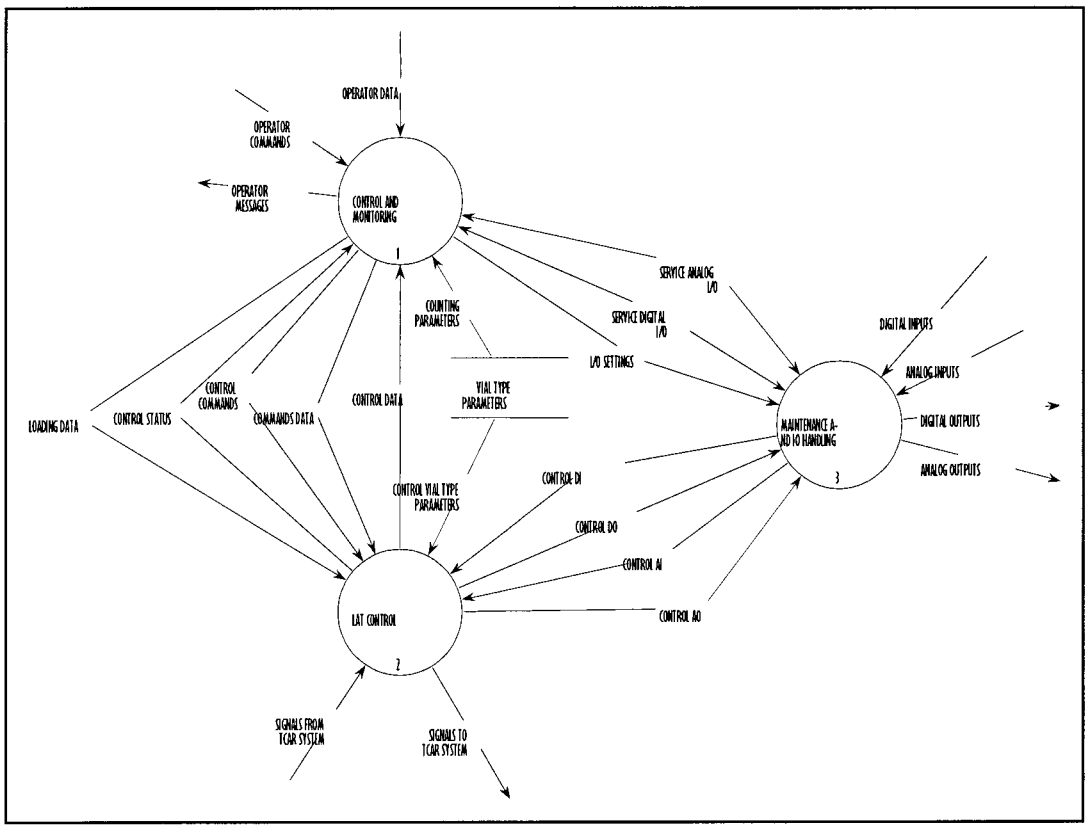


**Figure 2-19: Data Flow Diagram 0 of the LAT**

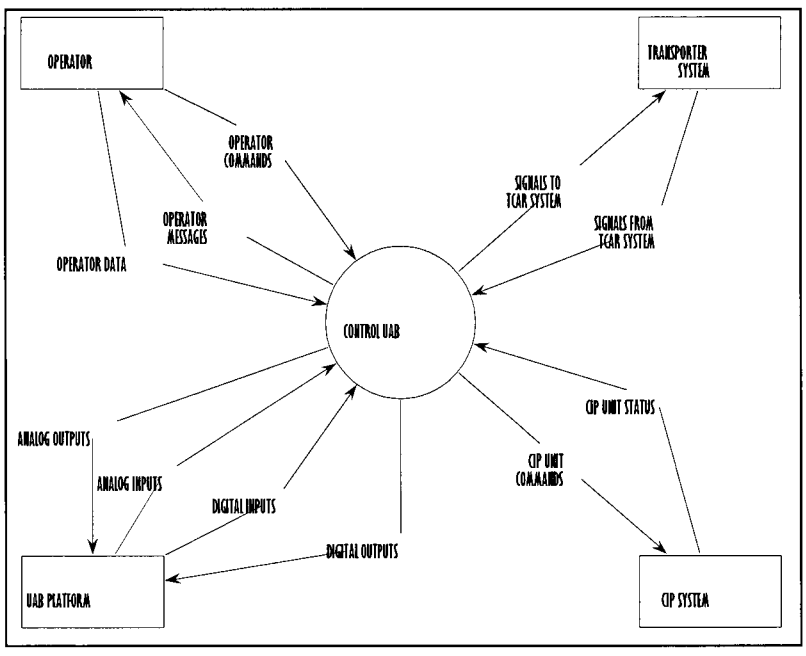


**Figure 2-20: Context Diagram of the UAB**

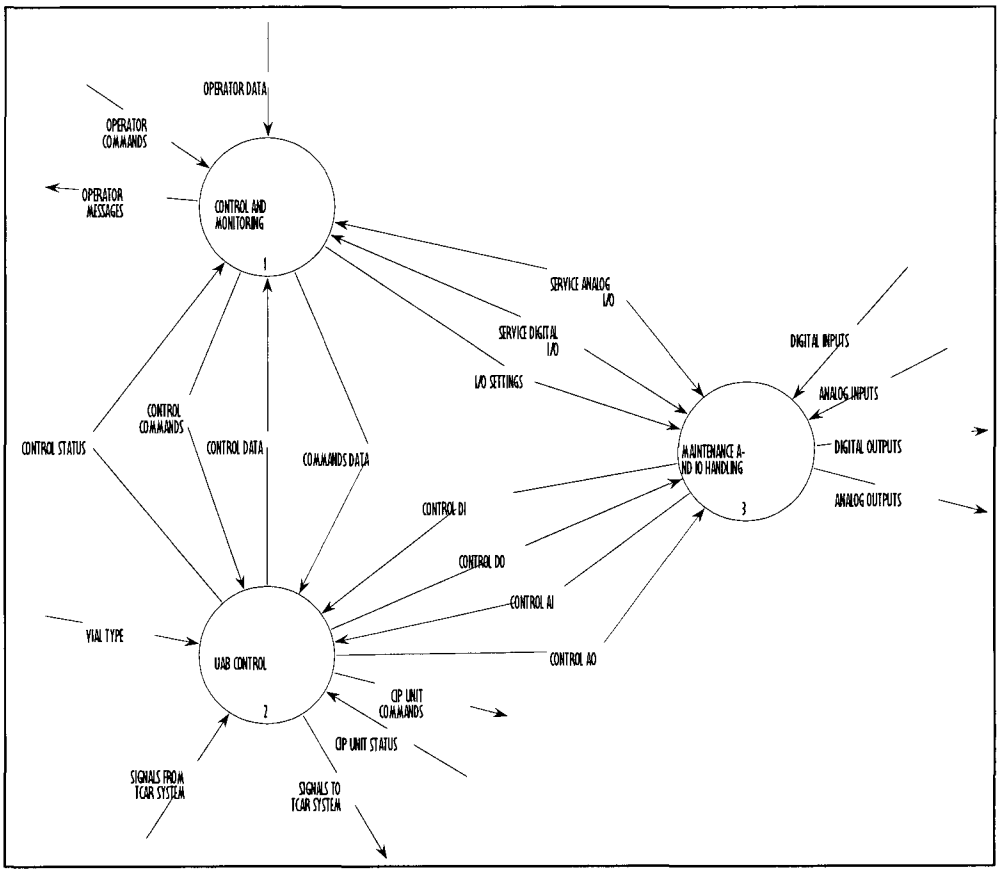


Figure 2-21: Data Flow Diagram 0 of the UAB

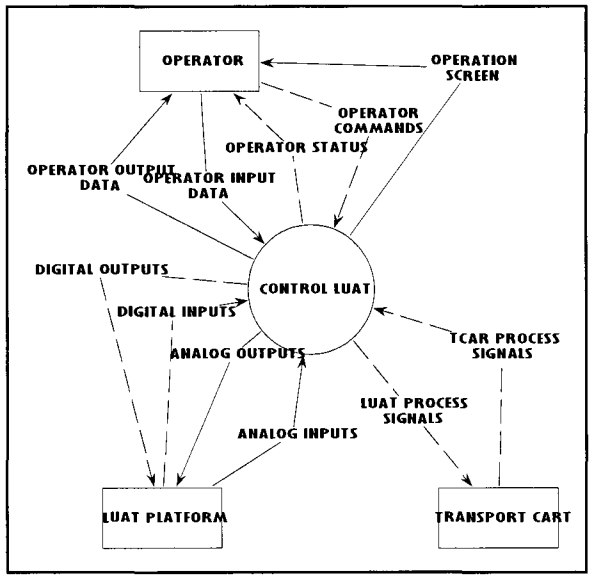


Figure 2-22: Context Diagram of the LUAT

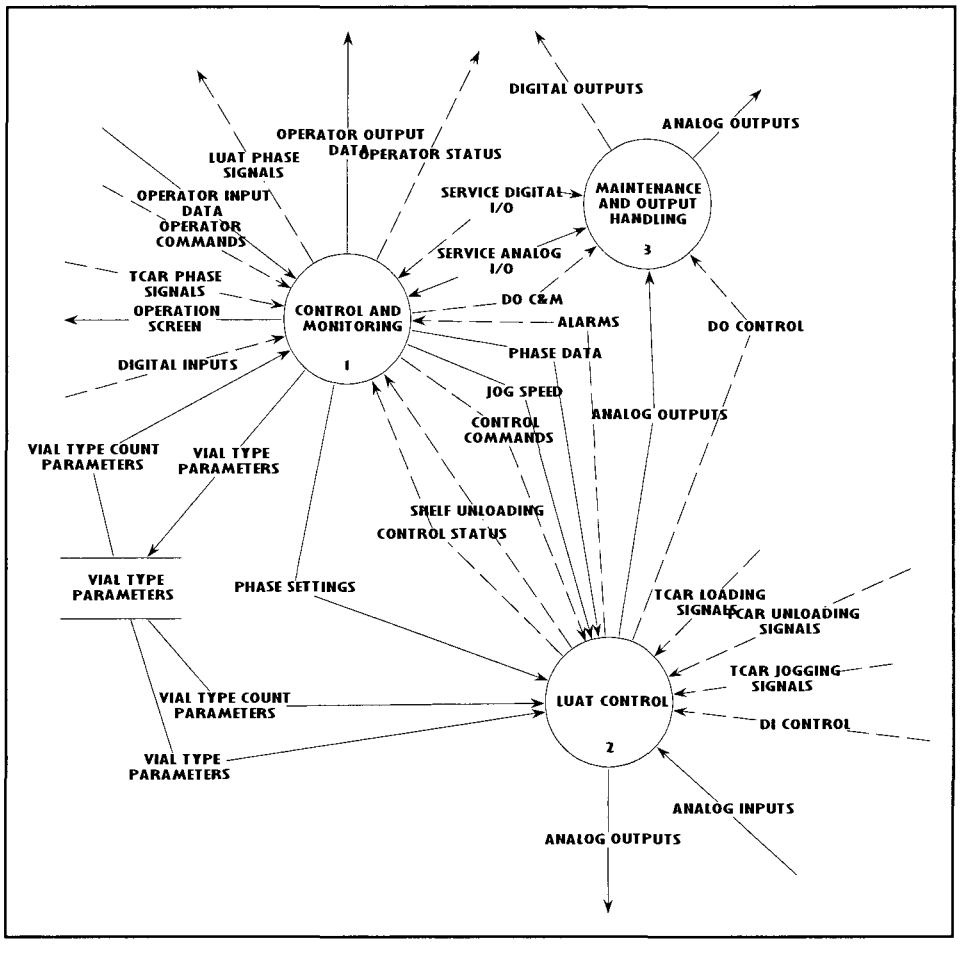


Figure 2-23: Data Flow Diagram 0 of the LUAT

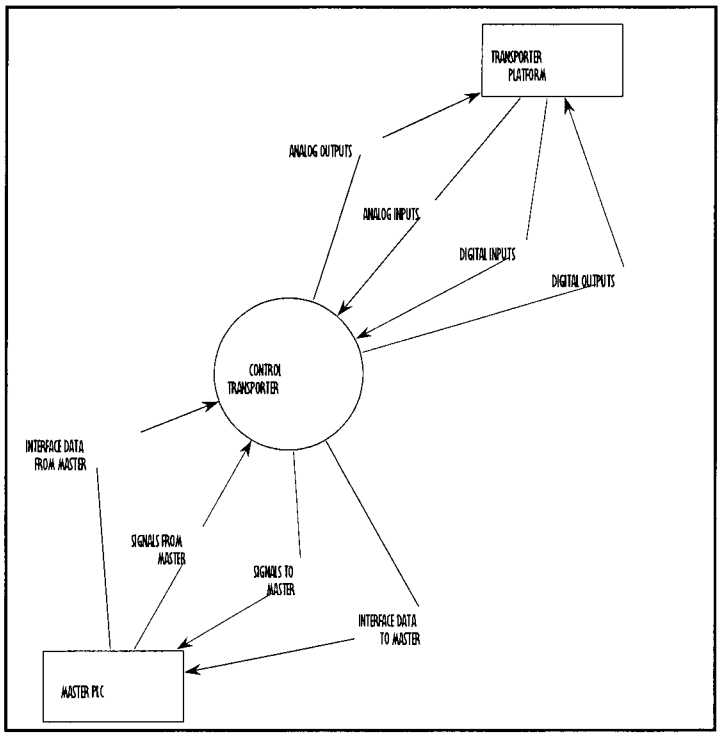


**Figure 2-24: Context Diagram of the TCAR**

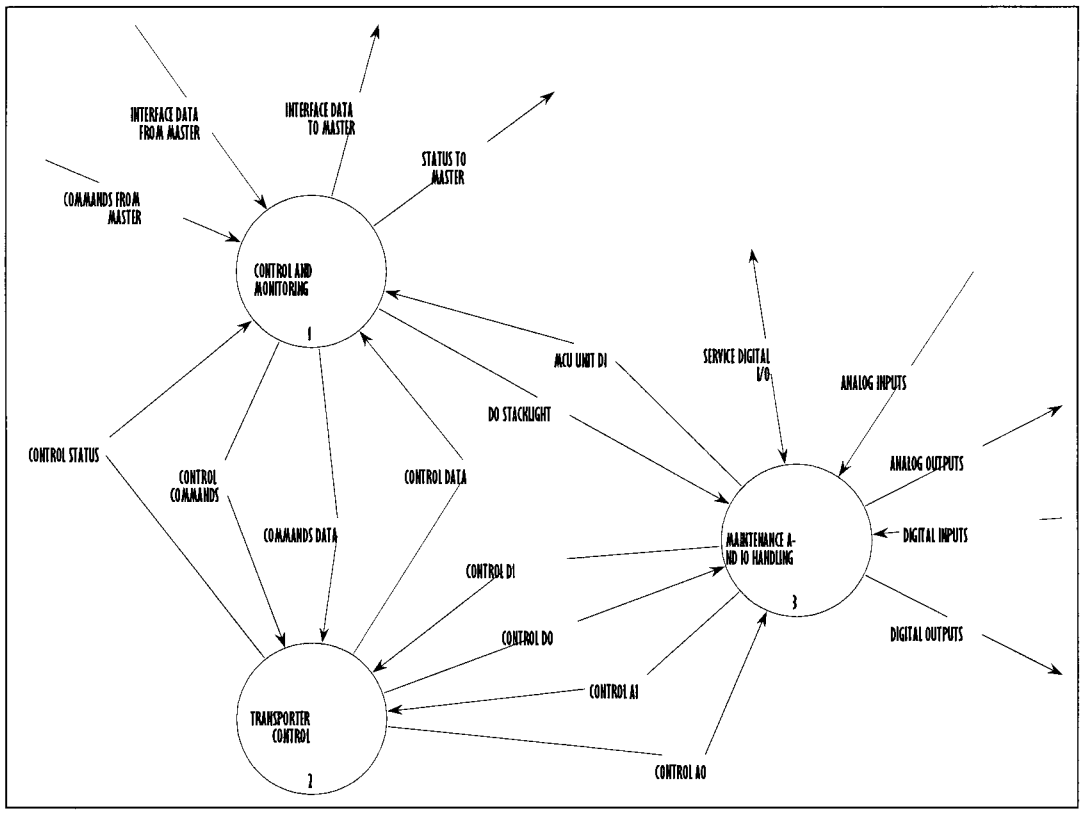


**Figure 2-25: Data Flow Diagram 0 of the TCAR**

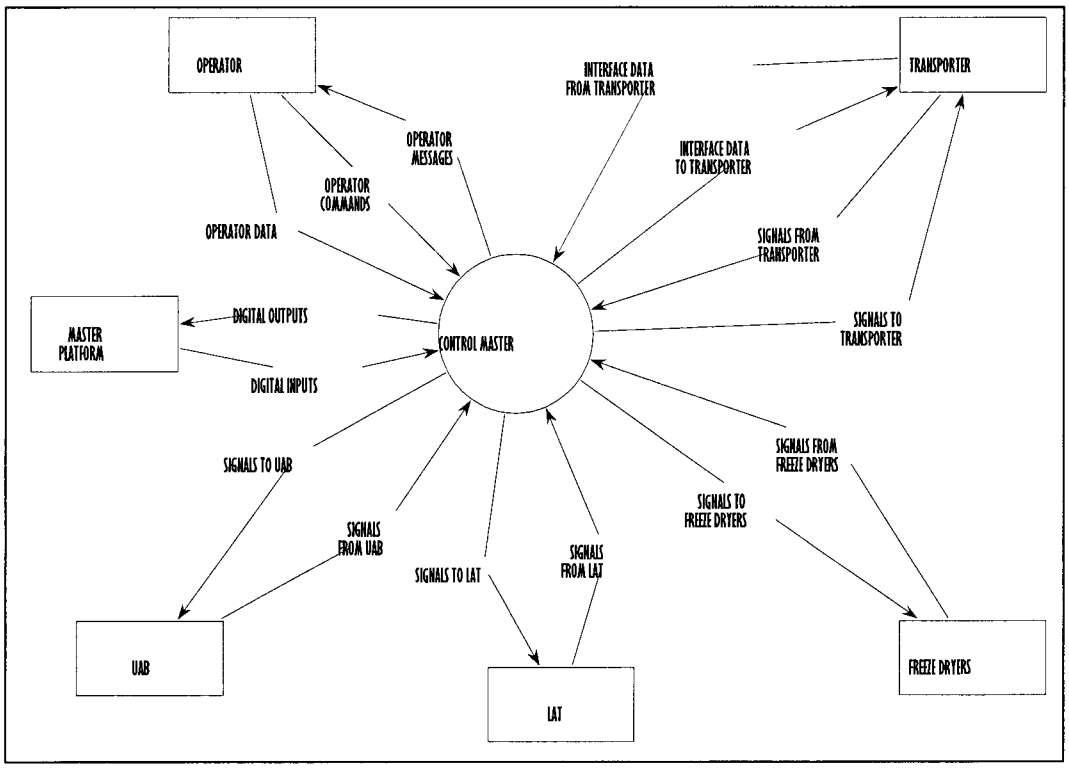


**Figure 2-26: Context Diagram of the Master PLC**

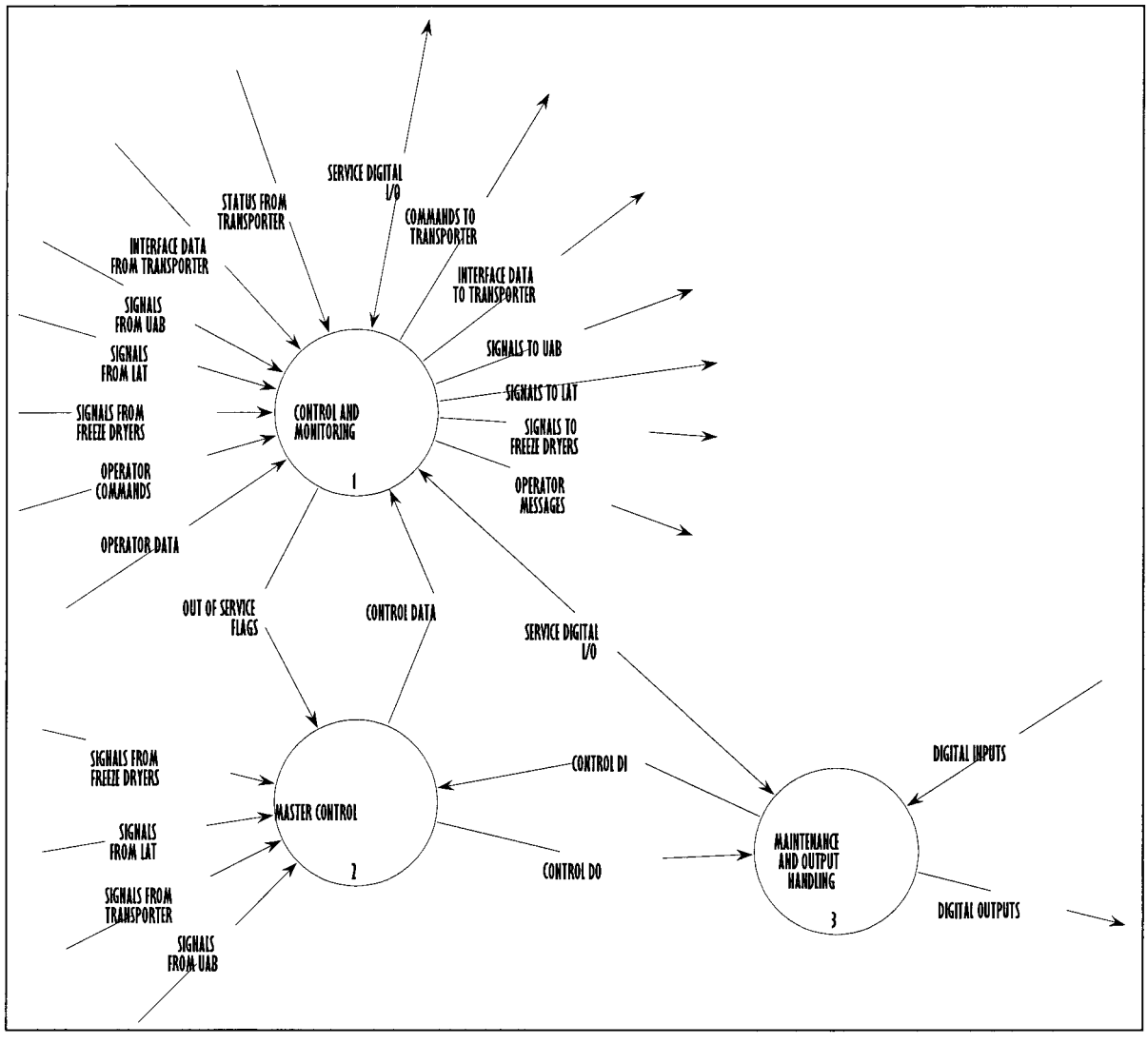


**Figure 2-27: Data Flow Diagram 0 of the Master PLC**

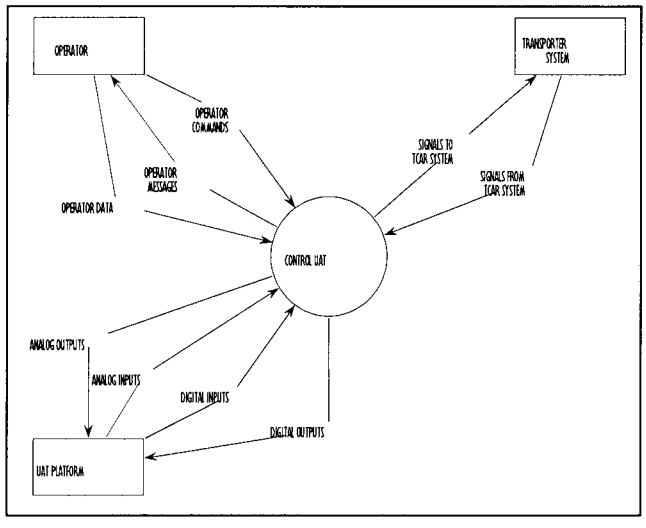


**Figure 2-28: Context Diagram of the UAT**

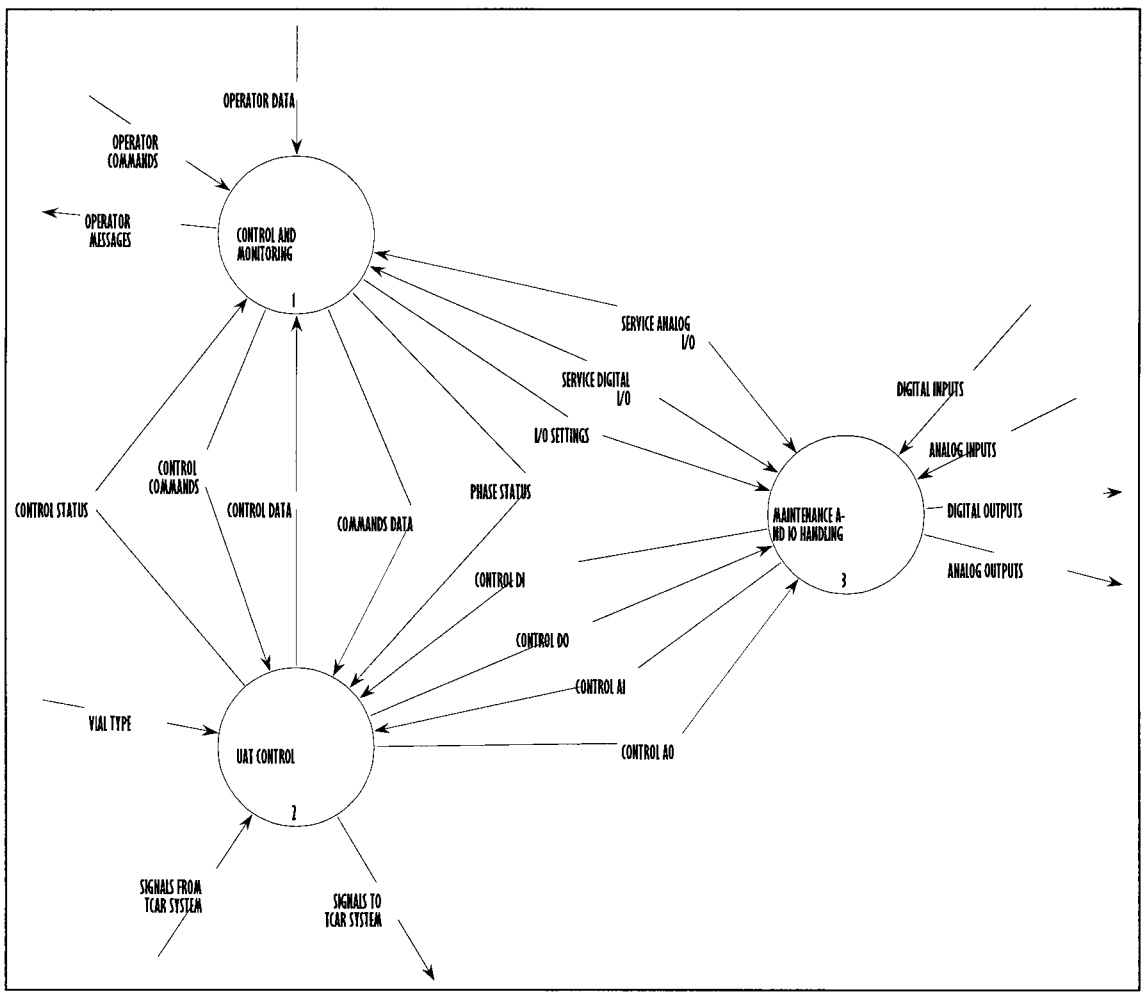


**Figure 2-29: Data Flow Diagram 0 of the UAT**

The software in the loading and unloading machines is divided in different units. Each unit has a specific function. Some units are related to a part of the machine and are responsible for the correct behavior of that specific part of the machine. Other units have a more general function and take care of the overall behavior of the machine or take care of the communication between other units. The software units per machine are:

- LAT
  - General unit: the general unit is responsible for all general machine functions. It takes care of the power supply, safety control, etc.
  - Infeed unit: the infeed unit is responsible for feeding rows of vials on the pusher conveyor.
  - Pushing unit: the pushing unit is responsible for pushing the rows of vials, created by the infeed unit, onto the table, thereby creating a vial pack.
  - Unloading unit: the unloading unit is responsible for the communication with the docked transporter. It controls the docking and unloading sequence signaling with the docked transporter.
- UAB
  - General unit: the general unit is responsible for all general machine functions. It takes care of the power supply, safety control, etc.
  - Indexing unit: the indexing unit controls the up and down movement of the shelves to the correct loading, unloading or draining position.
  - Loading unit: the loading unit is responsible for the communication with the docked transporter. It controls the docking and loading sequence signaling with the docked transporter.
  - Pusher unit: the pusher unit controls the unloading of vials via a pusher bar and the rear door.
  - Outfeed unit: the outfeed unit controls the single lining of the vials.
- LUAT
  - General unit: the general unit is responsible for all general machine functions. It takes care of the power supply, safety control, etc.
  - Infeed unit: the infeed unit is responsible for feeding rows of vials on the pusher conveyor.
  - Loading unit: the loading unit is responsible for the communication with the docked transporter. It controls the docking and loading sequence signaling with the docked transporter.
  - Single lining and accumulation unit: the single lining and accumulation unit is responsible for the unloading of the vials. It controls the single line section.
- TCAR
  - General unit: the general unit is responsible for all general machine functions. It takes care of the power supply, safety control, etc.
  - Transport unit: the transport unit is responsible for moving the transporter to the required station and for rotating the upper part to the correct position.
  - Docking/Undocking unit: the docking/undocking unit is responsible for providing an interface with the vial platform of any of the other machines.
  - Loading/Unloading unit: the loading/unloading unit is responsible for transferring the vial packs between the transporter and the shelves of the other machine the transporter is docked to.
  - Master
  - General unit: the general unit is responsible for all general machine functions. It takes care of the power supply, safety control, etc.
- UAT

- General unit: the general unit is responsible for all general machine functions. It takes care of the power supply, safety control, etc.
- Loading unit: the loading unit is responsible for communication with the docked transporter. It controls the docking and unloading sequence signaling with the docked transporter.
- Outfeed unit: the outfeed unit controls the single lining of the vials.
- Guarding unit: the guarding unit monitors the status of the machine guarding system.

## 2.7 Summary and Conclusions

BOC Edwards PS produces a wide range of machines. The control systems of the various machines very a lot. The most simple machine can be controlled by relays and the most complex machines are equipped with control system that is based on an industrial computer that is build around a VME rack. A real-time operating system, OS-9, is running on this industrial computer.

### 2.7.1 Software design method

The software design method used is the same for all machines. This method is the structured analysis method that is developed by Hatley and Pirbhai. [STR RTS] This method is used for the design of the software of the most simple through the most complex control systems. The software package *select yourdon* is the tool that is used to produce the necessary diagrams.

### 2.7.2 Control systems

Together with the great variance in complexity of the various machines also the control systems of the machines vary a lot. The most simple machines can be controlled by relays. If this is the case, the machines can not be integrated in SCADA systems. When these simple machines need to be integrated with a customers SCASA system, the relay based control system is replaced by a PLC.

Most machines are controlled by a PLC. The customer almost always determines the brand and type of PLC that is used on a machine. As a result, the control system of the machines has to be developed more than once. Each brand and type of PLC has its own programming environment and instruction set. Thus code for one PLC can not be used on another type of PLC.

The most complex machines are equipped with a control system that is based on an industrial computer. It is running a real-time operating system, OS-9. These control systems have a higher performance and greater flexibility than PLC. This makes them more usable for complex machines than PLC's. A disadvantage of the control systems that are based on an industrial computer is that there is no standard way for integrating them with existing SCADA systems.

## 3. THINGS THAT CAN BE IMPROVED

The use of a wide variety of control systems, the use of hardware that is getting old and the use of development methods that are being replaced be newer standards are candidates for improvements. A combination of these possible problem areas results in difficulties with integration aspects.

### 3.1 The software development method

The software design method developed by Hatley and Pirbhai has its origin in 1987. [STR RTS] Thus it is a rather old method. The method is very well suited for designing software that is developed in a procedural way with a lot of parallel processes. This matches very closely with the way software is designed for PLC's. PLC's do not really run processes parallel, but to the outside world it looks like they do. In practice a PLC processes its tasks in a cyclic manner. Each task can be seen as a process.

When the software design method developed by Hatley and Pirbhai is used for developing the software for computer based control systems some problems arise. The method designs the software in a way that is produces a lot of parallel processes. Together with the number of processes created an even greater number of communication paths between the processes is created. When writing the software according to the outcome of the software development method, the software would be build as a lot of small processes and a lot of communication channels between them. This would result in control system with a low performance. The operating system would have to take a lot of processing time just to schedule all processes and synchronise the communication channels. Thus a compromise had to be found. This is done be integrating some related, small processes, that came out of the software development phase, into one larger process. This reduces the number of processes and if the processes are combined in a smart way also the number of communication paths.

In the 1990's another type of software development methods quickly gained a lot of acceptance and usage in the market. This new type of software development works in an object-oriented way. Several independent methods were developed. [RT SS] Later on three scientists, Ivar Jacobson, Grady Booch and James Rumbaugh, discovered that all methods had its weaknesses and plus points. They started with some methods that gained a significant market share and had become accepted by the market. They took the strong points of these methods and tried to replace the weak parts with parts from the other methods. The result was the origination of the Unified Modelling Language (UML). [UML SDP] [UML RM] [UML UG] Currently this is by far the most used object-oriented software development method in the world. It is seen as a standard for object-oriented software development.

Thus UML is a much newer method then Hatley and Pirbhai, but is has already gained significant acceptance and implementation in the market. Changing to another software development method would have big impact. Because it is desirable that all software design is done with the same method, the software design for all types of control systems would have to change. On the other hand, sticking with an old method is not advisable. No new development is done for these methods and no new software packages that support the Hatley and Pirbhai method are expected to be designed.

#### 3.1.1 The use of UML

When developing software using UML the result is completely different than the results of the Hatley and Pirbhai method. Where the result of the Hatley and Pirbhai method is a procedural and process based design, the result of the UML method is an object-based design. The UML design defines an object for all objects in the real world and objects that perform the internal operations of the control system. Besides the objects, the method also describes the relations between the objects and the way the objects communicate with each other.

##### 3.1.1.1 Using UML for real-time operating system based software

In order to develop software that closely matches the result of the software design phase when this is done using UML, the software has to be implemented in an software language that supports the design of object-oriented software. Examples of this are C++, Java, etc.. Because the software for the latest developped

machines that are based on a real-time operating system, the ILF and the NCCW, are already programmed in C++, C++ would be a good choice for the programming language to use when UML would be applied.

The use of UML for control systems that are based on real-time operating systems could improve the relation between the final software and the software design. UML matches closely with the outside world and if the design phase is done properly, the writing of the software is restricted by the design. There should be little flexibility left over after the software design phase. The only thing that has freedom of implementation is the actual implementation of the functions that are embedded in the created objects.

There are several development tools on the market for implementing UML in the design process. [UML MT] These tools provide an interface for creating the various UML diagrams. Most of the available tools also have a feature to create the definitions of the classes that are used to represent the objects. The tools generate a piece of source code, where only the implementation of the functions in the objects needs to be filled in. When these features are used, the programming is restricted to implementing the functions of the objects according to the result of the software development phase.

A disadvantage of UML is that is does not provide a mechanism for splitting the control system in several parallel processes. The coding could result in one large executable, which is not preferable. Splitting the functionality of the control system into more processes is preferable for testing and debugging of the software. Different parts of the software can be tested independently. When using UML, different processes can be made by combining objects that have a lot of communication lines between them and together perform a certain task within the control system.

#### 3.1.1.2 Using UML for PLC's

The way PLC's are programmed does not seem to match with an object-oriented software design method. PLC programs look like parallel processes that perform certain tasks and communicate with each other. There is little visible relationship with real world objects.

From this is seems that UML is not very well suited for developing software for PLC's. The problem is that PLC software development tools are not made to support object-oriented programming. T. Heverhagen and R. Tracht have developed a method for introducing objects to PLC software. They do this by implementing so-called Function Block Adapters (FBA). [IMP FBA] [INT UML] [UST UML] Functional Block Adapters combine data that is related to real world devices and also implement functions for communicating with these devices. This is very much like objects in real object-oriented programming languages. The same thing can be done for data that is used within the control system itself. Function Block Adapters can be used to introduce object-oriented programming to PLC's.

### 3.1.2 Conclusion

The Hatley and Pirbhai software design method is getting rather old and is very much supported anymore by new software design tools. There is a trend in the direction of object-oriented software design, especially UML is getting more and more market share. When systems get more and more complex the Hatley and Pribhai method may get less suitable to properly design the software for the machines, because it would result in to much processes.

UML is becoming a standard for software design in all domains, also in machine control. UML is suited for very small designs as well as very large designs and everything in-between. Replacing the currently used method of Hatley and Pirbhai with UML would have great impact on the design process. All employees involved in the software design process would have to learn the new method.

## 3.2 One control system for all machines

Having a lot of different control systems results in even more problems with the control systems. In an ideal situation, one control system could be designed that could be used for all types of machines.

### 3.2.1 Modular approach

If one control system is designed that should be able to control all types of machines, the starting point would be a control system that is capable of controlling the most complex machines. A control system that could control the most complex machine is must too extensive and expensive to be used without modification for controlling the most simple machines. Thus the control system should be build out of several modules. It must be possible to integrate these modules and so create a control system that is just suited for a specific machine.

The separate modules should be fully developed with modularity in mind. Returning parts of mechanical hardware should be equipped with the same electrical hardware and be controlled by the same software. To optimise the modules, the modules should be developed in such a way that a module can be used on as many different machines as possible.

Because the modular control system should be capable of controlling the most complex machine, and a PLC is not capable of controlling these machines, the modular control system would be build using an industrial computer.

### 3.2.2 Customer driven decisions

BOC Edwards PS is a company with project based development. Many design decisions are made in consultation with the customer.

#### 3.2.2.1 Customers demand the use of a specific brand of PLC

Many customers demand that the machines produced by BOC Edwards PS are equipped with a certain brand of PLC, because the machines they already have are equipped with the same PLC's. For machines that are too complex to control with a PLC a customer accepts the use of an industrial computer as the control system for the machine. For machines that can be controlled by a PLC a customer often demands the use of a certain PLC and the use of an industrial computer is not an option.

### 3.2.3 Conclusion

In the ideal situation, one modular control system could be designed that is able to control all types of machines. Customers however do not accept this approach, because they demand a certain control system to be used on a machine.

## 3.3 Updating the OS-9 based control systems

The hardware that is used in the machines that are controlled by an industrial computer that is running OS-9 is getting outdated. The newest hardware that is used is used in the ILF and the NCCW. The boards that are used are already 8 years old. The result is that the processors on the board have a very low performance compared with recent developments and are very expensive. Nyquist, the supplier of the main control board of the named control systems, no longer do any development regarding controller boards.

Thus the control system is ready for a new revision. If the control system is not renewed, problems can arise in the near future regarding support and obtainability of spare parts.

### 3.3.1 Evaluation of available operating systems

When the control system is updated, a new version of the operating system is necessary, too. The question is if OS-9 is still a good choice as the operating system for the control system. The advantage of sticking with OS-9 over any other real-time operating system is that there is knowledge at BOC Edwards PS about it. Switching to another operating system would imply that all programmers have to become familiar with the new operating system. Thus it is only advisable to change to a new operating system if there is one that is much better than OS-9. Table 3-1 shows a comparison of some many used real-time operating systems. [RTOS MS] I used several evaluation and comparison reports for generating this table. [CMP EVA] [RTOS EV] The reports don't all use the same method of evaluation and comparison of operating systems. So I introduced

weighing factors for the results in the reports. Doing this I was able to produce one table that shows the comparison of several operating systems that were evaluated with different methods. To check the introduced weighing factors, I applied the weighing factors to the comparison and evaluation results of operating systems that were in more than one report.

**Table 3-1: Comparison of some real-time operating systems**

*Description of the column names in the table (points per category can be from 0 to 10):*

PPC: The operating system has support for the Power PC processor.
Intel: The operating system has support for the Intel Pentium, and newer processors.
Price: Initial investment price for the development environment.
R.t.p.100: Price of runtime licences when bought in a quality of 100 at a time.
ROMa: The operating system can be run from a ROM on the destination board.
BOOTP: The operating system has support for booting via the BOOTP protocol.
PCI: The operating system has support for PCI and Compact PCI bus systems.
VME: The operating system has support for VME bus systems.
In/Cfg: Installation and configuration of the development environment and the operation system.
Arch: Architecture of the operating system.
API: Richness of the Application Programming Interface.
Inet: Support for Internet and Internet applications.
Tools: Supplied tools.
Doc: Documentation and support.
Dev: Development.
Perf: Real-time performance of the operating system.

| OS | PPC | Intel | Price | R.t.p.100 | ROMa. | BOOTP | PCI | VME | In/Cfg | Arch | API | Inet | Tools | Doc | Dev | Perf |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| VxWorks | yes | yes | $16,500 | $30,000 | yes | yes | BSP | BSP | 6 | 6 | 8 | 8 | 9 | 7 | 8 | 7 |
| QNX | yes | yes | $1,500 | $12,000 | yes | yes | yes | yes | 7 | 9 | 7 | 8 | 7 | 5 | 7 | 9 |
| OS-9 | yes | yes | $8,000 | $10,000 | yes | no | yes | yes | 7 | 7 | 7 | 7 | 7 | 8 | 8 | |
| Win CE | yes | yes | $3,000 | | | | yes | | 4 | 7 | 7 | 9 | 8 | 5 | 7 | 6 |
| Win NT | no | yes | $1,000 | $23,350 | no | no | yes | no | 5 | 3 | 6 | 8 | 7 | 7 | 7 | 2 |
| pSOS | yes | yes | $8,000 | $20,000 | yes | yes | BSP | BSP | 5 | 7 | 6 | 8 | 8 | 4 | 8 | 6 |

When comparing the numbers in Table 3-1, no real winner can be pointed out. The advantage of another operating system over OS-9, in order to switch to the new operating system, has to come from another domain than a pure technical comparison.

When looking at the amount of available drivers for hardware that is often used in machine control systems, it stands out that VxWorks and OS-9 are the most supported real-time operating systems. Vxworks even a bit more than OS-9. A disadvantage of VxWorks when compared with OS-9 is its high price, not only for the initial investment, but also for the recurring runtime licenses.

It can be concluded that OS-9 is still a good choice as the real-time operating system for the control system of complex machines.

### 3.3.2 Evaluation of available hardware solutions

There are a currently a lot of manufacturers that produce controller boards for industrial computers. By far the most of these boards are equipped with an Intel Pentium processor or a Motorola Power PC processor. There is also a wide range of available interface boards from a wide range of manufacturers.

One of the main arguments for selecting a certain controller board and the necessary interface cards is the number of necessary suppliers. When one or two suppliers supply all hardware it is much easier to get good support. If hardware is used from a lot of different suppliers, the situation that they blame each other when

something is not working is very likely to occur. It is also a lot easier when buying hardware when is can all be ordered at one supplier.

I investigated some possible replacements of the current control system. As a base for the investigation I took the control system of the ILF and looked for a replacement with newer hardware and only a few suppliers. I found the possible solutions that can be found in

Table 3-2 through Table 3-5. For all solutions I assumed that the processor board has enough performance for performing the tasks that were performed by the old controller board as well as the tasks that were performed by the DSP board.

**Table 3-2: Possible control system with almost all hardware from MEN, based on a VME bus**

| **Men VME** | | |
|---|---|---|
| 070000-03 | 19"VME rack 7/12 slot | EUR 2,641.00 |
| A12b | MPC8245@300, 3M-Mod | EUR 1,313.00 |
| | 2 Eth, 2 Com(1 adapted) | |
| 0752-0007 | DRAM | EUR 150.00 |
| 8SA0x-0x | Serial Adapter | EUR 90.00 |
| 0751-0001 | 32 MB cpFlash | EUR 123.00 |
| M66 | 32 D-I/O | EUR 560.00 |
| M59 | 4*16 A-I | EUR 1,439.00 |
| M52 | Centronics | EUR 236.00 |
| A201S | M-Carrier | EUR 417.00 |
| M6 | RS232/RS485 | EUR 330.00 |
| M6 | RS232/RS485 | EUR 330.00 |
| M6 | RS232/RS485 | EUR 330.00 |
| M49 | Stepper motor controller | EUR 582.00 |
| **Phoenix Contact** | | |
| IBS VME6H SC/I | Interbus-S master | EUR 750.10 |
| | | EUR 9,291.10 |

**Table 3-3: Possible control system with almost all hardware from MEN, based on a cPCI bus**

| **Men cPCI** | | |
|---|---|---|
| 0701-0010 | 19" cPCI rack 8 slot | EUR 2,121.00 |
| D3b | MPC8245@300, 3M-Mod | EUR 1,313.00 |
| | 2 Eth, 2 Com(1 addapted) | |
| 0752-0007 | DRAM | EUR 150.00 |
| 8SA0x-0x | Serial Adapter | EUR 90.00 |
| 0751-0001 | 32 MB cpFlash | EUR 123.00 |
| M66 | 32 D-I/O | EUR 560.00 |
| M59 | 4*16 A-I | EUR 1,439.00 |
| M52 | Centronics | EUR 236.00 |
| D201 | M-Carrier | EUR 461.00 |
| M6 | RS232/RS485 | EUR 330.00 |
| M6 | RS232/RS485 | EUR 330.00 |
| M6 | RS232/RS485 | EUR 330.00 |
| M49 | Stepper motor controller | EUR 582.00 |
| *other* | Interbus-S master | EUR 750.00 |
| | | EUR 8,815.00 |

**Table 3-4: Possible control system with all hardware from Motorola and Tews, based on a VME bus**

| **Motorola VME** | | |
|---|---|---|
| *other* | 19" rack | EUR 4,000.00 |
| MVME5101-0131 | 7400-400MHz, 64 Mb, 2PMC | EUR 3,672.93 |
| | 2 Eth, 2/4 Com, 1 Par | |
| MVME761-0x1 | Transition module | EUR 424.02 |
| *other* | cpFlash + Carrier | ? |
| | | |
| | | |
| | | |
| *other* | PMC Carrier | EUR 500.00 |
| *other* | PMC Carrier | EUR 500.00 |
| **TEWS** | | |
| TPMC670-10 | 16 D-I + 16 D-O | EUR 625.00 |
| TPMC551 | 32 channel 16bit A/D | EUR 1,670.00 |
| TPMC861 | 4*RS422/RS485 | EUR 865.00 |
| TPMC118 | 6 channel motion controller | EUR 1,495.00 |
| TPMC821 | Interbus-S master | EUR 1,080.00 |
| | | EUR 14,831.95 |

**Table 3-5: Possible control system with all hardware from Motorola and Tews, based on a VME cPCI bus**

| **Motorola cPCI** | | |
|---|---|---|
| CPX2408 | 19" rack 8 slot | EUR 4,182.90 |
| MCP750-1xx2 | MPC750-233/366/466, 1PMC | EUR 2,939.49 |
| | 16/32/256 MB, KB, mouse, IDE | |
| | 1 Eth, 2/4 Com, 1 Par, cpFlash | |
| TMCP700-001 | Transition Module | EUR 424.02 |
| *other* | 32 MB cpFlash | EUR 150.00 |
| | | |
| | | |
| CPV8540B | PMC Carrier | EUR 487.05 |
| CPV8540B | PMC Carrier | EUR 487.05 |
| **TEWS** | | |
| TPMC670-10 | 16 D-I + 16 D-O | EUR 625.00 |
| TPMC551 | 32 channel 16bit A/D | EUR 1,670.00 |
| TPMC861 | 4*RS422/RS485 | EUR 865.00 |
| TPMC118 | 6 channel motion controller | EUR 1,495.00 |
| TPMC821 | Interbus-S master | EUR 1,080.00 |
| | | EUR 14,405.51 |

When looking at Table 3-2 through Table 3-5, it can be seen that the solutions using only hardware from MEN are a lot cheaper than with Motorola and Tews hardware. Another advantage of the solution with MEN's products is that everything comes from one manufacturer, except the Interbus-S master. Getting everything from on manufacturer is a good thing for getting support. A disadvantage of MEN's products is that MEN only guarantees a product lifetime of 5 years. An end of life announcement is only made three months before the end of life of the specific product. Another disadvantage of using only products from MEN is that they use so-called M-modules for their interface cards. There are not so many manufacturers that produce a wide range of M-modules. Thus, it can be difficult to find a second source for a product if MEN stops producing the specific product.

The solutions that are based on products from Motorola and Tews are much more expensive than the solutions based on products from MEN. The advantage that Motorola has over MEN is that Motorola guarantees a lifetime of a product of at least 15 years. This can become important for maintenance and the availability of spare parts. The solutions that are based on products of Motorola and Tews are build of a main controlled board from Motorola and interface cards from Tews. The interface cards are so-called PMC modules. There are very much manufacturers that produce PMC modules, so it will be possible to find a

second source if Tews stops the production of a specific card. Another advantage of PMC modules over M-modules is that PMC modules are based on a PCI bus. This is a very widely accepted and used bus. All current operating systems have support for the PCI bus and PCI bridges between PCI buses. M-modules use an interface that is only used for M-modules. It is a specific interface that is only supported by a small amount of manufacturers that produces M-modules.

Motorola and Tews do not produce a suitable VME rack and no VME carrier boards for PMC modules. Thus a third manufacturer is necessary for supplying these hardware.

The main supplier of MEN in the Netherlands is Arcobel. Arcobel also offers the delivery of a fully configured system. When a fully configured system is ordered, Arcobel builds together all hardware components and installs the operating system and necessary drivers.

Chess and EBV can supply Motorola and Tews in the Netherlands. Chess offers the same service as Arcobel regarding the delivery of a completely configured system. They can also build together a VME rack with Motorola and Tews hardware. In this case they use a rack and carriers from a third manufacturer.

A control system build together with components from Motorola and Tews based on a cPCI bus has the advantage that all buses between the processor and the interface cards are PCI buses. When all buses are of the same type, the change of incompatible hardware is very little. Almost every current controller board has a PCI bus on the controller board. This PCI bus connects all devices on the board. If a rack is used with another bus than one that is based on PCI, there needs to be a bridge between the different types of buses. It is not very likely that a lot of problems will arise with different buses and different hardware suppliers, but the change is bigger that with one single bus system and only one supplier.

### 3.3.3 Conclusion

A control system that is build together of components from Motorola and Tews seems to be the best solutions. Motorola promises a long lifetime of their products and for the products of Tews there is a good change of a second source when a product is not produced anymore. When choosing for a Motorola and Tews based control system, the best bus system would be cPCI. With a cPCI bus, only one bus type exists in the control system.

A control system based on products from MEN is a lot cheaper than a control system that is based on products from Motorola and Tews. An advantage is that everything comes from one manufacturer. The disadvantage is that there are less second sources. Between the control systems that are based on components from MEN there is not much different. The only difference between the solutions in

Table 3-2 and Table 3-3 is the rack, and thus the bus, the main controller board and the M-module carrier boards. Also a solution has to be found for the field bus controller when choosing for cPCI. A possible solution is to choose the Interbus-S master from Tews together with a carrier board for PMC's. This is the same Interbus-S master controller as is used with the Motorola and Tews solutions. Another solution could be to choose another type of field bus. MEN produces M-modules for Profibus. In that case still all product come from MEN.

So it is not possible to call-out one solution to be the best. There are three solutions that could be used. The choice for a certain solution depends on decisions that have to be made regarding support, lifetime management and price.

## 3.4 Integrating OS-9 based control systems with SCADA systems

The control systems that are based on an industrial computer and are running OS-9 are difficult to integrate with a SCADA system. SCADA systems can get their information from various data sources. However, no standard solutions are available for control systems that are based on real-time operating systems.

### 3.4.1 Possible solutions

In the past very little machines produces by BOC Edwards PS that are controlled by an industrial computer are integrated with a SCADA system. A solution that is used is to place on VME PLC in the control system.

This PLC is only used for the communication between the control system and the SCADA system. This solution can be seen in section 2.3.2. This is an expensive and not very flexible solution.

### 3.4.2 OPC Server

Most of the current SCADA systems can get their data from an OPC Server. OPC is the definition of a standard interface between a client and a server for transferring process information. There are various types of OPC Servers and clients that all use their own set of interfaces. The currently available interface standards are Data Access, Alarms and Events, and Historical Data Access. The OPC interfaces standards are based on Microsoft' DCOM technology. [MS DCOM] As a result almost every OPC Server and OPC Client is running on a Microsoft Windows operating system.

A possible solution for integrating control systems that are based on OS-9 with SCADA systems is to implement an OPC Server for OS-9 controlled systems. The control system would have to make its data available to the OPC Server and the OPC Server provides this data to the SCADA system.

## 4. EXPERIMENT PERFORMED DURING MY GRADUATION PROJECT

After having finished the evaluation of the current situation regarding the control systems of the machines produced by BOC Edwards PS, I did an experiment with the design of an OPC Server for OS-9 Controlled Systems. I started with looking at the context in which SCADA systems are used. Chapter 5 is a description of the context in which SCADA systems are used for process and machine monitoring and how an OPC Server can be used within this context.

After I described the context in which an OPC Server can be used with SCADA systems I describe the specifications an OPC Server for OS-9 control systems has to satisfy in chapter 6. Chapter 7 describes the hardware that is necessary for implementing the proposed OPC Server.

In chapter 8 the software design of the OPC Server is described. This chapter describes the configuration and implementation of the various parts of software that implement the OPC Server.

After I developed the software as described in chapter 8, I did some experiments in order to test if the design meets the requirements as stated in chapter 6.

# 5. CONTEXT DESCRIPTION FOR THE DESIGN OF AN OPC SERVER FOR OS-9 BASED CONTROL SYSTEMS

## 5.1 Introduction

Manufacturers of fluid medications need equipment that conforms to very stringent regulations. These regulations cover the whole production process. Not only the production of the fluids is subject to these regulations, but also the packing of them. BOC Edwards PS designs and produces machines for packing fluid medications. These machines cover the whole packing process from washing and sterilisation to freeze drying and loading and unloading systems. All these machines need to comply with the same very stringent regulations.

One of the results of the regulations is that almost all machines are placed in isolators and clean rooms. To comply with the regulations, pharmacists want to be able to operate the machines with a minimum of operators in the clean rooms. Thus, they want to be able to operate a machine from a remote location. Besides operating the machine remotely, they also need to monitor the processes on the machine. They may want to gather statistical information from it or log some data, so they can process the data later on.

Because a pharmacist doesn't always buy his equipment from one machine manufacturer, but wants to be able to control the entire production line from one location, the machines of different manufacturers must be able to be integrated in one control system. This is only possible if all their control systems comply with standards that are widely used in the world of pharmaceutical machine control.

Most machines that are produced by BOC Edwards PS and other manufacturers are controlled by PLC's. Remote control for these machines is done with so called SCADA systems. Some machines that are produced by BOC Edwards PS are not controlled by a PLC. They are controlled by a control system that is based on an embedded computer and a real-time operating system. In order to be able to integrate these machines with the same SCADA systems that are used for remote control of PLC controlled machines, these machines must be able to communicate with the same SCADA systems that are used for PLC controlled machines.

Because the embedded computers that are used to control the machines mentioned above, have a much higher performance then PLC's, these control systems can be used to produce much more information about the processes then traditional PLC based control systems. Pharmacists may want to use this data and thus it must be made available to s remote system by the control system.

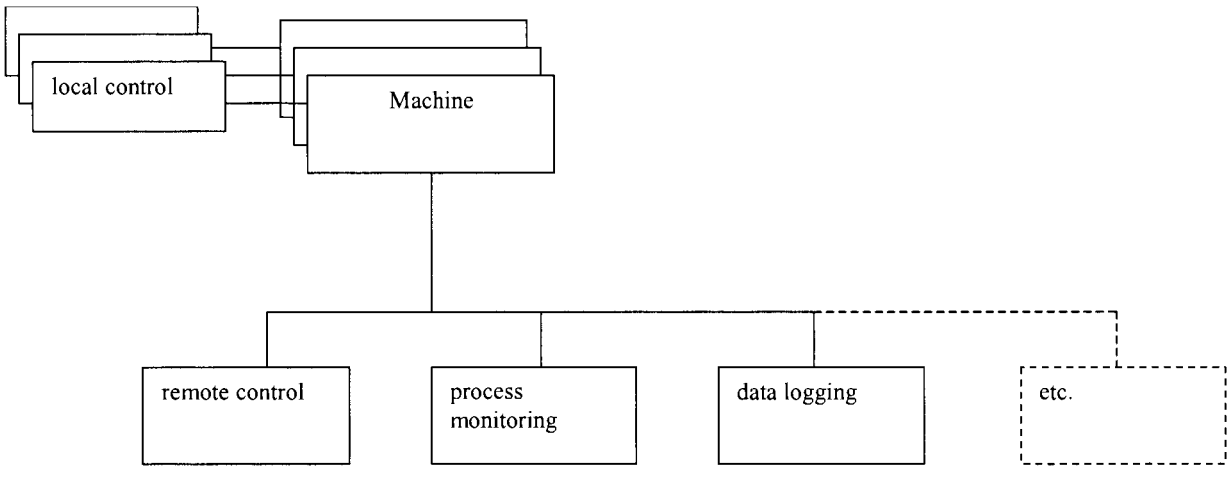


**Figure 5-1: Schematic overview**

### 5.2 Current standards accepted by the market

In the past different SCADA systems existed for every brand of PLC. It was very difficult to integrate the control of different systems into one SCADA system. Later on standards have been developed to ease the integration of the different control systems and to be able to use the same SCADA systems with different control systems.

There is currently one standard that is widely accepted and used in the world of machine control. This is the OPC standard. OPC stands for OLE for Process Control. The technology behind this standard is based on and uses Microsoft's DCOM technology. The OPC standard defines interfaces that are used by OPC clients and OPC servers that are used to communicate with each other. Almost every PLC manufacturer provides OPC servers that are able to communicate with almost every type of PLC. Almost every current SCADA system is able to function as an OPC client and to communicate with an OPC server. Because OPC is based on Microsoft's DCOM technology, almost every OPC server runs on a Microsoft Windows NT or Windows 2000 system.

Most OPC servers in the market are OPC servers for PLC's. A second type of OPC server that is offered by many manufacturers is an OPC server that communicates directly with connected hardware. In this case the computer that is running the OPC server is equipped with an interface that can communicate with the hardware directly.

Almost no OPC servers are available for real-time operating systems. No other standard is available for the integration of real-time operating systems with current SCADA systems. This makes it very difficult to integrate the machines that are produced by BOC Edwards PS that are controlled by an embedded computer running a real-time operating system with a current SCADA system.

### 5.3 How to integrate a control system based an a RTOS with SCADA

There are several possibilities to produce an OPC server interface for a control system that is based on a real-time operating system. Because OPC servers are available for almost all PLC's, a possible solution can be to integrate a PLC with the control system. The only functionality of this PLC will then be to provide an interface to an available OPC server. The data that is provided by the control system will be written to the PLC, which then forwards it to the OPC server.

Another solution is to implement the OPC server completely on the real-time operating system. To do this, the complete OPC server interfaces have to be implemented in the control system's operating system. Because the OPC interfaces are based on Microsoft's DCOM technology, at least a part of DCOM has to be developed for the real-time operating system too.

A third possible solution is using an OPC toolbox to develop the OPC server. In this case the OPC server will run on a PC that is running Microsoft Windows NT, 2000 or XP. This OPC server communicates with the real-time operating system in the control system by a custom interface that can be specified by the develop himself. The different solutions have their own advantages and disadvantages as shown in Table 5-1.

Table 5-1: Comparison of different OPC solutions

1. *Place a PLC in the rack of the control system.*
2. *Implement the OPC server completely on the real-time operating system.*
3. *Design an OPC server using an OPC server toolbox and let this OPC server communicate with the real-time operating system.*

| | Developmen t time | Work when standard updated | Implementation flexibility | Initial costs for licenses | Recurring costs for hardware and licenses | Recurring work |
|---|---|---|---|---|---|---|
| 1 | little | very little | low | low | very high | Little |
| 2 | very high | probably very much | very high | very low | very low | Implementatio n dependent |
| 3 | medium | little; toolbox update | as flexible as wanted | medium | very low | Implementatio n dependent |

When a PLC is placed in the control system and a standard OPC server that communicates with this PLC is used, there is very little flexibility in the implementation of the OPC server. The performance and capabilities of a PLC are much less then than those of an embedded computer. When a PLC is placed in the control system with the purpose of communicating with the OPC server, the possibilities are limited by the capabilities of the PLC. The recurring costs for hardware and licenses are very high when this solution is used. A PLC that can be placed in a rack of an embedded PC is very expensive. Also, licenses have to be paid for the use of the standard OPC server.

The development of an OPC server entirely on the real-time operating system is very much work. A lot of software has to be developed. The developed software has to comply with several standards. The advantages of this solution are that almost no recurring costs are there and that the developer can freely implement the OPC server as long compliance with the standard is maintained.

The third solution can been seen as a compromise of all solutions. The development time is more than with the first solution, but a lot less than with the second. The most difficult part of the second solution, compliance with standards, does not have to be implemented by the developer of the OPC server. It is provided by the toolbox. The design of the part of the OPC server that has to be implemented on the real-time operating system is very flexible with this solution. The developer of the OPC server can freely design the functionality in the control system and the communication between the real-time operating system and the OPC server. The OPC Server for OS-9 controlled systems will be developed using a toolbox.

### 5.4 Different types of OPC servers

OPC does not stand for one single standard. There are several different standards within OPC. They can be divided into 3 groups. The existing groups of standards are Data Access, Alarms and Events and Historical Data. [OPC OVW]

#### 5.4.1 Data Access servers

Data Access servers are designed for efficiently reading and writing of data between an application and a control system in a flexible way. Data Access servers allow clients to maintain information provided by the server in groups. The client creates groups on the server and places items in these groups. These items are links to the actual tags that provide the data. This mechanism provides a very flexible way to work with the provided tags that provide the data. [OPC DAC]

### 5.4.2 Alarms and Events servers

Alarms and Events servers provide mechanisms for OPC clients to be notified of the occurrence of specified events and alarms conditions. The conditions that are used to generate the alarms and events are set in the server. Thus the alarms and events that are being send to the clients are generated in the server and not in the control system. The tags in an Alarms and Events server have conditions associated with them. When one of these conditions is met, this is signalled to the connected clients.

### 5.4.3 Historical Data Access servers

Historical Data Access servers provide the possibilities for reading, processing and editing of historical data. A Historical Data Access server can be seen as an extension to a Data Access server. It stores the data provided by a Data Access server in a database for processing it later. There are several types of Historical Data Access servers. There are simple trend data servers that just store raw data that can be retrieved later on. There are also complex data compression and analysis servers. These servers process the incoming data as well as they store the raw data. Various operations on the data are possible.

### 5.4.4 Combining the different types of servers

The different types of OPC servers use separate DCOM interfaces to communicate with OPC clients. As a result OPC clients need to be equipped with the different types of OPC interfaces to be able to communicate with the different types of OPC servers. One of the advantages of using different interfaces for different types of OPC servers is that the different types of OPC servers can be integrated in one executable. This single executable provides the different DCOM interfaces for the different types of OPC servers. Each type of OPC server has a set of interfaces that are designed with the purpose of the specific type of server in mind. Thus the interfaces of the different types of OPC servers differ between the different types of OPC servers.

#### 5.4.4.1 Current SCADA systems

Most current SCADA systems are designed as a client server application. The server is responsible for retrieving the data from data providers in the system. The clients display the data in a user-friendly way. Normally the clients are programmable by the customer.

The communication between the server and the client in a SCADA system can be an OPC interface, be it can also be a manufacturers specific interface. The functionality that a server in SCADA systems provides often includes the functionality of a Data Access, Alarm and Event, and Historical Data Access OPC Server. It can be seen as a combination of all three types of OPC Servers. The server acquires its information for one or more data sources. These data sources often can be Data Access OPC Servers. The server in the SCADA system uses this data for all three types of internal servers.

SCADA clients connect to the SCADA server and use it as the source of their data. The SCADA server uses the data it acquires from the data sources for all internal servers. The Data Access server makes the real data available to the clients in a structured way. The Alarms and Events server compares the real data with thresholds that are set in the Alarms and Events server by the connected clients. When a tag's value passes a certain threshold that is set by a client, an alarm or event is generated and sent to connected clients. The Historical Data Access server stores the data that is acquired in a database and provides an interface to clients to access the data in the database. Figure 5-2 shows an example of a SCADA server with integrated Data Access, Alarms and Events, and Historical Data Access servers in a SCADA system.

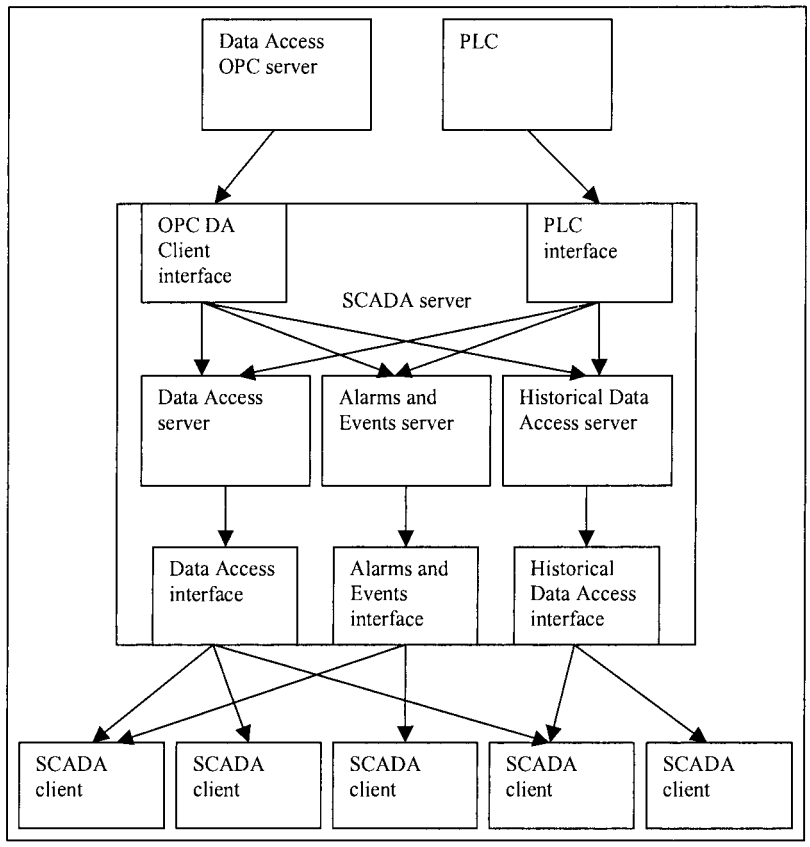


**Figure 5-2: Example of a SCADA system configuration**

### 5.4.4.2 Needed functionality of the OPC Server for the OS-9 controlled systems

The main purpose of the OPC Server for OS-9 controlled systems is to provide an interface to SCADA systems for accessing the tags of the control system, whether these tags are hardware devices or internal variables of the control processes of the control system. The Data Access server is the type of OPC server that provides this functionality as its main purpose. Thus a Data Access server would be the best solution.

However, the control system can generate a lot of alarms when something goes wrong during production. It seems that an Alarm and Events server would also be a good solution to be used next to a Data Access server. However, all alarms that occur in the control system are generated in the control system itself. They are alarms and events that are provided to the outside world by the control system. This does not match with the operation of the Alarms and Events server, where the alarms and events are generated inside the OPC server. The alarms and events that are generated inside the Alarms and Events server are not necessarily caused by an alarm or event in the control system. It is normally based on a condition that is met for a specific tag. Of course it is possible to let the OPC server generate an alarm or event when an alarm or event occurs in the control system. The conditions associated with the tag that is related to the specific alarm or event must be set to appropriate values to achieve this.

A Historical Data Access server can be seen as an extension to a Data Access server. Its main goal is retrieving and storing data from a control system. It is not intended as a real-time system that provides real-time data for an operator. The way data the data is stored and what operations are provided and performed by a Historical Data Access server is up to the developer of the specific server. All data processing operations can be implemented as wanted. Nowadays, many SCADA systems are capable of storing historical data itself.

#### 5.4.4.3 What types of OPC servers are to be integrated in the OPC Server for OS-9 controlled systems

Because most current SCADA systems use a client server architecture and the servers in the system provide Data Access, Alarms and Events, and Historical Data functionality, the OPC Server for OS-9 controlled systems does not need to have all these functionalities when used with an existing SCADA system. The SCADA systems use a Data Access server to acquire their data and use that data for the three types of servers.

A reason for implementing Alarms and Events and Historical Data Access functionality in the OPC Server for OS-9 controlled systems is when no existing SCADA systems is used, but when the SCADA system is developed by BOC EPS themselves. The functionality of Alarm and Events and Historical Data Access needs to somewhere in the system. To be flexible and modular it is wise to use standardised interfaces in the system. Alarms and Events and Historical Data Access OPC interfaces are by far the most used interfaces in process monitoring applications regarding these functionalities.

BOC EPS does not plan to design its own SCASA system in the near future. They will use existing SCADA systems. Thus there is not must need to implement Alarms and Events and Historical Data Access functionality in the OPC Server for OS-9 controlled systems. Current SCADA systems provide this functionality and use only a Data Access OPC Server for retrieving the data for these servers.

The OPC Server for OS-9 controlled systems will be implemented as a Data Access server. It has to comply with the OPC Data Access interface standard. In order to make sure that this compliance is ensured a toolbox will be used that provides this interface. The toolbox consists of source code and libraries. On one side the toolbox provides the OPC DA interface. On the other side it provides an Application Programming Interface to make the data available to the server that is integrated in the toolbox.

# 6. FUNCTIONAL SPECIFICATION OF THE OPC SERVER FOR 0S-9 CONTROLLED SYSTEMS

## 6.1 Introduction

This document describes the functional specifications for the OPC Server for OS-9 controlled systems of BOC Edwards Pharmaceutical Systems. It is not intended to be a detailed system design document, but a functional description of how the system is required to work.

Almost every current SCADA system can gather its information from an OPC Server. OPC Servers are available for almost every brand of PLC available nowadays. This makes it relatively easy to integrate PLC controlled systems with current SCADA systems. Most machines developed by BOC EPS are controlled by a PLC, but some are controlled by an embedded PC, that is running on OS-9. However, no OPC Servers are available for OS-9 controlled systems. This document describes the functionality that the OPC Server for OS-9 controlled systems must provide.

Figure 6-1 shows a possible configuration for a fully configured SCADA system with an OPC Server that gets its information from a PLC, an OPC Server with direct IO and an OPC Server for OS-9 controlled systems. The parts of Figure 6-1 that are bold are described in this document. A toolbox is used for creating the OPC Server for OS-9 controlled systems. This toolbox provides an OPC 2.0 compliant interface, which is used by OPC Clients and SCADA systems. The OPC Server application, which provides the OPC Server with an interface to the data providers in the system, is combined with the toolkit in an executable program.

The specifications laid down in this document are to be used as a base for the development of the OPC Server for OS-9 controlled systems. The hardware and software related information shall be taken into account when the OPC Server for OS-9 controlled systems is designed.

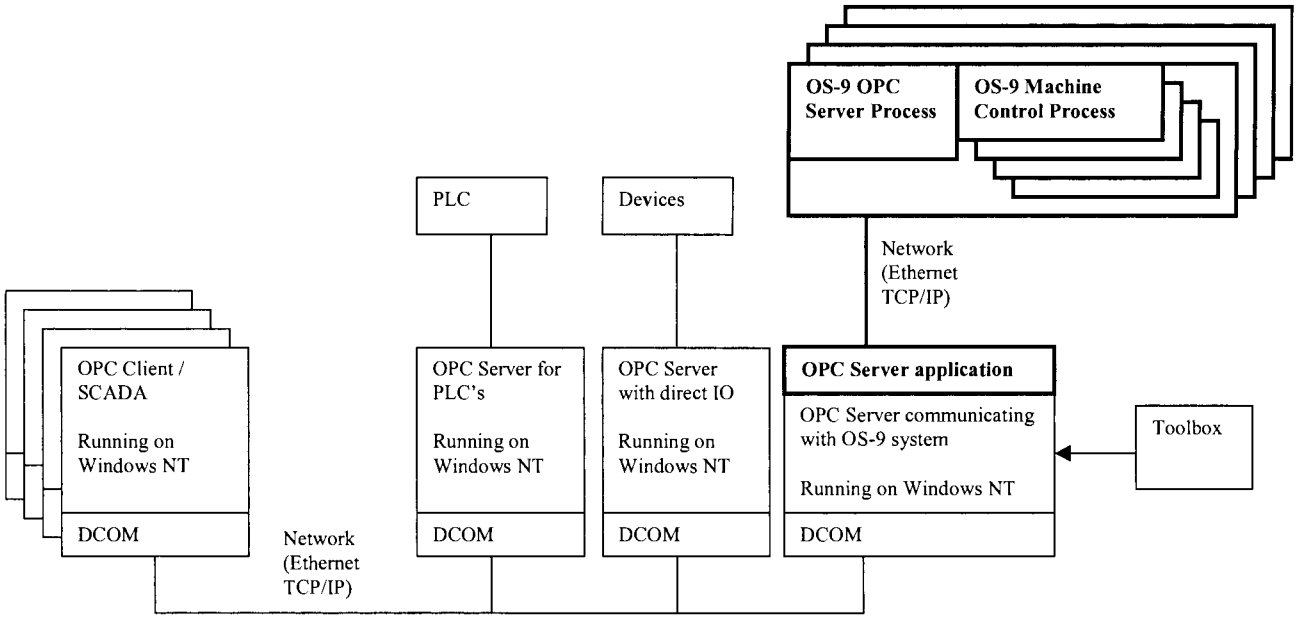


**Figure 6-1: Schematic overview of a fully configured SCADA system**

## 6.2 Description of the OPC server for OS-9 controlled systems

### 6.2.1 Functional description

The OPC Server for OS-9 controlled systems is designed to be able to integrate the systems, that are controlled by a control system that is based on OS-9, with existing SCADA systems. OPC Servers are standardised applications that provide an interface to SCADA systems. They make it possible for SCADA systems to access devices in a machine in a standardised manner.

OPC is based on Microsoft's DCOM technology. Because of this, almost every OPC Server runs on a standard PC running Microsoft's Windows NT, Windows 2000 or Windows XP. Existing OPC Servers gather their information from PLC's or directly from an interface card in the PC that is running the OPC Server.

The OPC Server for OS-9 controlled systems is able to gather information from a machine that is controlled by a control system that is based on OS-9. The OPC Server provides a standard OPC interface to SCADA clients on one side. On the other side the OPC Server for OS-9 controlled systems communicates with a process in an OS-9 controlled system. This communication is used to read and write information from and to the devices of the machine.

### 6.2.2 Functional units

The OPC Server for OS-9 controlled systems is divided into 4 functional units:

1. OPC Server
2. OS-9 OPC Server process
3. OS-9 machine control processes
4. Data providers

#### 6.2.2.1 OPC Server

The OPC Server is a program that runs on a standard PC running Microsoft's Windows NT, Windows 2000 or Windows XP. The OPC Server is started when a SCADA client wants to connect to it. The OPC Server then provides an interface to all configured devices in the OS-9 controlled system. The OPC Server can gather information from one or more OS-9 controlled systems at the same time.

The OPC Server has a graphical interface for maintenance. The interface allows an administrator to change and save the configuration of the OPC Server. The administrator is able to add and remove machines from the OPC Server and he can download and save the list of available data providers from the machines.

The OPC Server is compliant with the OPC Data Access 2.0 specifications. In order to comply with this specification, the OPC Server is developed using a toolbox. The toolbox used is the OPC Data Access Server Toolbox of Softing. This toolbox provides the OPC Server developer with an object-based interface for providing access to the data providers for the OPC Server.

#### 6.2.2.2 OS-9 OPC Server process

The OS-9 OPC Server process is a process running on the OS-9 controlled system. It runs on the control system's mainboard, together with all processes that are responsible for controlling the machine. The OS-9 OPC Server process takes care of all communications with the OPC Server. The OS-9 OPC Server process also performs the communication with the machine control processes on the system.

The OS-9 OPC Server process functions as a server for TCP/IP communications. The OPC Server is the client that connects to it. The OS-9 OPC Server process also functions like a server inside the control system. All machine control processes connect to the OS-9 OPC Server process.

When a machine control process connects to the OS-9 OPC Server process it provides the OS-9 OPC Server process with a list of available data providers. The list is provided as names. For performance, a number is associated with each item by the OS-9 OPC Server process and negotiated with the connecting machine control process.

When the OPC Server connects to the OS-9 OPC Server process. The OS-9 Server process provides the OPC Server with its list of available items.

#### 6.2.2.3 OS-9 machine control processes

The OS-9 machine control processes are the processes that actually communicate with the data providers in the machine. These processes already communicate with the physical devices on the machine, because they need to communicate with these devices for controlling the machine. The OS-9 machine control processes function as an interface between the OS-9 OPC Server process and these devices. The OS-9 machine control processes communicate with the OS-9 OPC Server process in order to provide this process with information about the status of the devices and to allow the OS-9 OPC Server process to write to the devices. OS-9 machine control processes can also create data providers for other data than from physical devices. This can be machine parameters (for example, a recipe parameter) or process variables.

#### 6.2.2.4 Data providers

The devices on the machine can be connected to the control system with a fieldbus or directly to the control system with an analog, digital or serial interface. The OS-9 machine control processes read and write information from and to all devices on the machine. The information read from the devices is passed to the OS-9 Server process by the OS-9 machine control processes. The OS-9 Server process subsequently forwards these values to the OPC Server. Next, the OPC Server can provide these values to a connected SCADA system. When a connected SCADA system wants to write information to a device, it sends a write request to the OPC-Server. The OPC Server forwards the request to the OS-9 Server process. This process forwards it to the specific OS-9 machine control process. This process sends the information to the device.

In addition to the physical devices on the machine, the OS-9 machine control processes can also provide information about the status of the process via accessible variables. Other process-related information that is not directly related to hardware devices can also be accessible. The machine control processes determine what information is provided to the OS-9 OPC Server process. The OS-9 OPC Server process also provides some accessible variables. These variables give information about the status of the OS-9 OPC Server process itself.

### 6.3 Hardware specification

#### 6.3.1 Hardware units

The different units of the OPC Server for OS-9 controlled systems run on different platforms.

##### 6.3.1.1 OPC Server

The OPC Server runs on a standard PC that is running Microsoft's Windows NT, Windows 2000 or Windows XP. This PC needs to be equipped with one or more Ethernet network interface cards. It needs to be able to communicate with the PC's that are running the SCADA systems. This can be the same PC on which the OPC Server is running, but normally it isn't. The PC running the OPC Server must also be able to communicate with the OS-9 controlled machine using TCP/IP over Ethernet. The same network interface card as for the communication with SCADA systems can be used for this, but it is also possible to use a separate network interface card.

#### 6.3.1.2 OS-9 processes

The OS-9 processes are running on the control system of the machine. This control system needs to be equipped with an Ethernet network interface card. The OS-9 OPC Server process uses this interface to communicate with the OPC Server using TCP/IP.

### 6.4 Software specification

#### 6.4.1 OPC Server

The OPC Server is programmed using an OPC Server toolbox. This toolbox provides the OPC Server interfaces that are compliant with the OPC Data Access Server 2.0 specifications. This ensures that the OPC Server conforms to the OPC specifications. The toolbox used is Softing's OPC Toolbox 3.04. The part of the OPC Server that communicates with the OS-9 OPC Server process is integrated in the OPC Server application. It is build around the interface provided by the OPC Server toolkit. The OPC Server application is programmed in C++ and compiled with Microsoft's Visual C++ 6.0.

#### 6.4.2 OS-9 OPC Server process

The OS-9 OPC Server process is a separate process that is running on the machine's control system. It communicates with the OPC Server and with the other processes in the control system. It is programmed in C++ and compiled with the same compiler that is used for compiling the other process of the control system.

#### 6.4.3 OS-9 machine control processes

The main function of the OS-9 machine control processes is controlling the machine. These processes' functionality needs to be extended with the functionality to communicate with the OS-9 OPC Server process and provide this process with a list of accessible data providers.

### 6.5 Performance requirements

#### 6.5.1 Introduction

The following calculations are based on a machine that will be developed by BOC Edwards PS in the near future. This machine is an integration of the Inline Filler and the Non Contact Check Weigher. This machine fills vials while they are transported in a single line. The vials are transport through the machine at a constant speed while they are filled. The machine is capable of weighing the amount of fluid that is put in the vials during filling without touching the vials. The weighing is done using nuclear magnetic resonance technology.

#### 6.5.2 Future expectations

The machine must be capable of processing 600 vials per minute with the first release. In about 5 years this can be increased to 1000 vials per minute. The number of vials that are filled per minute increases in time. In Table 6-1 you can see developing of the number of vials that could be filled in the past and what the expectations are for the near future.

Table 6-1: number of vials per minute that can be filled

| When | vials / minute |
|---|---|
| 5 years ago | 250 |
| 2 years ago | 400 |
| In 2 years | 600 |
| In 5 years | 1000 |

Because machines get more and more complex, a same kind of trend can be seen when looking at the number of devices on a machine. For each new machine or each new version of a machine, more and more is monitored during the production process. This monitoring causes more and more data to be generated. The result is that not only the amount of sensors and other devices increases, but also the number of process variables that exist inside the machine control processes. An expectation for the number of devices and variables that will exist on future machines as well as that exist on recent machines can be seen in Table 6-2.

Table 6-2: Number of devices and variables on a machine

| When | Digital input | Digital output | Analog input | Analog output | Internal variables | Process variables | Alarms |
|---|---|---|---|---|---|---|---|
| 5 years ago | 50 | 70 | 5 | 5 | 700 | 15 | 100 |
| 2 years ago | 120 | 110 | 10 | 20 | 750 | 20 | 150 |
| in 2 years | 150 | 150 | 20 | 25 | 850 | 30 | 200 |
| in 5 years | 200 | 200 | 40 | 30 | 1000 | 35 | 250 |

### 6.5.3 Base for calculations

As said before, the calculations in this chapter are based on a machine that will be developed in the near future by BOC Edwards PS. This machine will be called the Inline Weight Filler (IWF). The IWF will have about the amount of devices and variables as shown in Table 6-3.

Table 6-3: Number of devices and variables on the IWF

| Type of data provider | Number of data providers |
|---|---|
| Digital inputs | 150 |
| Digital outputs | 150 |
| Analog inputs | 20 |
| Analog outputs | 25 |
| Result values / counters | 25 |
| Alarms | 200 |
| State information | 100 |
| Recipe configuration parameters | 50 |
| Engineering parameters | 300 |
| Operator commands | 400 |
| Statistical information | 5 |

#### 6.5.3.1 Digital inputs

Digital inputs are mostly sensors that only have an on and an off state. Most of them are connected to the control system by the field bus and some are connected to it directly. About a quarter of the digital inputs have a rate of change that is double the number of vials that is handled per minute. The others don't change regularly.

#### 6.5.3.2 Digital outputs

Digital outputs are mostly actuators that only have an on and an off state. Most of them are connected to the control system by the field bus and some are connected to it directly. About a quarter of the digital outputs have a rate of change that is equal to the number of vials that is handled per minute.

### 6.5.4 Analog inputs

Analog inputs are mostly connected to sensors that check process values like pressure and temperature. Because of the nature of analog values they are not constant and change continuously. These continuous changes would cause an enormous flow of data. To reduce the amount of data, a dead band and/or a maximum update rate can be set for each analog input. For calculations all analog inputs are said to have a rate of change that is equal to the number of vials that is handled per minute.

### 6.5.5 Analog outputs

Analog outputs are mostly used for setting the speed of motors on the machine. Most of them are connected to the control system by the field bus and some are connected directly. The nature of analog values allows the analog outputs to change very frequently and with very small differences. This is normally not the case. If an analog output has this kind of behavior the created flow of data can be reduced by defining a dead band and/or a maximum update rate. For calculations all analog outputs are said to have a rate of change that is equal to the number of vials that is handled per minute.

### 6.5.6 Result values / counters

During production some internal variables and counters are maintained. They normally change at a rate that is equal to the number of vials that is handled per minute.

### 6.5.7 Alarms

Alarms are internal variables of the machine control processes. They normally don't change during normal operation. When an alarm occurs, a chain reaction of alarms occurs. One alarm causes several other alarms to occur.

### 6.5.8 State information

State information consists of internal variables of the machine control processes. They don't change often during normal operation. During startup and alarms they do change in bursts.

### 6.5.9 Recipe configuration parameters

Recipe configuration parameters are set before a batch is started. They are not changed during production.

### 6.5.10 Engineering parameters

Engineering parameters are not changed when a machine is delivered to a customer. However, they can be made visible. They are only changed during tuning and testing of the machine. During normal operation they can be considered as constants.

### 6.5.11 Operator commands

Operator commands are commands that an operator can send to the machine to operate the machine. During normal operation they are not very frequently used. They occur in small bursts when the machine is operated.

### 6.5.12 Statistical information

Statistical information contains values that are related to the machine's process. They give information about temperature, pressure, etc. per vial. These values may not be overwritten by a new value before they are used by a client. Thus, handshaking has to be performed for these values. Statistical information is generated at a rate that is equal to the number of vials that is handled per minute.

## 6.6 Performance calculations during normal operation (production)

About a quarter of the digital inputs will have a rate of change that is double the number of vials that is handled per minute and the analog outputs change at a rate that is equal to the number of vials that is handled per minute. The other digital IO's will have a much lower rate of change. They are mainly used to monitor the state of the machine and to turn devices on or off.

The analog IO's will normally not have a constant value, due to the nature of analog signals. A dead band and/or a maximum rate of change are used to determine the rate of change for these IO's. For the following calculations all analog IO's are said to change at a rate that is equal to the number of vials per minute that is handled by the machine.

Besides the values of IO's on the system, some process values that can be used for statistics need to be available. These values include weight, filling time, filing pressure, liquid temperature during filling, temperature during weighing. It is not allowed to overwrite these values with new ones before they are processed by a client. Thus, handshaking needs to be performed for these items.

Normally it is allowed to miss the change of a single high rate IO, because these changes are not counted neither are used for statistics. Normally it is not very useful to make these devices visible on a SCADA system at such a high rate. It is much more convenient to display the results of the changes (ie. number of rejected vials, number of not-stoppered vials, number of vials filled, etc.). The rate at which these values change is typically the same or in the same range as the number of vials handled per minute.

(1) Maximum number of vials per minute that the machine can handle: 1000 (vials/minute)

(2) Number of digital IO's that change at a rate that is double the number of vials per minute: 40

(3) Number of digital IO's that change at a rate equal to number of vials per minute: 40

(4) Statistical information per vial: 5

(5) Estimation for the number of counters that change per vial: 25

(6) Number of analog IO's. As said before, they are assumed to change for every vial: 45

(7) Estimation for the number of low rate IO's: 150

(8) Estimation for the rate of change for low rate IO's per minute: 1 (changes/minute)

Alarms, state information, recipe configuration parameters, engineering parameters and operator commands don't change or need to be written during normal operation.

Because it is not always necessary to update the value of a tag for every change in a high-speed digital input to calculations are made. For the first calculation it is assumed that no digital input data is used. This calculation is shown in Table 6-4.

**Table 6-4: Calculation without high-speed digital inputs**

| | count | | rate | | | messages | |
|---|---|---|---|---|---|---|---|
| (4)(1) | 5 | * | 1000 | * 2 | = | 10,000 | (* 2 because of handshaking) |
| (5)(1) | 25 | * | 1000 | * 1 | = | 25,000 | |
| (6)(1) | 45 | * | 1000 | * 1 | = | 45,000 | |
| (7)(8) | 150 | * | 1 | * 1 | = | 150 | |
| | | | | | | ----------+ | |
| | | | | | | 80,150 | |

Total number of changes per second: 80150 / 60 ≈ 1350

The opposite situation is when all high-speed digital inputs are used. This calculation is shown in Table 6-5.

**Table 6-5: Calculation with all high-speed digital inputs**

| | count | | rate | | | messages | |
|---|---|---|---|---|---|---|---|
| (2)(1) | 40 | * | 1000 | * 2 | = | 80,000 | (* 2 because of double rate) |
| (3)(1) | 40 | * | 1000 | * 1 | = | 40,000 | |
| (4)(1) | 5 | * | 1000 | * 2 | = | 10,000 | (* 2 because of handshaking) |
| (5)(1) | 25 | * | 1000 | * 1 | = | 25,000 | |
| (6)(1) | 45 | * | 1000 | * 1 | = | 45,000 | |
| (7)(8) | 1 | * | 150 | * 1 | = | 150 | |
| | | | | | | ----------+ | |

Total number of changes per second: 20150 / 60 ≈ 3350

The results of these calculations denote the number of changes of values for all used data providers in the system. If no special action is taken, each change of value results in a message in the system. Typically this message is generated by a machine control process and send to the OS-9 OPC Server Process. This process forwards the message to the OPC Server. In the OPC Server the value of the tag associated with the data provider is updated. If the tag has connected items and one or more of the items is active in a client, the updated value is forwarded to the clients.

When all tag information is sent in a separate message for each tag, the number of messages calculated in Table 6-4 and Table 6-5 result in real messages through the system. This is not a very efficient way of transferring the available data. Digital IO's for example can be represented by only one bit. Each message that holds information about a tag's value has a header of about 10 bytes. This is explained in chapter 8. When only one bit holds the actual information and 11 bytes (1 byte is necessary for storing the information bit), the overhead is ((8*11)–1)/(8*11)=98,9%. Thus is a good idea to combine the information of several tags in a message in order to reduce the overhead and at the same time the number of messages that has to be sent. This is further discussed in section 8.3.4.

When no optimisations are performed and all information is sent in a separate message, the results of the following calculations denote the data rate that the system produces. These calculations are made with the assumption that all real values are stored in 64 bit floating-point variables, which are more accurate then 32 bit floating-point variables.

- The information for digital IO's can be stored in a one byte integer, (2) and (3).
- The information for statistical information can be stored in a floating-point variable, (4).
- The information for counters can be stored in a four byte integer, (5).
- The information for analog IO's can be stored in a floating-point variable, (6).
- The information for low rate IO's can be stored in a one byte integer, (7).

**Table 6-6: Data rate calculations without high-speed digital inputs**

| | count | | rate | | | | messages | | header | | data | | B/min | B/sec |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (4)(1) | 5 | * | 1000 | * | 2 | = | 10,000 | *( | 10 | + | 8 | )= | 180,000 | 3,000 |
| (5)(1) | 25 | * | 1000 | * | 1 | = | 25,000 | *( | 10 | + | 4 | )= | 350,000 | 5,833 |
| (6)(1) | 45 | * | 1000 | * | 1 | = | 45,000 | *( | 10 | + | 8 | )= | 810,000 | 13,500 |
| (7)(8) | 150 | * | 1 | * | 1 | = | 150 | *( | 10 | + | 1 | )= | 1,650 | 28 |
| | | | | | | | ---------- | | | | | | ---------- | ---------- |
| | | | | | | | 80,150 | | | | | | 1,341,650 | 22,361 |

**Table 6-7: Data-rate calculations with high-speed digital inputs**

| | count | | rate | | | | messages | | header | | data | | B/min | B/sec |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (2)(1) | 40 | * | 1000 | * | 2 | = | 80,000 | *( | 10 | + | 1 | )= | 880,000 | 14,667 |
| (3)(1) | 40 | * | 1000 | * | 1 | = | 40,000 | *( | 10 | + | 1 | )= | 440,000 | 7,333 |
| (4)(1) | 5 | * | 1000 | * | 2 | = | 10,000 | *( | 10 | + | 8 | )= | 180,000 | 3,000 |
| (5)(1) | 25 | * | 1000 | * | 1 | = | 25,000 | *( | 10 | + | 4 | )= | 350,000 | 5,833 |
| (6)(1) | 45 | * | 1000 | * | 1 | = | 45,000 | *( | 10 | + | 8 | )= | 810,000 | 13,500 |
| (7)(8) | 150 | * | 1 | * | 1 | = | 150 | *( | 10 | + | 1 | )= | 1,650 | 28 |
| | | | | | | | ---------- | | | | | | ---------- | ---------- |
| | | | | | | | 200,150 | | | | | | 2,661,650 | 44,361 |

The results in Table 6-6 and Table 6-7 denote the data rate in bytes/minute and bytes per second that is needed to send all messages over the network. The system should be capable of handling this amount of data.

### 6.7 Jitter requirements

Some items can be used to generate representations between its value and the time. In order to be able to create good representations, one must be able to say something about the interval at which the items are presented. The values that are used for creating trend graphics need to be handled in a constant manner. The jitter between updates needs to be minimized for these items.

Different situations that are dependent on the design of the different parts of the OPC Server components can occur. Some situations also occur as the result of missing capabilities of client applications. Table 6-8 gives an overview of situations that can occur and what situations can cause jitter problems.

The design of the different parts of the OPC Server must minimise the jitter in the problematic situations. The design of the OPC Server makes it impossible to fully prevent the occurrence of jitter in the system, because of the non-deterministic network parts in the system.

**Table 6-8: Different possible situation concerning jitter in the system**

| OS-9 | Client | Problem |
|---|---|---|
| Generate local timestamp as item-property | Use generated timestamp for X-values | No problem |
| Generate values at equal distance in time | Place values at equal distance on the X-axis | No problem |
| Generate local timestamp as item-property | Place values at equal distance on the X-axis | Values are not placed at the correct X-location because values are not created at equal distance in time |
| Generate local timestamp | Use timestamp assigned by the OPC Server | Jitter in the communication between the OS-9 system and the OPC server causes the X-values not to be placed at the correct position. |
| Create values at equal distance in time | Use timestamp assigned by the OPC Server | Jitter in the communication between the OS-9 system and the OPC server causes the X-values not to be placed at the correct position. |

## 6.8 Performance requirements concerning alarms, and start and stop operations

Unlike in normal operation, in which the updates form a continuous stream of data, alarms and start and stop operations cause a bursty data flow.

Bursts occur in the following situations:

[1] Start-up: When the OPC Server connects to the OS-9 OPC Server Process, the Machine Control Processes need to sent their complete list of available items. The OS-9 OPC Server Process forwards these messages to the OPC Server.

[2] Connection of the OPC Server: When the OPC Server connects to the OS-9 OPC Server process, the Machine Control Process need to send information about their available tags to the OPC Server.

[3] Network error: When the OPC Server has disconnected from the control system and connects again special action needs to be taken. It is possible the some information is not allowed to get lost. If this is the case, the machine needs to be stopped when the OPC Server gets disconnected and all information that could not be send needs to be buffered. When the OPC Server connects again all buffered data needs to be sent.

[4] Batch start: When a production batch is started almost all IO's and variables will change.

[5] Alarm and State information: When an alarm occurs, this normally causes a chain reaction of alarms throughout the processes. When alarms occur, information about the states of different parts of the

machine is changed. The information about the changes in states needs to be passed to the OPC Server along with the alarm information.

[6] Configuration: When configuration information is written to the control system, normally a lot of parameters need to be set at once. They consist of groups of recipe parameters.

[7] Operator commands: The operator can send commands to the control system. These commands relate to the operation of the machine or can be used for checking the state of some devices for maintenance or problem searching.

[8] Some OPC Clients perform a read request for all items in a group when the specific group is enabled.

### 6.8.1 Priorities when handling alarms, and start and stop operations

When handling start, and start and stop operations it is not only important to handle all events within a certain amount of time, but also that it does not disturb other data flows that can have a higher priority. Therefore a priority scheme must be taken into account when handling alarm, and start and stop operations.

Different scenarios are possible. One possible scenario is that the control system sends information to the OPC Server that may not get lost. If a network error occurs this information can not be send. Thus the machine should be stopped if possible internal buffers are full or immediately if no buffers are available.

Another scenario is when the control system sends no information to the OPC Server that is not allowed to get lost. In other words: the control system is allowed to run without a connected OPC Server. In this scenario it is important that a connection or disconnection from the OPC Server does not disturb the machine control.

In both scenarios the occurrence of a lot of alarms normally causes the machine to stop. In this case there is a burst of alarms, but the normal data flow will stop. As a result the burst of alarms has no influence on the normal data flow.

### 6.8.2 Required handling times

[1] At start-up, when all processes are started, and the OPC Server connects, the machine control processes all send their list of available items to the OS-9 OPC Server. This results in a burst of about 1425 messages from the Machine Control Processes. Immediately after the tag information messages are sent, the Machine Control Processes send the initial values of all tags to the OPC Server. The result is that about 2850 messages are passed to the OPC Server. Before the OPC Server is connected only a few messages are passed between the OPC Server Process and the Machine Control Processes for registration. The time in which the burst that is caused by the initial connecting of the OPC Server is not very much restricted. The tag information messages are sent like normal data messages and are sent before the normal data flow is started. The amount of data that needs to be sent is bigger than the normal data flow, because information about tags needs to be sent. This information is in the form of several strings. For operator convenience it is advisable that the burst is handled within 3 seconds.

[2] When the OPC Server connects to the control system the OS-9 OPC Server process informs all Machine Control Processes about this. The Machine Control Processes respond with sending information about all available tags the OPC Server Process, which forwards the messages to the OPC Server. This causes a burst of about 1425 messages. If the machine is not running, the situation is the same as under [1]. Another situation occurs if the machine is still or already running. This can only be the case if the machine is allowed to run without a connected OPC Server and thus produces no data that is not allowed to get lost. In this situation the most important thing is that the machine control is not affected by the connecting of the OPC Server. The time in which the burst is sent is not critical, because there is no critical data that may not get lost. For operator convenience it is advisable that the burst is handled within 3 seconds.

[3] When a network error has occurred or the OPC Server stopped functioning for some other reason. The data that is not allowed to get lost must be buffered in the Machine Control Processes. If the buffers get full, the machine needs to be stopped. When the machine is not stopped and the OPC Server connects again, the Machine Control Process need to send information about their available tags to the OPC Server. When these tag information messages are sent, the Machine Control Processes need to send all

buffered data. During the sending of the buffered data, all other data, that is allowed to get lost, does not need to be sent. The buffered data normally holds process values that are used for statistics or that need to be stored for tracking or for some other reason. These values may not be overwritten. This means that a handshaking protocol needs to be used between the machine control process that produces the data and the OPC client that receives the data. The most important is that the data does not get lost. A handshaking protocol ensures this. If the Machine Control Processes buffer this data and thus the machine does not stop immediately when the OPC Server gets disconnected, the OPC Client can read a buffered data from the Machine Control Process by using a handshaking protocol.
The burst that has to be handled by an OPC Server that is reconnecting is the burst that is caused by the tag registration messages. The time in which this is finished is not very important, but may not take too long, because it could cause that the machine has to stop, because the internal buffers for critical data get full. For operator convenience it is advisable that the burst is handled within 3 seconds.

[4] When a production batch is started almost all IO's, state information variables and result variables change. This causes a burst of about 1425 updates. Because the machine is not running when a batch is started, this is not really a burst that can disturb the normal data flow, because there is nearly no data flow. The start-up of a batch can be seen as the start of the normal data flow. The difference is that almost all tag values change as almost the same time. During normal operation the change of tag values is more spread out in time. The following tag value updates cause the normal data flow to start. The time between the initial tag value changes and the start of the normal data flow can be shorter than the time between tag value changes during normal operation. The burst of data and the first normal data changes need to be handled before the second data change occurs. This is not necessary for all data to be handled properly, but it reduces the jitter during the start-up of a batch. Thus the system must be able to handle maximally twice the normal data stream. If the system is capable of doing this for a short period of time the burst requirements for batch start-up are met and the jitter is minimised.

[5] When an alarm occurs, it normally causes a chain of alarms and changes of states. A burst of about 50 alarms and about 50 state information variables can occur as a result of single alarm. The occurrence of an alarm does not necessarily mean that the machine is stopped at once. If it is a so-called soft stop, the machine is stopped slowly and process and statistical information is still valid. This means that the burst of alarms and state information is not allowed to disturb the continuous updates of the normal operation mode. The burst of updates regarding alarms and state information needs to be handled within 1 second.

[6] Recipe parameters are normally only changed before the start of a production batch. Thus, the machine is not running. Before the batch is started all parameters that belong to the recipe that is used during the batch, are sent to the control system. There are about 50 recipe parameters that need to be set. The OPC Client sends the recipe parameters to the OPC Server. The OPC Server forwards the written information to the OPC Server Process. The OPC Server Process forwards the messages to the destination Machine Control Process. The Machine Control Process receives the messages and sends an answer back to the OPC Server process, which forwards it to the OPC Server, which subsequently forwards it to the OPC Client. Because the machine is not running when recipe parameters are set, no requirements exist concerning the disturbance of the normal data flow. For operator convenience the messages need to be handled within the control system within 1 second.

[7] An operator can send commands to the control system. These commands are used for operating the machine. These command come in small bursts of about 10 at a time. The burst must be handled within 1 second for operator convenience.

[8] Some OPC Clients perform a read request for all items in a group when the specific group is enabled. In a worst case scenario, all items are placed in one group and thus a read request is done for all items. The result is a burst of about 1425 read requests and about 1425 answers to the requests. This burst may not disturb the normal data flow. The response time to the requests is not very critical, because all data updates are already sent to the OPC Server and thus the tags in the OPC Server already hold the latest value of the tag in the control system. Because some OPC Clients might wait for the completion of a request the answer messages need to be sent within 3 seconds for operator convenience.

# 7. HARDWARE DESIGN OF THE OPC SERVER FOR OS-9 CONTROLLED SYSTEMS

## 7.1 Introduction

The OPC Server for OS-9 controlled systems consists of several parts of software. These software applications run on at least two hardware platforms. Within a fully configured system the following components exist.

- At least one PC running SCADA software. This is normally a standard PC running Microsoft's Windows NT, Windows 2000 or Windows XP. This PC needs to be equipped with at least one Ethernet network interface card and be able to communicate with the OPC Server. More than one PC, running SCADA software, can connect to the OPC Server.
- One PC running the OPC Server. This can be the same PC as the PC that is running the SCADA software, but can also be running on a separate PC. This PC needs to be equipped with at least one Ethernet network interface card and have the TCP/IP protocol installed.
- At least one system that is controlled by a control system that is based on OS-9. The control system needs to be equipped with at least one Ethernet interface card

## 7.2 Block Diagram

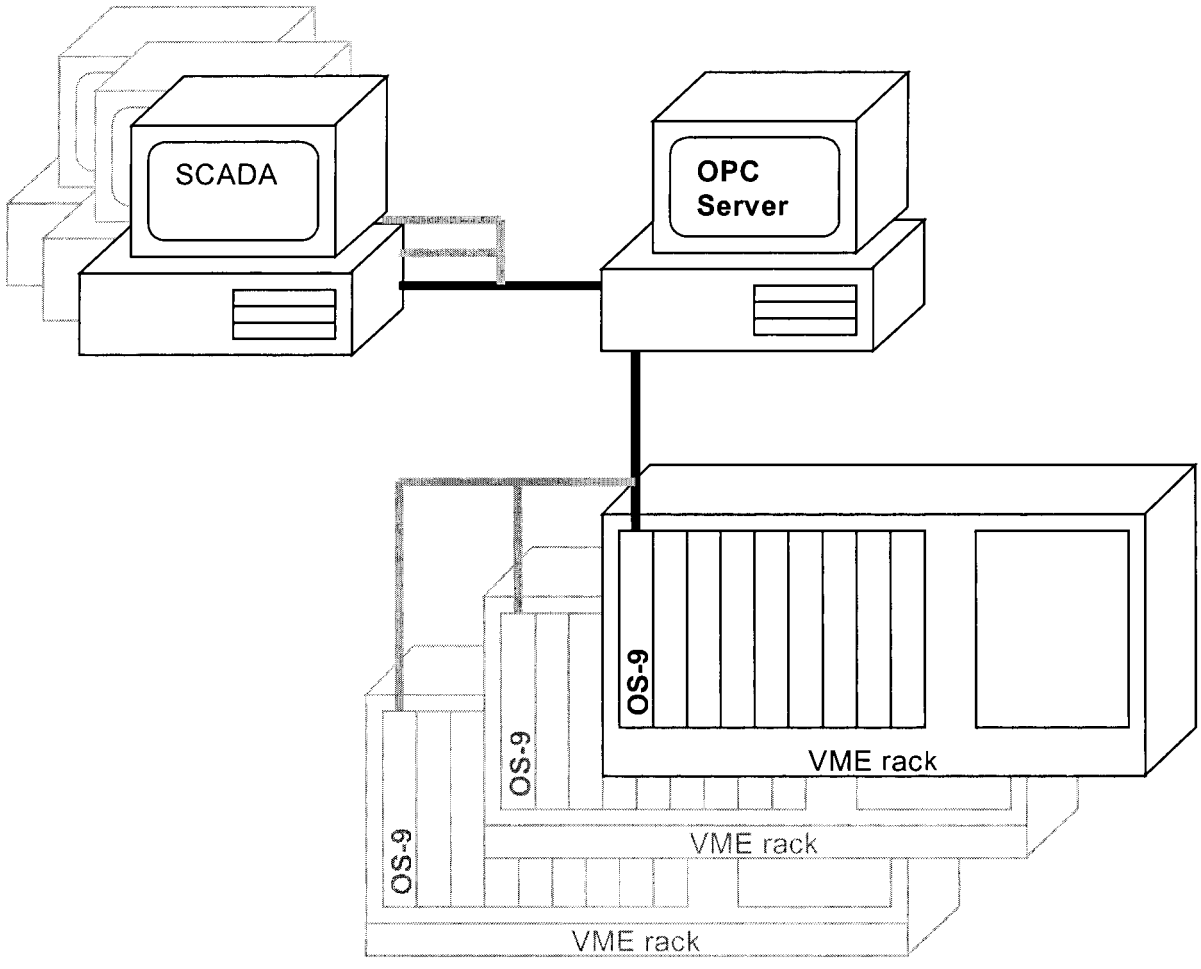


**Figure 7-1: Block diagram of the hardware configuration of the OPC Server for OS-9 controlled systems**

# 8. SOFTWARE DESIGN OF THE OPC SERVER FOR OS-9 CONTROLLED SYSTEMS

The software design of the OPC Server, the OPC Server Process, and the Machine Control Processes is done using UML. UML diagrams are used to explain and clarify the working of the several parts of the software. The diagrams are generated by using an evaluation version of Rational Rose from Rational. The tasks the OPC Server has to perform are explained by using sequence diagrams. Use cases denote the tasks that need to be performed and class diagrams show the base of the objects that are used to perform the various tasks.

## 8.1 System configuration

### 8.1.1 Introduction

The OPC Server for OS-9 controlled systems is build of several parts of software. The OPC server itself runs on a standard PC running Microsoft Windows NT, 2000 or XP. The OS-9 OPC Server Process and the Machine Control Processes run on the control system of the machine.

The OPC Server is implemented using Softing's OPC DA Server toolbox version 3.04. This toolbox provides the OPC DA interfaces, which are used by SCADA systems to communicate with the OPC Server. The toolbox provides an object-based interface for developing the part of the OPC server that provides the data access. Because the interface to the part of the OPC that has to be implemented is object-based, the implementation of the data access part is done with the use of objects.

The OPC server is programmed in C++. The Microsoft Visual C++ 6.0 environment and compiler are used for programming the OPC server. The OPC server is implemented in an executable file, that can be started locally or remotely by the DCOM mechanisms that are provided by Microsoft Windows.

The OPC server communicates with the OS-9 OPC Server Process. This process runs on the control system of the machine. The process handles data requests initiated by the OPC server and sends messages as a result of data changes to the OPC server for updating tag values in the OPC Server.

The OS-9 OPC Server Process communicates with the Machine Control Processes of the control system of the machine on one side and with the OPC Server on the other side. The process is mainly a bi-directional communication buffer between the Machine Control Processes and the OPC Server. It communicates with the Machine Control Processes using named pipes and with the OPC Server using TCP/IP over Ethernet. Figure 8-1 shows an overview of the communication lines in the system.

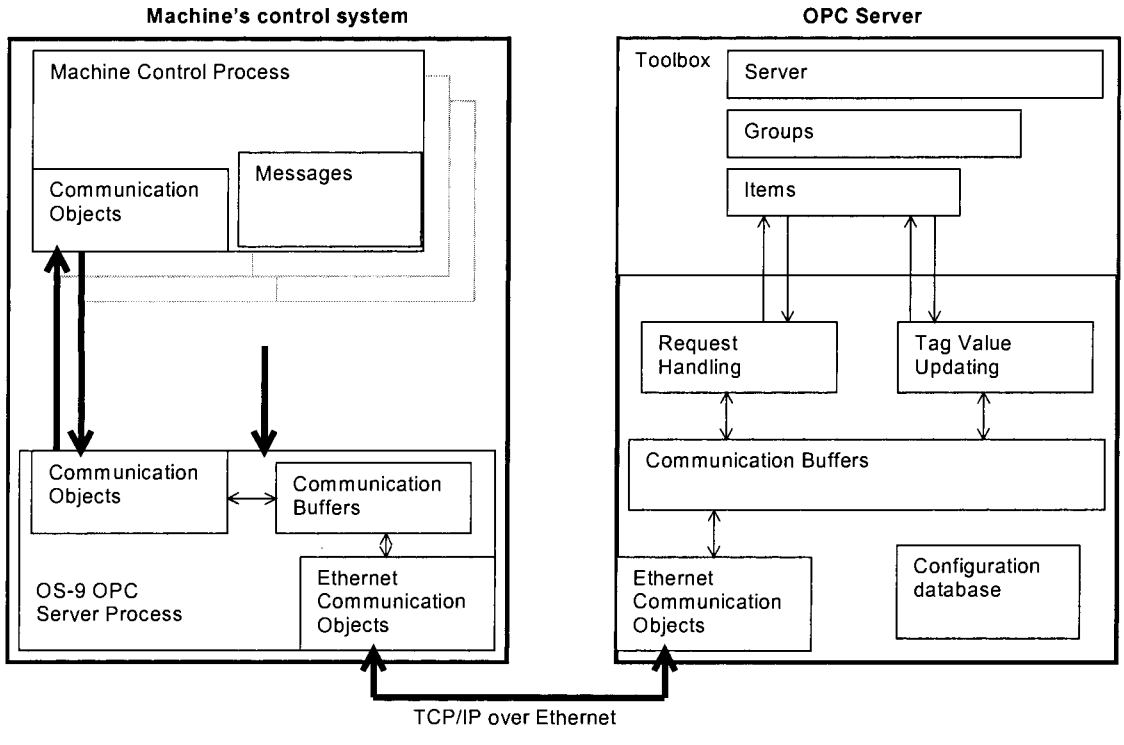


**Figure 8-1: Block diagram of the software configuration of the OPC Server for OS-9 controlled systems**

## 8.2 Design considerations and decisions for os-9 processes

### 8.2.1 Interprocess Communication

#### 8.2.1.1 What Interprocess Communication Mechanisms are Available?

Inside OS-9 there are several mechanisms for inter-process communications. These mechanisms include signals, alarms, events, semaphores, pipes and data modules. All these mechanisms differ in the way they are used and their purpose.

##### 8.2.1.1.1 Signals

Signals can be used for interprocess communication, for communication within OS-9, and for communication between the operating system and processes. A process can subscribe to various signals. When a signal is generated and a process is subscribed to it. The signal is placed in the process' signal queue. When the process in ready to handle a signal, the oldest signal is handled first. A process can set its state to receive or not to receive signals at a certain time by masking and unmasking all signals. When a process has not masked signals and a signal is generated to which the process is subscribed, the signal cause the process to interrupt its current task and jump to the signal handler routine. When the signal is handled, the process continues processing where it was left when the signal occurred. Thus a signal acts as a kind of interrupt.

Several types of operations can generate these signals. The operating system can generate them when certain operations are performed. For example: a signal can be generated when a process writes to a pipe. Signals can also be generated within device drivers. Not only the operating system and device drivers can generate signals, but also any process can generate them.

##### 8.2.1.1.2 Alarms

Alarms are mainly used for generating timeouts. They are created by a process. When a process creates an alarm. It sets up a timeout time and a signal. When the alarm is not deleted before the timeout time has elapsed, the supplied signal is sent to the process.

##### 8.2.1.1.3 Events

Events are mainly used for synchronisation of access to shared resources. An event is mainly an integer value that can be incremented and decremented by a process. The current value of an event can be read and set by a process. Also a process can wait for the value of an event to reach a value between a given minimum and maximum value. When a process is waiting for an event, the process is activated when the value of the event gets between the minimum and maximum value supplied to the waiting function.

##### 8.2.1.1.4 Semaphores

A semaphore is basically a very limited event. It only has two values. Its values cannot be read. The only operations that are possible on semaphores are 'acquire exclusive access' and 'release exclusive access'. A semaphore is a very fast implementation for synchronising access to shared resources.

##### 8.2.1.1.5 Pipes

A pipe is a First In First Out queue that can be used to send data from one process to another. There are two versions of pipes, named pipes and unnamed pipes. Unnamed pipes can be used to start a process with its standard output and/or input going to or coming from pipes. Unnamed pipes are created using the default path numbers to standard out and standard in.

Named pipes are created like normal files except that they are created in virtual pipe directory on the file system, which is '/PIPE' on an OS-9 system. Almost all operations that are possible on normal files are

possible with pipes, except for operations that position the file pointer somewhere else in the pipe than at the beginning. This is because a pipe is a FIFO queue.

When a process writes data to a pipe, further write access to the pipe is denied for all other processes until the write is completed. When another process wants to write to a pipe while a write operation is in progress, the process blocks. It continues with writing to the pipe when the first write operation is finished. Access to pipes is controlled inside the OS-9 kernel.

When a process wants to write data to a pipe, but there is not enough free space in the pipe. The write operation blocks until some other process has read enough data from to pipe to fit all data in the pipe.

#### 8.2.1.1.6 Data Modules

Data modules are a piece of memory that can be shared among processes. The operating system does not provide automatic synchronisation of access to the data module nor does it have any restrictions for the data that is placed in them. Thus synchronisation needs to be performed used other mechanisms such as events and semaphores.

### 8.2.1.2 What Interprocess Communications can be used?

There are two interprocess communications mechanisms that are suited for transferring data between processes. These are pipes and data modules. Both mechanisms have advantages and disadvantages for transferring data between processes.

Pipes have automatic protection of blocks of data. Write operations are completed before another write operation can be performed. If a pipe is large enough, different processes can write data to the pipe sequentially before data is read. Because a pipe acts as a FIFO queue, the oldest data is always read first. When a pipe is full a write operation blocks the process that performs the write.

Data modules have no built in protection mechanisms. All data that is in the data module can be randomly accessed. A process can directly read or write the memory where the data module is located. This makes access to the data very fast. Other interprocess communication mechanisms need to be used to synchronise access to the data module.

### 8.2.1.3 What Interprocess Communication is Necessary?

The biggest stream of data regarding the OPC Server that will flow through the processes within the control system consists of value updates for the tags in the OPC Server. The machine control processes generate these data. The OPC Server process needs to forward this data to the Ethernet network to the OPC Server. The generated messages have different priorities and the messages with the highest priority need to be handled first.

When the control systems starts up, the machine control processes need to communicate with the OPC Server process to negotiate and set up different identifiers. These operations are only performed at start up and when the OPC Server application connects to the OPC Server process.

A third stream of data is generated by the OPC Server and is send over the Ethernet network to the OPC Server process. These data consists of data write messages and device read requests. The OPC Server process receives these messages and forwards them to the machine control processes.

### 8.2.1.4 What Different Implementations of Interprocess Communications bring about

#### 8.2.1.4.1 How Pipes can be used

When pipes are used for interprocess communication this brings about that not every message can be written to the pipe separately, because this would have a dramatic effect on the throughput of the system. This is shown in section 8.2.1.5. This implies that a buffer needs to be filled with data from multiple messages and subsequently be written to the pipe. The write operation to a pipe is basically a memory copy that is

performed by the internals of the OS-9 kernel. When data is read from a pipe it is copied to a receiving buffer. This again is an internal memory copy.

The main data stream needs to be forwarded from the pipe to the drivers for the Ethernet network card in the control system by the OPC Server process. The throughput of communication over Ethernet using TCP/IP is a lot faster with larger messages than with small messages, because the overhead generated by headers is a lot larger with small messages. When complying with the network standards, the data needs to be sent over the network in big endian mode. Because the control system is running on a 68k processor, which is also a big endian processor, no marshalling needs to be performed on this side of the Ethernet communication. Thus data, holding value updates for tags in the OPC Server, can be forwarded from the pipe to the Ethernet network card driver without modification of the data by the OPC Server process.

Thus using pipes, the largest stream of data needs to be copied in memory four times within the control system. First it has to be copied from the original message to a buffer, this buffer is written to the pipe, subsequently the data is read from the pipe into a buffer, and finally this buffer is copied to the Ethernet driver.

#### 8.2.1.4.2 How Data Modules can be used

When data modules are used, the data can be written to the memory location of the data module directly. In order to keep the order of the messages intact the data module needs to be encapsulated in a class that behaves like a FIFO queue.

There are two different solutions for forwarding the data in the largest data stream from the data module to the network. The first solution is to copy the data from the data module into a buffer and subsequently writing this buffer to the Ethernet network driver. The second solution is to design the encapsulating class in such a way that all data in the data module is always ordered in a successive way. This means that the tail of the data always needs to be further in the data module than the head, so it is not possible to use the buffer in a cyclic way. When the encapsulating class guarantees this ordering of data, the data can be copied from the data module to the Ethernet driver directly.

Thus using data modules, the largest stream of data needs to be copied in memory three or two times depending on which solution is used.

In addition to implementing a FIFO queue and possibly preserving the data ordering, the encapsulating class also needs to synchronise all access to the data module.

#### 8.2.1.4.3 What Interprocess Communication is used?

Although data modules have higher throughput, pipes are used. Pipes are designed to communicate between processes in a FIFO way. This is exactly what is necessary in this application. Synchronisation and access control is build into the kernel and does not need to be designed.

When the performance of pipes is too low, data modules and a specialised class that needs to be designed can replace their functions. In order to make it relatively easy to replace the functionality provided by pipes with a data module, all operations on pipes are encapsulated in a class which functions as a base class for other classes. When the decision is made to remove the use of pipes from the system and use data modules instead, only the class that handles the pipes needs to be replaced by a class that handles the data modules.

### 8.2.1.5 Performance of Interprocess Communication using Pipes

In order to be able to estimate the maximum throughput of the system and to calculate the time that is necessary to write the data to a pipe, performance tests were performed. These tests were performed using different sizes of pipes and different sizes of messages. The results of the tests can be seen in Figure 8-2 through Figure 8-5.

These tests where performed with only two running processes. One process writes data to the pipe and another process reads data from the pipe. The process did not allocate new memory for each read or write operation. Continuously the same memory buffers where used. No real data was generated. Thus the figures show the maximum throughput for the system. When the processes have to create the data and put the

resulting messages into buffers, the overall throughput will be much lower. The figures can not be seen as a measurement of the possible resulting throughput in the final system. However, they do show the differences in throughput with different sizes of pipes and messages. The higher the maximum possible throughput with chosen sizes the higher the actual possible throughput in the system.

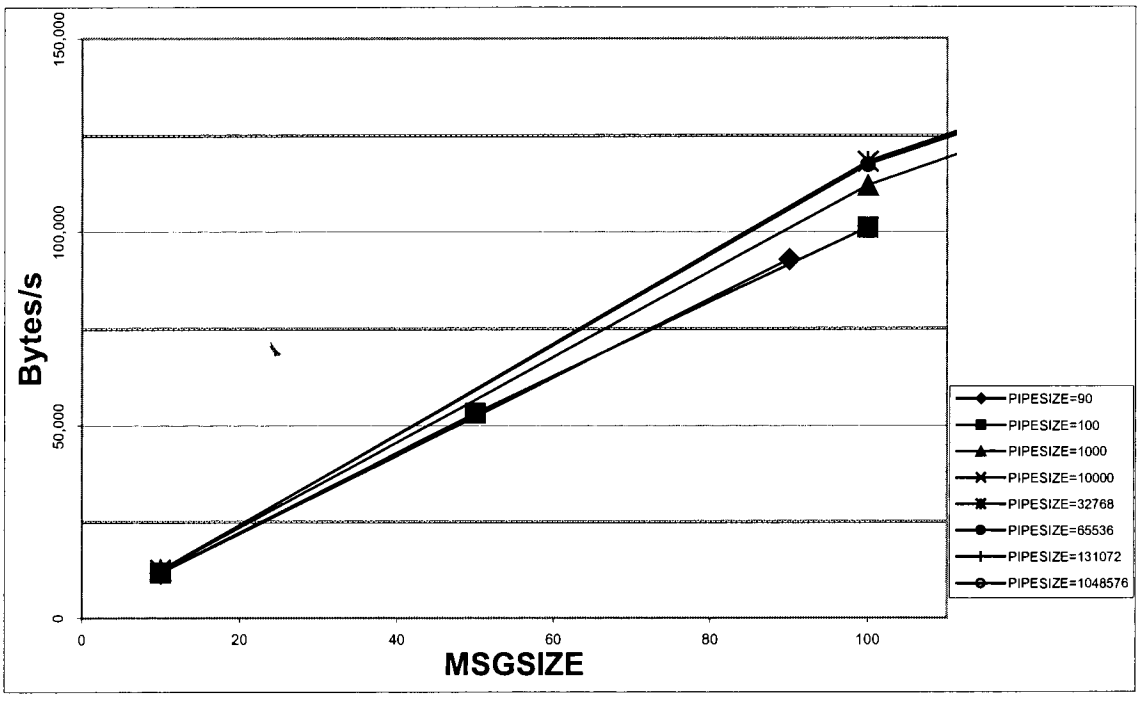


**Figure 8-2: Pipe performance (I)**

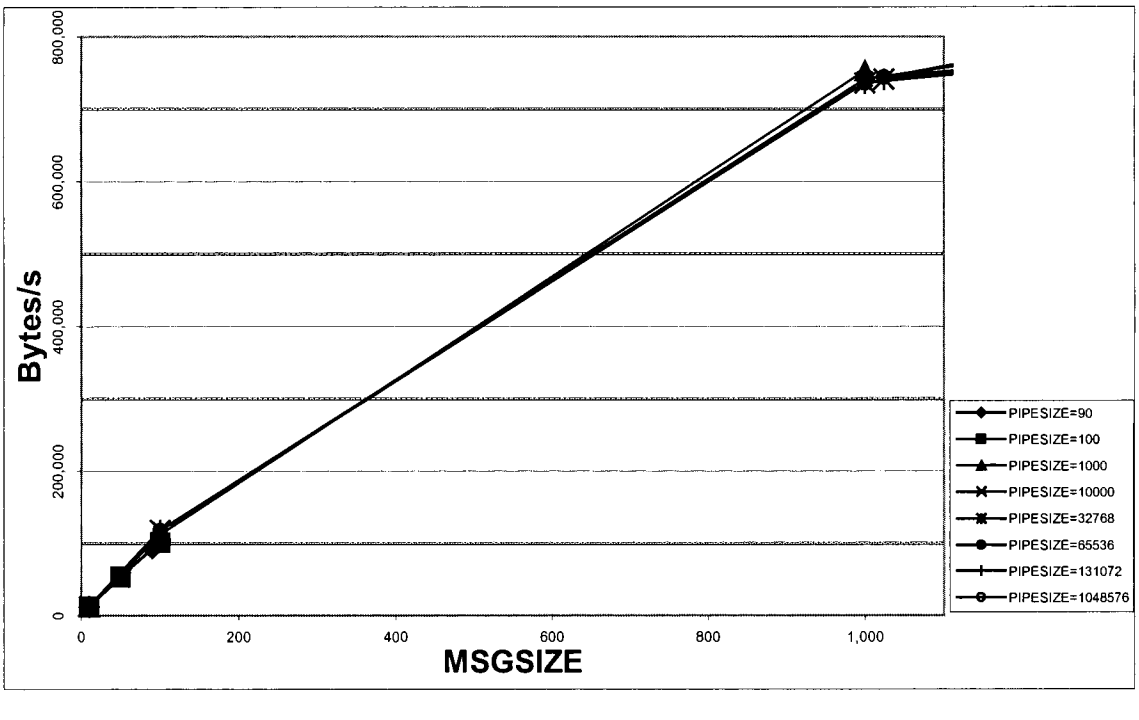


**Figure 8-3: Pipe performance (II)**

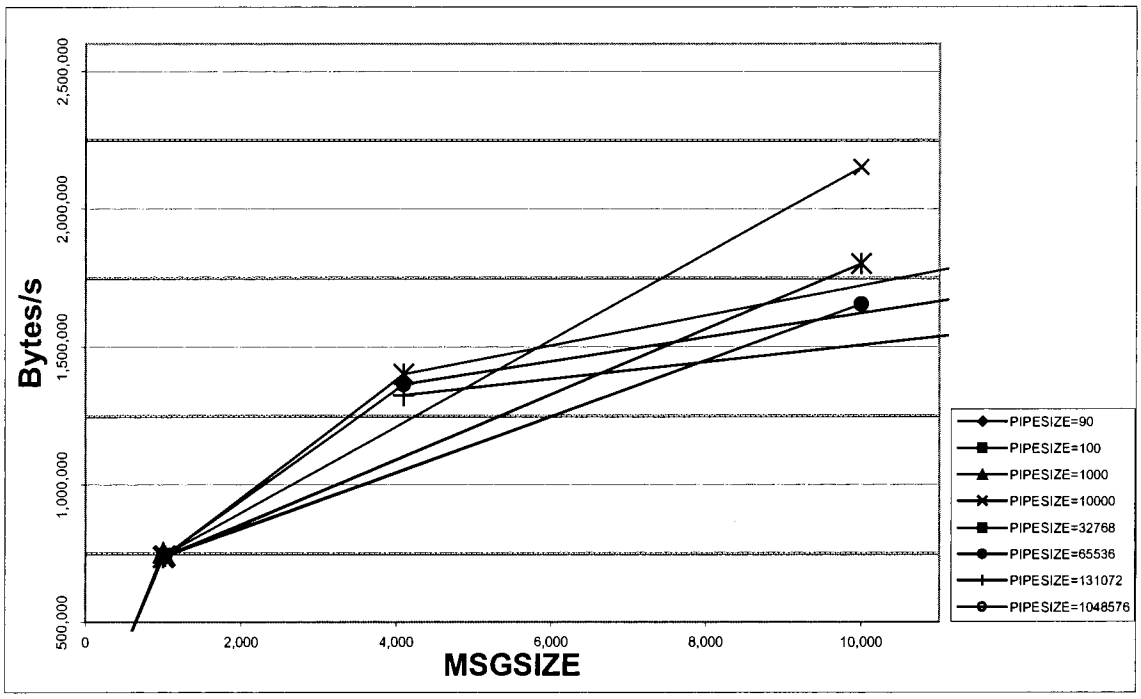


**Figure 8-4: Pipe performance (III)**

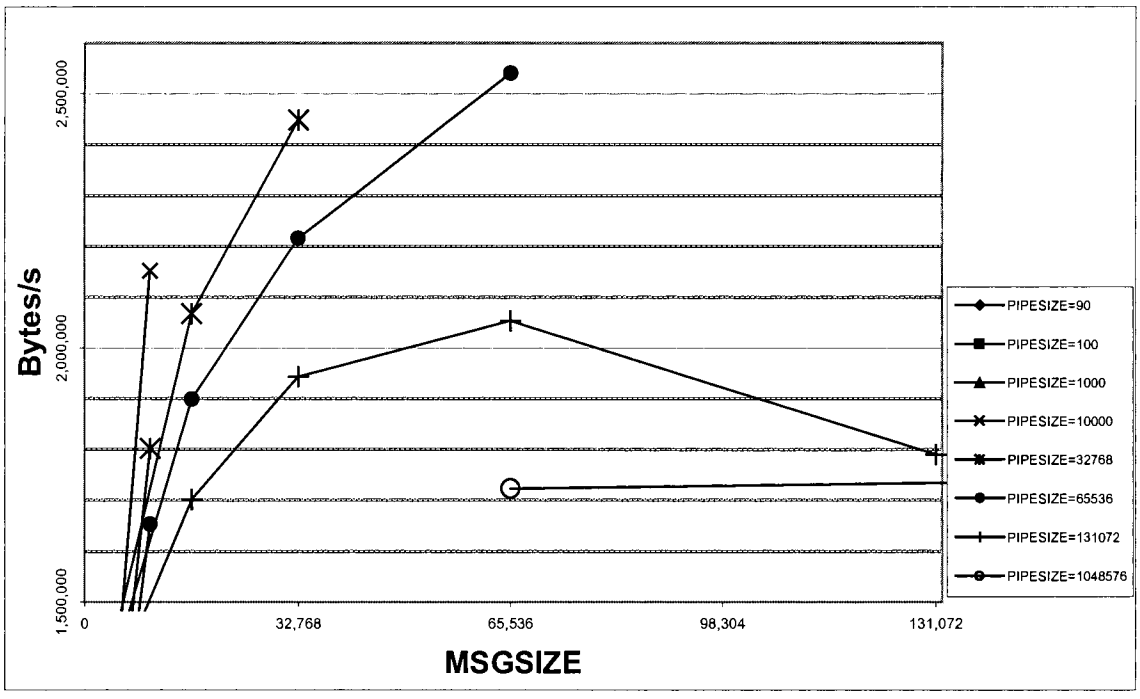


**Figure 8-5: Pipe performance (IV)**

Those figures show different parts of the same graph. The first figure shows the leftmost part of the graph. This part of the graph is the result of tests with small pipes and small messages. The last figure shows the right part of the graph. The rightmost part is not shown in the figures, because the performance was decreasing with even bigger pipes and that sizes of the pipes is too big for the system. The last figure shows the pipe sizes that give the highest possible throughput. As can be seen from this figure, the pipe size and the message size have influence on the performance. The highest performance can be reached with a pipe size of 64 KB and a message size of 64 KB.

Although the highest throughput for messages through pipes can be reached by with a pipe and message size of 64 KB, it is not very likely that messages of this size will be used in the system. Combining a lot of small messages into one larger message with a size of about 64 KB takes a relatively large amount of time. Because the combining of the messages is done in the machine control processes where the data messages are generated, it is not allowed that combining messages takes a long time. If a specific process is busy combining messages is can not perform machine control tasks at the same time. The result could be that the machine's response to external events is to slow. Thus a compromise between throughput and blocking processing time has to be made.

Another reason for not using messages with a size of 64 KB is that most messages have to be forwarded from the pipe to the network. The maximum size of messages that can be send to the network is 32 KB in OS-9. When messages with a size larger than 32 KB are sent through the pipes, the OPC Server process has to process the data in the message and split it into multiple smaller messages. When the messages are smaller than 32 KB they can be forwarded without further processing. This improves the overall performance of the system.

### 8.2.2 Implementation of FIFO queues in OS-9 processes

A FIFO queue is very well suited to function as a buffer for the messages that are send by machine control processes. The created messages have different priorities. For each priority a separate FIFO queue is created. When is message is ready to be sent, it is appended to the queue that corresponds with its priority. When the process is ready to send the messages that are in the queue to the interprocess communication pipes, the messages in the queue with the highest priority are sent first. This mechanism automatically sorts the messages by priority and creation time.

A very common implementation of a FIFO queue is to organise the items in the queue in a singly linked list. When an item is appended to the queue a link element is created and appended to the tail of the list. When the first element of the queue is removed from the list the head of the list is set at the next element and the old first element is freed.

#### 8.2.2.1 Problems with OS-9 and FIFO Queues

Experiments with FIFO queues that are based on a linked list show that OS-9 loses its real-time behaviour, when the memory gets fragmented. When many fragmentations exist in a process' memory, an operation that frees a block of memory can last relative long. When several ten thousands of holes exist a free operation can last up to 500 times the time that is needed in a situation where the memory is not fragmented.

##### 8.2.2.1.1 Problems with fragmented memory and OS-9

In order to further investigate the problem with freeing memory when the memory is fragmented a simple test program was used. When this program is started it first allocates some small blocks of memory. Pointers to these memory locations are stored in an array. Subsequently the program frees all blocks, whose pointers are on the even locations in the array. These free operations have are executed with high performance. No problems occur with these free operations. The last part of the program frees all memory locations that are still allocated. The pointers to these locations are on the odd positions in the array. These free operations last significantly longer than the first free operations. The first free operations of the second group of free operations take significantly more time than the last free operations in this group. When almost all free operations of the second group are executed, the memory very much fragmented anymore, because almost all blocks are already freed. Figure 8-6 shows the order in which the memory operations are performed. The different states in which the memory can be during the execution of the test program are shown. In Text 8-1

you can see the main part of the test program. For readability the routines that perform time measurements are left out. The program consists of three loops. In the first loop all memory allocation is done. The second loop frees all even memory blocks. The third loop freed all odd memory blocks. Timing this loop and comparing the result with a timed second loop shows the described problems.

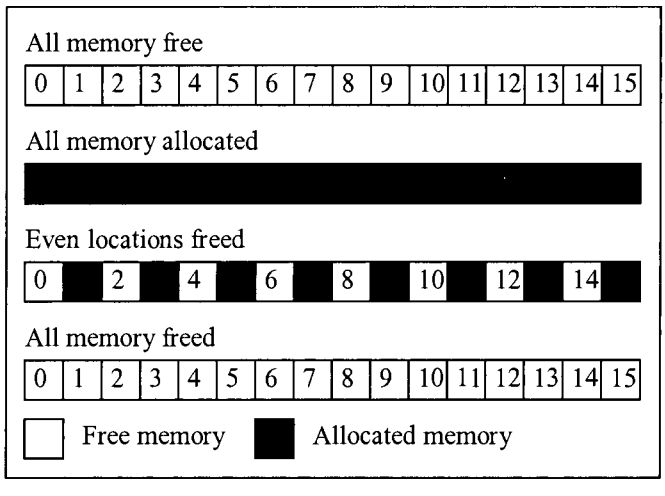


**Figure 8-6: Sequence of memory operations in the test program**

```
int i;
void a[NUMBER_OF_MEMORY_BLOCKS];

// Allocate all memory blocks
i = 0;
while (i < NUMBER_OF_MEMORY_BLOCKS)
{
        a[i] = malloc(MEMORY_BLOCK_SIZE);
        i = i + 1;
}

// Free all even locations
i = 0;
while (i < NUMBER_OF_MEMORY_BLOCKS)
{
        free(a[i]);
        i = i + 2;
}

// Free all odd locations
i = 1;
while (i < NUMBER_OF_MEMORY_BLOCKS)
{
        free(a[i]);
        i = i + 2;
}
```

**Text 8-1: Sample code to show the problems that OS-9 has with memory fragmentation**

#### 8.2.2.1.2 Why FIFO queues suffer from the memory fragmentation problem

When using multiple FIFO queues, which work with a linked list, memory fragmentation is likely to occur. When messages are generated in a random order concerning their priority, the allocated memory gets randomly divided over the FIFO queues. When subsequently all messages in one queue are removed, and thus the memory of the linking elements is freed, the memory fragmentation is a fact. This behaviour is dramatic for the predictability of the behaviour of the machine control processes. Thus it is not allowed to use FIFO queues that work this way.

Figure 8-7shows an example of memory fragmentation caused by the use of FIFO queues. The first part of the figure shows the starting situation, where all memory is free and the queues hold no items. The next figure shows the situation, where items are added to the queues in a random order. All memory is allocated. Each time an element is added to a queue, a link item is created within the queue. In this figure the first item is added to queue 2, the second to queue 1, the third to queue 2, etc.. Subsequently all items are removed from queue 0. Together with the removal of the items from the queue the linking elements are freed from memory. This causes memory fragmentation to occur. The last figure shows the situation where the second queue is also emptied. In this situation the memory fragmentation can be better or worse than when only the first queue is emptied. This is the result of the memory optimisation performed by the operating system after each free operation. The memory management system combines successive free blocks into one bigger free block of memory.

#### 8.2.2.1.3 What causes the performance decrease when the memory is fragmented

The OS-9 memory manager holds a double linked list of free memory locations. This is the case at two different points. The operating system holds a global list of memory blocks that are assigned to processes. Globally free blocks are stored is a double linked list that holds all free memory locations in the global memory. Within each block of memory that is assigned to a process, a double linked list of free memory locations is maintained for the memory inside the memory block. [OS9 INS]

Each time a block of memory is freed, the memory manager checks if the memory just before and/or just after the block of memory is free. If this is the case, the memory manager combines the memory blocks into one bigger block of free memory. When the list of free memory locations is very big, because of many memory fragmentations, searching the neighbour memory blocks can take a very long time. Thus the memory optimisation takes a long time and thus a free operation also takes a long time.

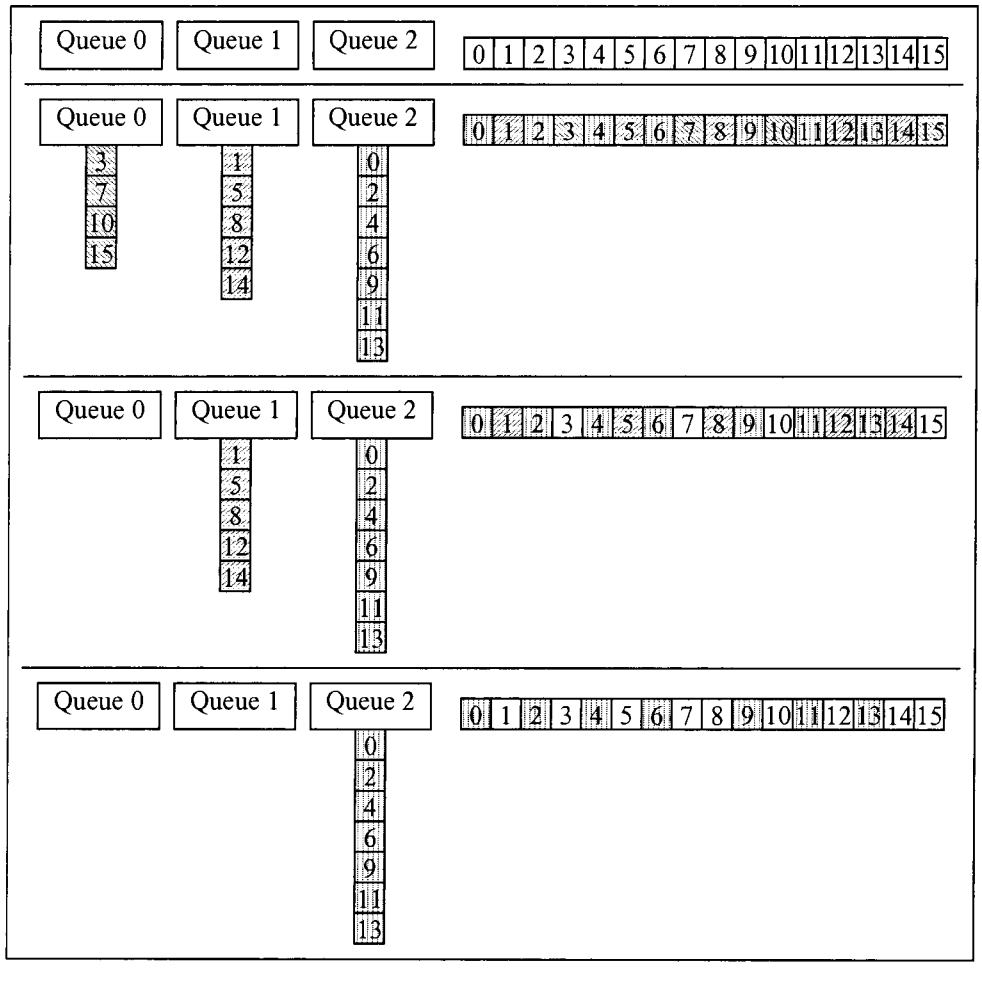


**Figure 8-7: Memory fragmentation caused by the use of FIFO queues**

#### 8.2.2.2 Alternative Solutions for FIFO Queues

In order to be able to use FIFO queues for storing the messages that need to be sent to the OPC Server process, the FIFO queues must be implemented in another way.

##### 8.2.2.2.1 FIFO Queues with a Fixed Maximum Size

Instead of using a linked list to store the items in a FIFO queue, they can also be stored in an array. With this solution a head and tail pointer move though the queue indicating where the data is located. The array in such a FIFO queue has a fixed size, and thus there is a maximum number of items that can be stored in the queue.

##### 8.2.2.2.2 FIFO Queues with a Variable Size Array

It is also possible to use a queue with a variable size array. With this solution the array is resized when the existing array is full and data is appended to the list. A disadvantage of this solution is that when an array is enlarged in memory it is often not possible to leave the array at its original location and thus all data needs to be copied to the new location. Additionally, it is very likely that when the queue is full, the head is not at position zero and thus the tail not at the end of the array. When this is not the case all data from the beginning of the array up to the tail pointer must be copied to the newly allocated memory in order to preserve the successive ordering of the items in the array. The result is that enlarging the array can take very long time and thus the real-time operation of the process is not assured.

#### 8.2.2.2.3 FIFO Queues with Multiple Fixed Size Arrays

Another solution is to use multiple fixed size arrays with a FIFO queue. Initially one array is created. When the array is full and an item is appended to the list, a new array is created and the appended item is placed in the new queue. All items that are subsequently added to the queue are placed in the new array. When more items are appended and the second array gets full, a third array is created. The newly created arrays are placed in a FIFO queue themselves. When the arrays are large enough, it is very unlikely that many arrays need to be created and removed, and thus it is no problem to implement the outer FIFO queue as a linked list. When items are removed from the queue, which always happens at the head of the queue, items are removed from the first array in the list. When all items are removed from the first array, the array is freed and the head pointer is placed at the beginning of the second array.

### 8.2.2.3 The Implementation of the FIFO Queues that is used

The FIFO queues that are used in the OPC Server process and the machine control processes that are used are implemented as a FIFO queue that is based on an array with a fixed size. This is possible, because it is known how many items of each priority are available in each process and thus the maximum size of the arrays is known.

The best way to implement the machine control processes is to create all messages, which are used for sending item data to the OPC Server, when the process is started. When an item's value changes the data value of the corresponding message is updated and a pointer to the message is placed in the right FIFO queue. When the process is ready to send the messages in the queues, the data is copied from the messages in the queue to a buffer and send to the pipe. The pointers to the messages are removed from the queues, but the messages stay in memory. When an items value changes again, it is not necessary to create a new message, but it is only necessary to update the data in its corresponding message, that is still in memory, and at the pointer to the message to the right queue again. This approach reduces the number of memory allocation and free operations a lot.

## 8.3 Messages in the system

The different parts of the OPC Server communicate with each via different communication channels. Inside the control system the communication between the processes is performed using pipes. The communication between the control system and the OPC Server is done over an Ethernet network.

### 8.3.1 The paths that messages follow in the system

The communication between the machine control processes and the OPC Server process takes places via pipes. These pipes behave like FIFO queues. The OPC Server process is the first process that has to be started within the system. It creates a pipe for each possible priority of tag value update messages. These pipes are called Priority Message Pipes. The messages that are sent through these pipes are sent to the OPC Server when the value of a data provider changes. The OPC Server process also creates a pipe that is used by the machine control processes for sending messages to the OPC Server process. This pipe is called the Connect Pipe, because the machine control processes meanly use it when they build a connection the OPC Server process.

Besides the communication with the machine control processes, the OPC Server process also has to communicate with the OPC Server. This communication takes place via TCP/IP over Ethernet. The OPC Server process acts as a server and the OPC Server as a client. When the OPC Server process has created the pipes and events, it also sets up a server socket for TCP/IP communication.

Within the OPC Server process one object is created for handling all data that comes from the pipes that are created for the tag value update messages. Another object is created for handling all messages that come from Connect Pipe and one object is created for handling all incoming and outgoing messages to and from the OPC Server.

When the OPC Server process has created all objects for handling the pipes and the network, the other processes can be created. When a machine control process starts, it creates an internal list of tags and the

accompanying messages. It also creates an object for sending messages to the Priority Message Pipes. This object connects to the pipes that are created by the OPC Server process. This object holds internal FIFO queue buffers per priority. All that are to be sent to the Priority Message Pipes are stored in these buffers. At certain times the machine control process sends all messages in one or more of the FIFO queues to the related pipe. When a machine control process has set up the commutation object for the Priority Message Pipes, it creates an object for sending messages to the Connect Pipe. This object also holds a FIFO queue and connects to the existing Connect Pipe. Subsequently the machine control process creates an object that creates a pipe for receiving messages from the OPC Server process. This pipe is called a Message Pipe. Now all communication objects for the machine control process are created.

The machine control process now has to establish a connection to the OPC Server process. The first thing the machine control process does is to create a message with its process name and the name of the message pipe it created in it. This message also holds the number of tags the process provides to the system and is called a Connect Message. This message is send to the Connect Pipe and received by the OPC Server process. When the OPC Server process receives a Connect messages, it knows that a new process has started and wants to be able to provide its data to the OPC Server. Before this can be done the OPC Server process needs to be able to send messages to the machine control process. Therefore it creates an object that connects to the Message Pipe. Before the provided tags can be accessed in an efficient way they need to be assigned a unique number inside the control system. This is done by assigning the numbers that just follow upon the maximum number assigned so far to the newly added tags. Thus when no tags are registered, the new tags are numbered from 1 to the number of tags provided. In following assignments the tags are assigned the maximum assigned number plus 1 to the maximum number assigned plus the number of tags provided. The OPC Server process holds a list of registered processes. Items in this list hold the name of the process, the name of the Message Pipe of the process and the minimum and maximum assigned tag identifier. Now the OPC Server process can send a message to the machine control process with the minimum and maximum assigned tag number. The machine control process can assign all numbers in between this minimum and maximum to the tags.

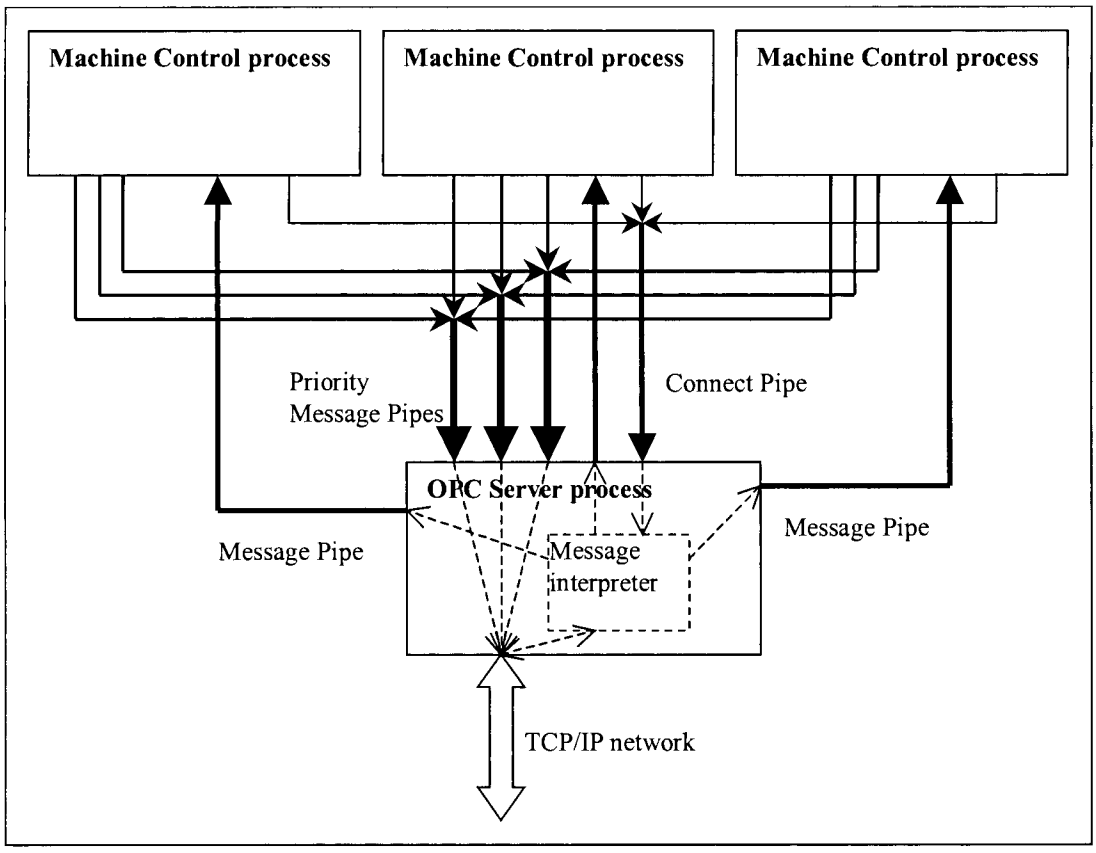


**Figure 8-8: Example of the pipes in that are used in the control system**

All machine control processes that provide data for the OPC Server perform this procedure. When all processes are ready setting up their communication with the OPC Server process and their tags are registered the OPC Server has to wait until the OPC Server connects to it via the network. In Figure 8-8 you can see an example of the pipes that are created in the control system when there are 3 machine control processes. When an OPC Server connects to the OPC Server process, this process accepts exactly one connection. When an OPC Server is connected the OPC Server process sends a message to all machine control processes to order them to send their tag information to the OPC Server. These tag information messages hold the name of the tag, its path, the process' name, the data type, and the assigned tag number.

The OPC Server Process does not interpret message that come from the Priority Message Pipes. Information from these pipes is read into a buffer and directly sent to the OPC Server via the network. The OPC Server interprets these messages. Therefore, the machine control processes can send the tag information messages to the Priority Message Pipes. The OPC Server process forwards these messages to the OPC Server, where they can be registered. When the OPC Server has received all tag information messages, it can identify each individual tag in the control system by its unique number. Now the machine control processes can send a message to the OPC Server when its data is updated inside the machine control process. The OPC Server can write the value to the right tag.

The OPC Server sets up a connection to every machine that is in its configuration database. Per connection a communication buffer, a request list, and a tag registration array are maintained. The communication buffer is used to store message before they are send across the network to the control system and for storing the received message from the control system. The request list holds a list of all not yet completed requests that are made by the internals of the toolbox.

### 8.3.2 Types of messages in the system

Several types of messages can be distinguished in the system. Messages have different sources and destinations. The following sections describe the messages that are used by the various parts of the OPC Server for OS-9 Controlled Systems.

#### 8.3.2.1 Messages between Machine Control Processes and the OS-9 OPC Control Process when connecting

When the control system is started, the OPC Server Process has to be started before the Machine Control Processes are started. This enables the Machine Control Processes to start the tag registration sequence immediately when they start. The OPC Server Process does not need to known anything about the Machine Control Processes when they start. All information about the names of the processes, names of the pipes, and information about the number of tags are provided by the Machine Control Processes.

While the control system is starting up and the Machine Control Processes are negotiating with the OPC Server Process, several types of messages are being sent between the processes. Table 8-1 and Table 8-2 show the messages that are sent when the connection between the OPC Server Process and the Machine Control Processes is being established. During the connection phase, the OPC Server Process builds an internal registration database with information about all Machine Control Processes. The information that is stored is the name of the process, the name of the Message Pipe of the process and the minimum and maximum assigned tag number. This information is used when a message arrives from the OPC Server. The OPC Server Process can determine to which process the message has be forward by comparing the information in the message with the information in the database.

The first thing the OPC Server Process does when it starts up is creating the Connect Pipe and the Priority Message Pipes together with the objects that are used for communicating via the pipes. These pipes are described in the introduction of this chapter and a graphical representation of their use can be seen in Figure 8-8. After the OPC Server Process has created the pipes, a server object for network communication is created. Subsequently the OPC Server Process creates a process registration object. Now the OPC Server Process starts waiting for Machine Control Processes or the OPC Server to connect to it.

When a Machine Control Process starts up, it creates the objects that are used for communicating via the Connect Pipe and the Priority Message Pipes. Subsequently it creates its Message Pipe and the object that is used for communicating through that pipe. When the communication objects are created, the Machine

Control Process creates a tag registration object and a message and registration entry for each tag the Machine Control Process provides. The registration entries are placed in the tag registration object and the messages are associated with the registration entries. The created messages are used for sending tag values to the OPC Server. They are created during start up and are not deleted during run time. They are reused every time a tag value update message is sent to the OPC Server.

When the Machine Control Process has finished creating all objects, it creates a connect message. This message is sent to the OPC Server Process. The connect message holds information about the process, the created Message Pipe, and the number of tags. The OPC Server receives the connect message and creates an entry in its process registration database. Subsequently it creates an object that is used for communication trough the Message Pipe that is created by the Machine Control Process that sent the connect message. The OPC Server Process can now assign free tag id's to the tags in the Machine Control Process. The minimum and maximum assigned tag id's are stored in the associated process registration entry in the process registration database. Subsequently a tag id message is created with the minimum and maximum assigned tag id's in it. This message is sent back to the Machine Control Process that sent the connect message.

The Machine Control Process receives the tag id message and uses it to assign an id to all tags in the Machine Control Process. The registration process is now ready for the specific Machine Control Process and the process waits for a registration message to arrive. This message indicated that the OPC Server has connected to the OPC Server Process. The tasks that are performed when a registration message arrives are described in the next section.

Figure 8-9 through Figure 8-12 show the sequences that are executed during the connection phase of the Machine Control Processes and the OPC Server Process in the control system. Text 8-2 through Text 8-5 give a brief explanation of these sequence diagrams.

Table 8-1: Messages from Machine Control Processes to the OS-9 OPC Control Process when connecting

| Message type | Description | Data in message | Purpose |
|---|---|---|---|
| CmessageConnect | Connection notification message | - Number of tags the Machine Control Process provides<br>- Name of the connecting process<br>- Name of the pipe the Machine Control Process reads from | Notify the OPC Server Process that a Machine Control Process has connected |
| CmessageTagInfo | Information about a tag | - The name, path and process name of the tag<br>- The assigned tag ID<br>- The data type | Inform the OPC Server about the ID that is assigned to the tag and to check the data type. This message is forwarded to the OPC Server |

Table 8-2: Messages from the OS-9 OPC Control Process to the Machine Control Processes when connecting

| Message type | Description | Data in message | Purpose |
|---|---|---|---|
| CmessageReg | Start sending registration information | None | The OPC Server Process tells the Machine Control Process to start sending CMessageInfo messages for all tags when the OPC Server connects to the OPC Server Process |
| CmessageTagId | Information about available Tag ID's | - The minimum and maximum available Tag ID's | The OPC Server Process responds to a CmessageConnect with this message. The Machine Control Process can assign Tag ID's to all available tags. |

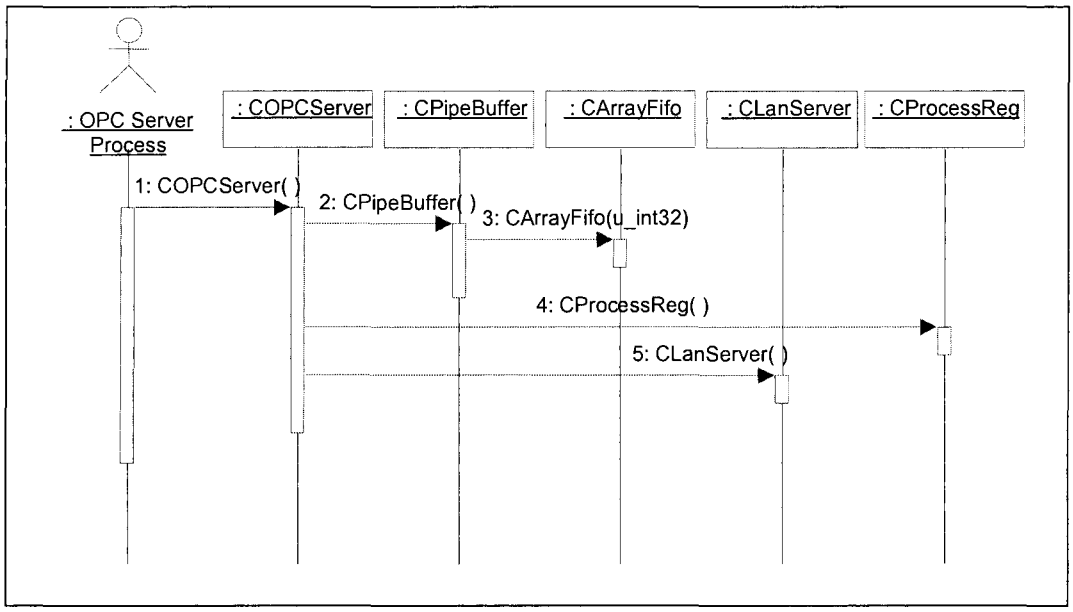


**Figure 8-9: Sequence Diagram of the OPC Server Process starting up**

1: When the OPC Server Process starts it creates a global object: the OPC Server Application

2: When OPC Server object is created, some buffers that are used for communicating through the pipes are also created. Two buffer objects are created:

- One for handling the messages that are to be received through the Priority Message Pipes.
- One for handling the messages that are to be received through the Connect Pipe.

3: When the buffers are created, also the FIFO queues that are used by the buffers are created.

4: A object is created for storing information about processes that are connected to the OPC Server Process.

5: When the objects that must be available when setting up the internal communication in the control system are created, an object is created for handling all communication with the OPC Server over the network. The OPC Server Process acts as a server for the OPC Server. The OPC Server is a client of the OPC Server Process.

**Text 8-2: Description of the sequence diagram in Figure 8-9**

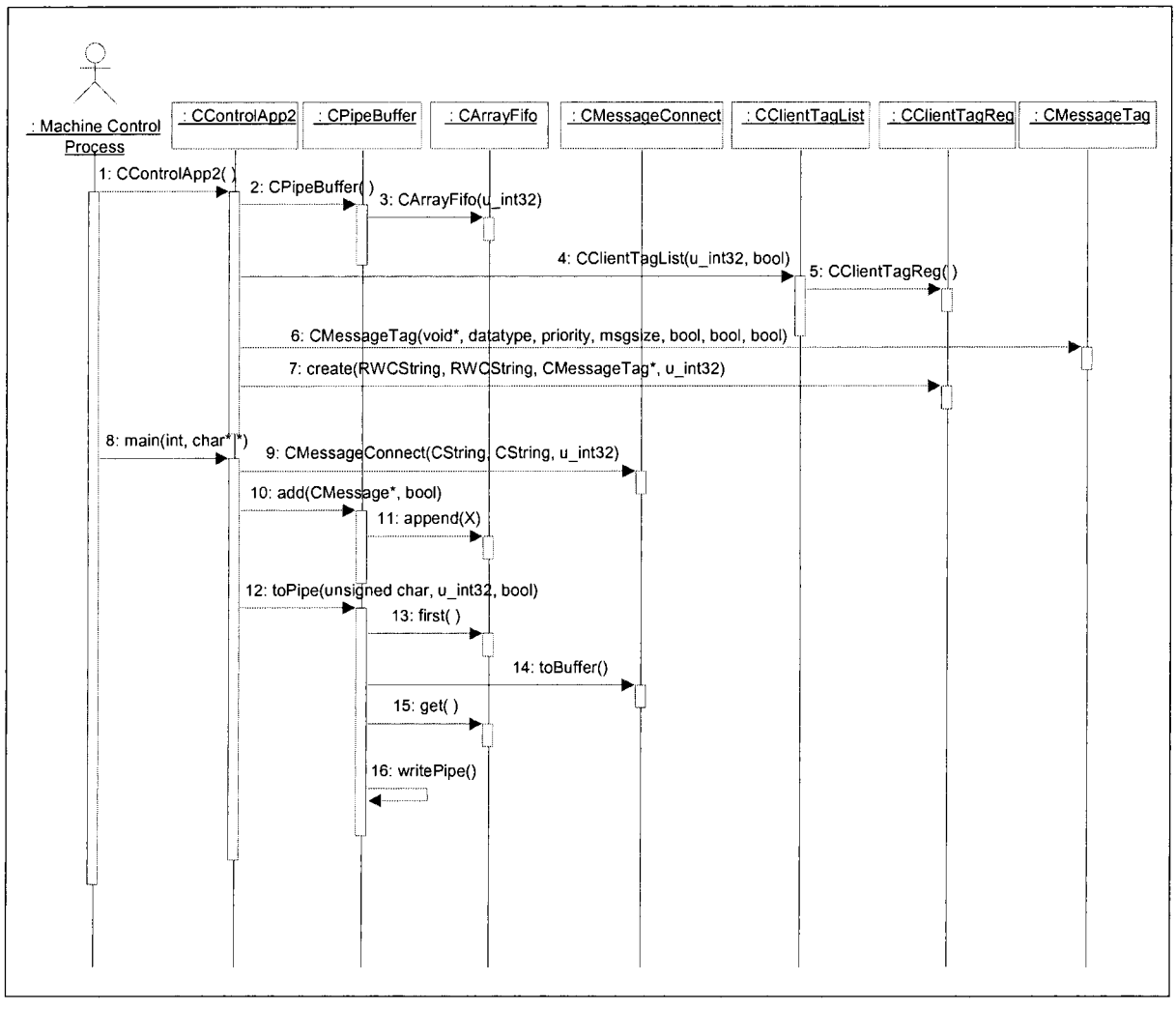


**Figure 8-10: Sequence Diagram of a Machine Control Process starting up**

1: When a Machine Control Process starts it creates a global object: the Control Application.

2: When the Control Application object is created, the buffers that are used for communicating through the pipes are also created. Three buffer objects are created:

- One for handling the messages, which are to be sent through the Priority Message Pipes.
- One for handling the messages, which are to be sent through the Connect Pipe.
- One for receiving messages from the Message Pipe to the Machine Control Process.

3: Together with the creation of the buffers, also the FIFO queues that are used by the buffer objects are created. One queue per pipe is created. Thus the amount of FIFO queues in the buffer object that handles the Priority Message Pipes is equal to the number of Priority Message Pipes.

4: A registration object is created. This objects holds a list of objects that hold information about the tags in the process.

5: A tag registration object is created for each tag in the Machine Control Process. These objects will hold the name, path, tag id, and a tag message for each tag in the process.

6: The messages that will be used for sending data values of a tag are created. One message is created for each tag.

7: A pointer to the message and the name, path and tag id are written to the tag registration objects. This is done for each tag in the system.

8: After the global Control Application object is created, the main function of the Control Application object is called. This function will run during the live time of the Machine Control Process.

9: The first thing the main function does is creating a Connect Message. This message is used to notify the OPC Server Process of the start up of the Machine Control Process.

10: The Connect Message is placed in the buffer that is related to the Connect Pipe.

11: The Connect Message is appended to the FIFO queue.

12: The messages in the buffer (this is only the Connect Message that is just created) are sent to the Connect Pipe.

13: The first thing the procedure that sends the messages to the pipe does is to get the first message from the FIFO queue.

14: The message that is just acquired from the queue copies itself to the buffer that is used for sending messages to the pipe.

15: If the message is successfully copied to the buffer it is removed from the top of the FIFO queue.

16: Because there was only one message in the queue, the buffer can now be copied to the pipe.

**Text 8-3: Description of the sequence diagram in Figure 8-10**

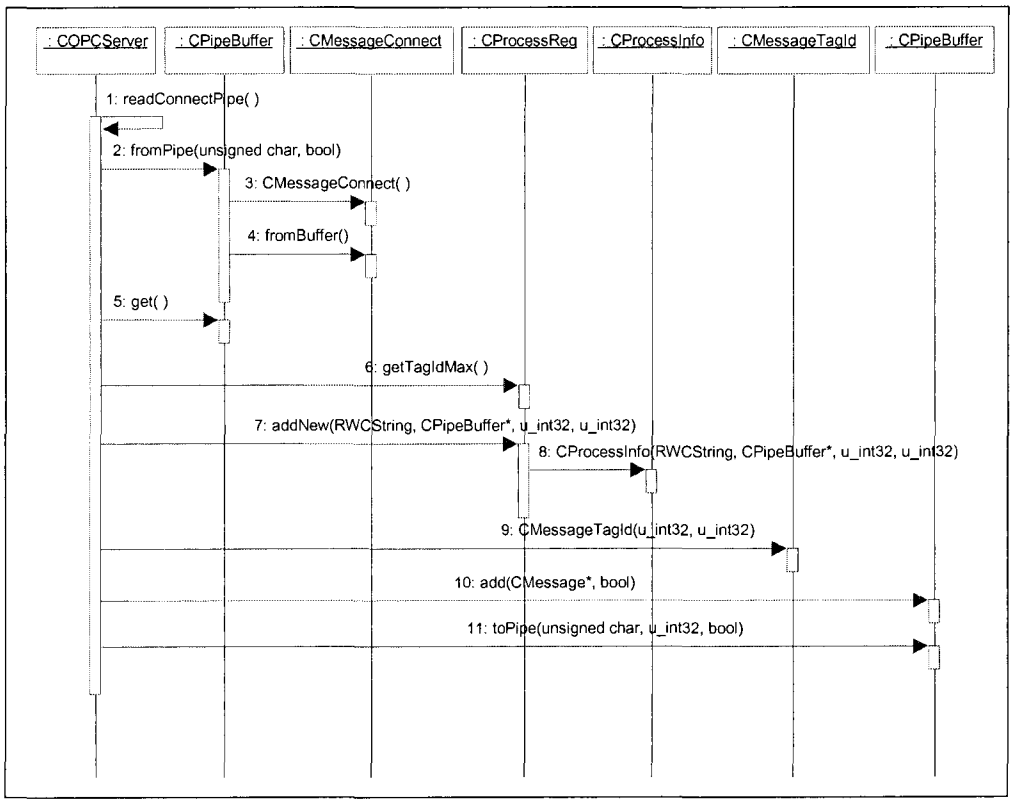


**Figure 8-11: Sequence Diagram of the OPC Server Process receiving a Connect Message**

1: When the a message is written to the Connect Pipe, the Operating System sends a signal to the OPC Server Process. The OPC Server receives this signal and reads information from the Connect Pipe.

2: The message is read from the pipe.

3: The OPC Server Process gets the first message from the queue and the message is removed from the queue.

4: The type of the received message is determined and if it is a Connect Message, the OPC Server Process gets the maximum assigned tag number so far from the registration database.

5: The acquired maximum assigned tag number together with the information from the received Connect Message is used to assign new tag numbers to the tags in the Machine Control Process that sent the Connect Message.A new entry in the registration database is made.

6: The minimum and maximum tag numbers that are assigned to the Machine Control Process are stored together with the name and pipe name that where in the received Connect Message.

7: A new Tag Id Message is created. This message will be used to inform the Machine Control Process that sent the Connect Message about the assigned tag numbers.

8: The message is added to the pipe buffer that is associated with the Message Pipe of the Machine Control Process.

9: The Tag Id Message is sent to the pipe.

(the internal operations of the 'add' and 'toPipe' functions of the CPipeBuffer class are not shown in Figure 8-13, these operations are the same as in Figure 8-10.)

**Text 8-4: Description of the sequence diagram in Figure 8-11**

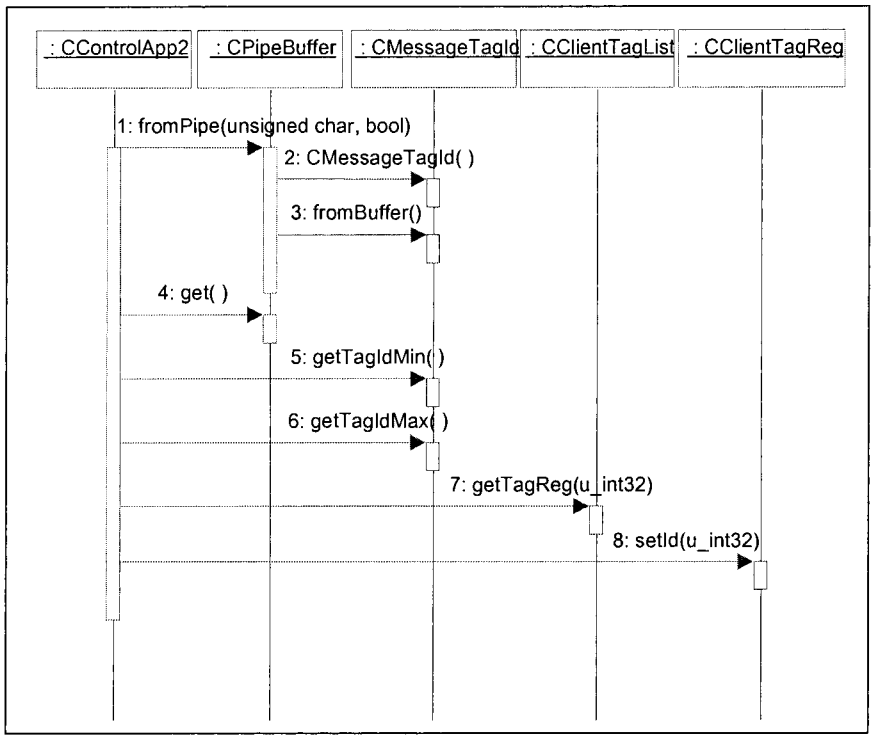


**Figure 8-12: Sequence Diagram of the a Machine Control Process receiving a tag id message**

1: The Machine Control Process receives a signal that data is written to its Message Pipe and reads the message from the pipe into a buffer.

2: A tag id message is created.

3: The message reads its data values from the buffer and is placed in the internal queues of the buffer object.

4: The message it removed from the queue in the buffer object.

5: The minimum id that is available for assigning to the tags is acquired from the message.

6: The maximum id that is available for assigning to the tags is acquired from the message.

(The following steps are repeated (TagIdMax-TagIdMin+1) times. Thus for every tag the Machine Control Process provides)

7: A tag registration object is acquired from the registration list.

8: An id is assigned to the tag, whose registration information is stored in the record that was acquired.

**Text 8-5: Description of the sequence diagram in Figure 8-12**

### 8.3.2.2 Messages between the OPC Server and the OS-9 OPC Server Process when connecting

The OPC Server does not send any messages to the OPC Server Process when connecting. The OPC Server Process detects a connection from the OPC Server via the network and starts sending messages as a response to the connection. When a connecting between the OPC Server and the OPC Server Process is established, the OPC Server Process sends a registration message to all Machine Control Processes. The Machine Control Processes respond to this message by sending a tag information message for every available tag in the Machine Control Process. These messages hold information about each available tag. This information contains the name of the process, the path to the tag in the namespace of the OPC Server, the name of the tag in the namespace, the data type of the tag, and the assigned tag id.

The OPC Server Process forwards all tag information messages to the OPC Server. It does not interpret the messages, but copies them from the Priority Messages Pipes to the network. The OPC Server receives the messages and interprets them. The OPC Server finds the according tag in the namespace and assigns the tag id from the message to the tag in the namespace. The OPC Server also builds a sorted array of pointers to the tags in the namespace. The array is sorted by the assigned tag id's. This sorted array is used to be able to find a specific tag in a very effective way. Each message that arrives in the OPC Server has a tag id in it. Because the array is sorted by the tag id's of the tags, the destination tag can be found in the array be using a binary search, which is very effective. The OPC Server does not send a response to the tag information messages.

In Table 8-3 you can see an overview of the messages that are being sent between the OPC Server Process and the OPC Server when connecting. Figure 8-13 trough Figure 8-16show the sequence of operations that are performed by the various parts of the OPC Server and Text 8-6 through Text 8-9 give a brief explanation of the sequence diagrams is these figures.

**Table 8-3: Messages from the OS-9 OPC Server Process to the OPC Server when connecting**

| Message type | Description | Data in message | Purpose |
|---|---|---|---|
| CMessageTagInfo | Information about a tag | - The name, path and process name of the tag<br>- The assigned tag ID<br>- The data type | Inform the OPC Server about the ID that is assigned to the tag and to check the data type. This message is forwarded to the OPC Server |

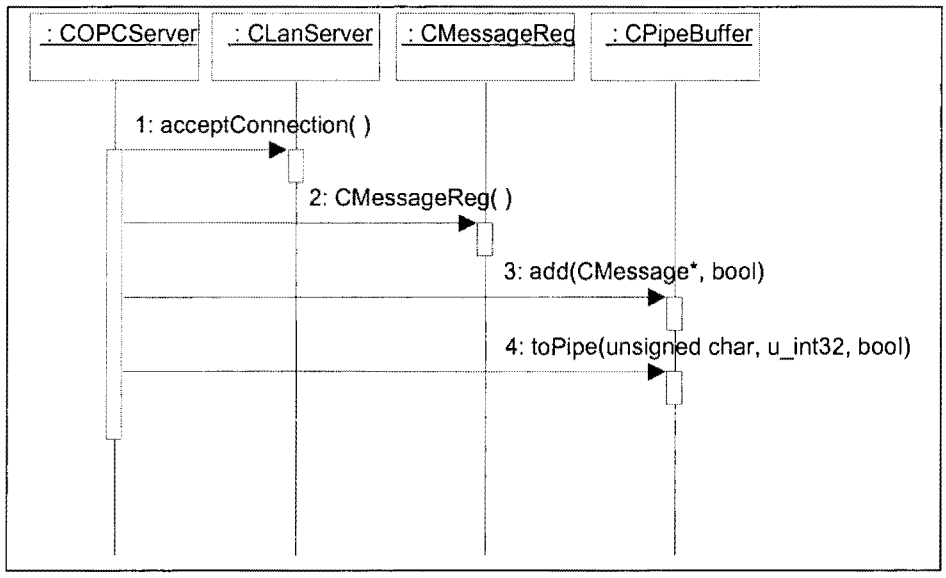


**Figure 8-13: Sequence Diagram of the OPC Server Process when the OPC Server makes a Connection**

1: The OPC Server Process receives a signal when a connection is being made to the network server. The OPC Server Process accepts the connection. Now, a connection over the network between the OPC Server Process and the OPC Server is established.

2: The OPC Server Process creates a registration message. This message is used to order the Machine Control Processes that they should send a tag information message for each available tag.

3: The created registration message is added to the buffer that is associated with a Message Pipe of a Machine Control Process. The message is copied to each buffer that is associated with a Message Pipe of a Machine Control Process. Thus each Machine Control Process will receive a registration message.

4: The registration messages are copied to the pipes. The write operation to the Message Pipes is performed for each registered Machine Control Process.

(the internal operations of the 'add' and 'toPipe' function of the CPipeBuffer class are not shown in Figure 8-13, these operations are the same as in Figure 8-10.)

**Text 8-6: Description of the sequence diagram in Figure 8-13**

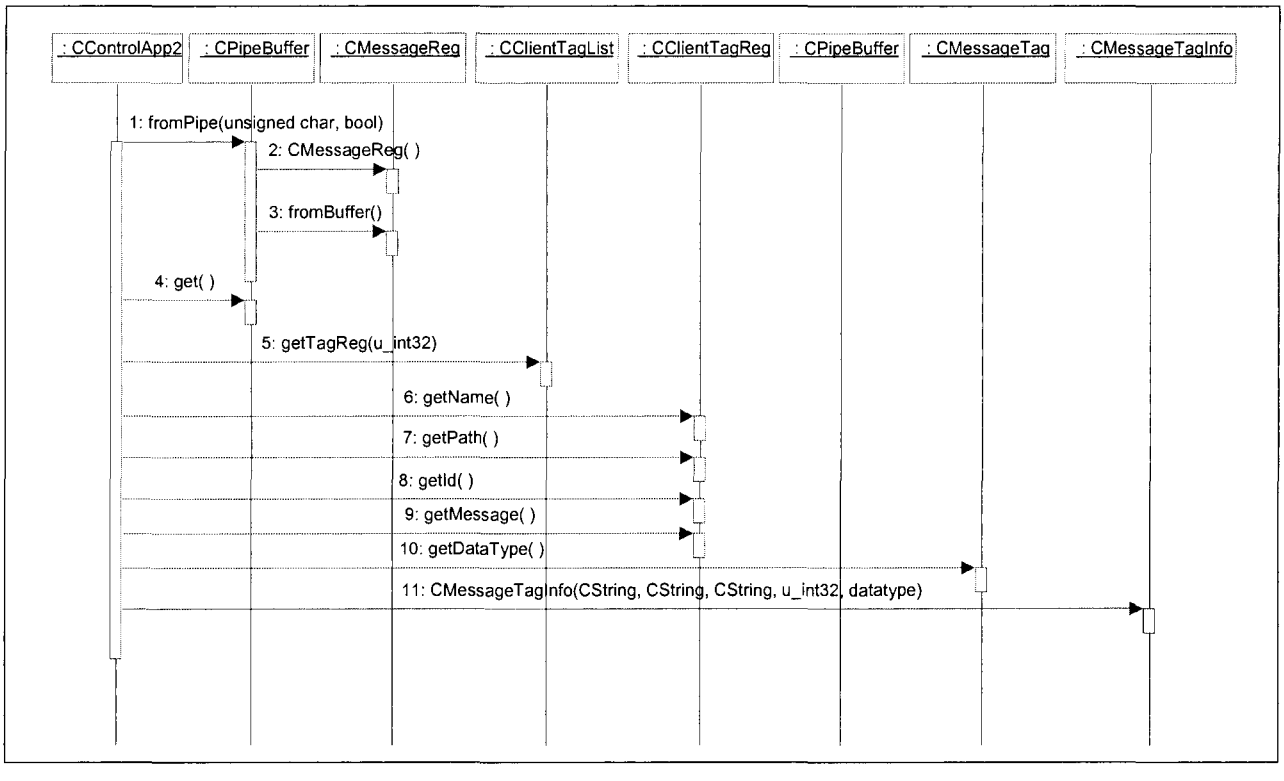


**Figure 8-14: Sequence Diagram of a Machine Control Process when sending registration messages**

1: Information is read from the pipe (not all internal operations are shown here).

2: A registration message is created.

3: The registration message is filled with data from the buffer that was filled from the pipe.

4: The Machine Control Process gets the registration message from the pipe buffer object.

(The steps 5 through 12 are repeated for each tag provided by the Machine Control Process)

5: Tag registration information is acquired from the registration list.

6: The name of the tag is acquired from the registration object.

7: The path of the tag is acquired from the registration object.

8: The tag id is acquired from the registration object.

9: The tag message that will be used for sending data values of the tag is acquired from the registration object.

10: The data type is acquired from the tag message that was acquired.

11: The information that was acquired on the lines 6 through 10 is used to create a tag information message.

(The following steps are not shown in the diagram for readability. These steps are already explained in previous diagrams)

12: The tag information message is added to the pipe buffer.

13: When all messages are added to the pipe buffer, the messages are send to the Priority Pipes.

**Text 8-7: Description of the sequence diagram in Figure 8-14**

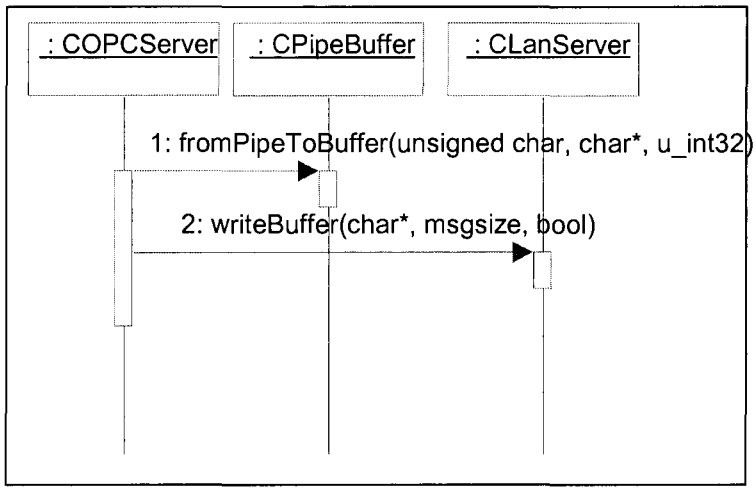


**Figure 8-15: Sequence Diagram of the OPC Server Process when forwarding messages from a pipe to the network**

| | |
|---|---|
| 1: | The OPC Server Process reads the messages into a buffer. It does not interpret the messages it reads. |
| 2: | The messages are written to the network. The OPC Server will receive and interpret the messages. |

**Text 8-8: Description of the sequence diagram in Figure 8-15**

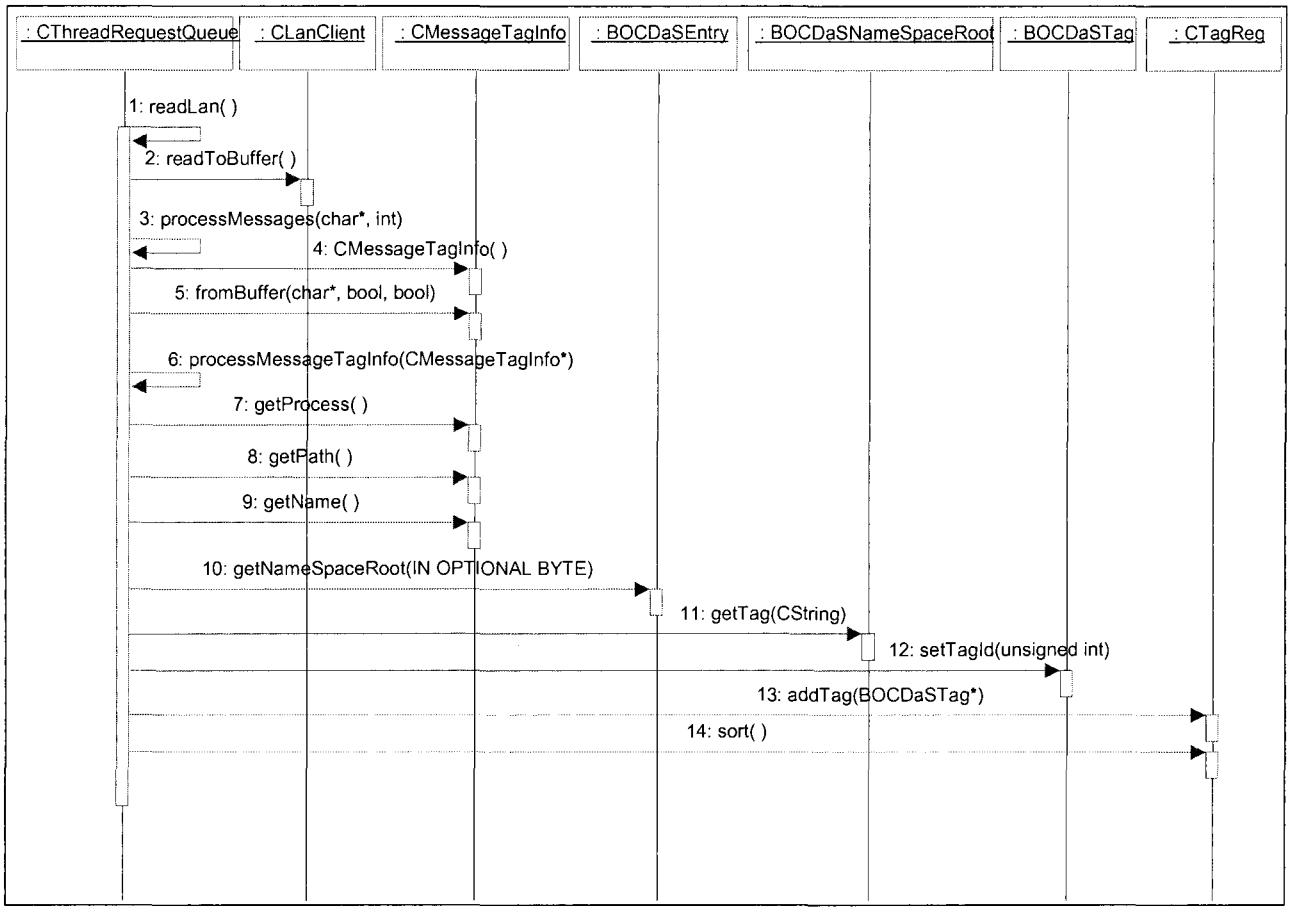


**Figure 8-16: Sequence Diagram of the OPC Server receiving a tag information message**

1: The thread that manages all messages and communications for the connection with the machine that sent the tag information message receives a message that there is data available from the network.

2: The data is read from the network into a buffer.

3: The thread starts processing the message in the buffer.

(The steps 4 through 13 are repeated for all messages that are read into the buffer)

4: The type of the message is determined and a tag information message object is created.

5: The created tag information message reads gets its data values from the buffer.

6: The thread starts the handling of the tag information message.

7: The tag's process name is acquired from the message.

8: The tag's path name is acquired from the message.

9: The tag's name is acquired from the message.

10: The root of the server's namespace is acquired from the entry point of the OPC Server.

11: The machine name, which is known to the thread, the process name, the path name and the tag's name are combined to uniquely identify the tag in the namespace. The tag is searched for in the namespace.

12: The tag id that is assigned to the tag by tag Machine Control Process is assigned to the tag in the namespace.

13: The tag is added to the tag registration object. This object holds all tags that are registered by the control system.

14: The tags in the tag registration object are sorted by their tag id. This enables the OPC Server to quickly find a tag when a message arrives for a specific tag.

**Text 8-9: Description of the sequence diagram in Figure 8-16**

#### 8.3.2.3 Messages between the Machine Control Processes and the OS-9 OPC Control Process when connected

When the connection phase is completed and all tag registrations are negotiated, the system is in its connected state. The Machine Control Process and the OPC Server Process do not send messages to each other that are that have their source and destination in one of the processes. The OPC Server Process forwards all messages it receives. If the OPC Server Process receives a message a Priority Message Pipe, the OPC Server Process reads the data into a buffer and forwards this buffer to the OPC Server via the network. The same sequence as for the forwarding of tag information message, as shown in Figure 8-15, is performed. The messages that are sent to the Priority Message Pipes by the Machine Control Processes are tag messages that are used to send a new data value of a tag to the OPC Server and the answer messages that are sent when a read or write request is sent to a Machine Control Process.

The OPC Server process can receive read and write request messages from the OPC Server. When a request message arrives the OPC Server Process interprets the message and acquires the tag id of the message. This tag id is used to find the Machine Control Process, the message should be forwarded to, in the process registration object. The OPC Server Process then places the message in the buffer that is associated with the Message Pipe of the destination Machine Control Process. When the messages that were read from the pipe are processed, the messages are written to the Message Pipes.

When a Machine Control Process receives a read or write request message it finds the destination tag in the tag registration object. If the request is a read request, a read answer message is created. The current value of the tag is copied to the answer message and the message is sent to the OPC Server Process via the Priority Message Pipes. The OPC Server Process forwards the message to the OPC Server. If the request is a write request, the Machine Control Process writes the data from the messages to the tag and creates a write answer message. The value that is written to the tag is copied into the write answer message. The write answer message is sent to the Priority Message Pipes. The OPC Server Process reads the message from the pipe and forwards it to the OPC Server.

There is one message that is send from the OPC Server process to the Machine Control Process after the connection phase. This is a disconnect message. It is sent to the Machine Control Processes when a disconnection from the OPC Server to the OPC Server Process is detected. The Machine Control Process stop sending tag messages to the OPC Server and wait for a new registration message to arrive. If a registration message arrives, the Machine Control Process sends all tag information to the OPC Server again. The disconnection of the OPC Server could be caused, because the OPC Server was stopped. When this is the case all registration information in the OPC Server is lost and needs to be received again.

Table 8-4 and Table 8-5 show all messages that are sent between the Machine Control Processes and the OPC Server Process when they are connected. Figure 8-17 through Figure 8-20 show the sequences that are executed when messages are sent between the Machine Control Processes and the OPC Server Process.

Table 8-4: Messages from Machine Control Processes to the OS-9 OPC Server Process when connected

| Message type | Description | Data in message | Purpose |
|---|---|---|---|
| CMessageTag | New value of a tag's data | Data of the same type as the data type of the tag the message is sent for | Send a changed value of a tag to the OPC Server |
| CMessageAnswerRead | The answer to a read request | Data of the same type as the data type of the tag the message is sent for | Used to send the value of the tag that received a read request back to the OPC Server |
| CMessageAnswerWrite | The answer to a write request | Data of the same type as the data type of the tag the message is sent for | The value written to a tag is by a write request is returned to the OPC Server to inform it that the request is completed |

Table 8-5: Messages from the OS-9 OPC Server Process to Machine Control Processes when connected

| Message type | Description | Data in message | Purpose |
|---|---|---|---|
| CMessageRequestRead | A read request for a specific tag | none | Used to perform a read request for a specific tag in the control system |
| CMessageRequestWrite | A write request for a specific tag | Data of the same type as the data type of the tag the message is sent for | Write a value to a tag in the control system |
| CMessageDisconnect | The OPC Server disconnected | none | Used to inform the Machine Control Process that the OPC Server disconnected from the OPC Server Process |

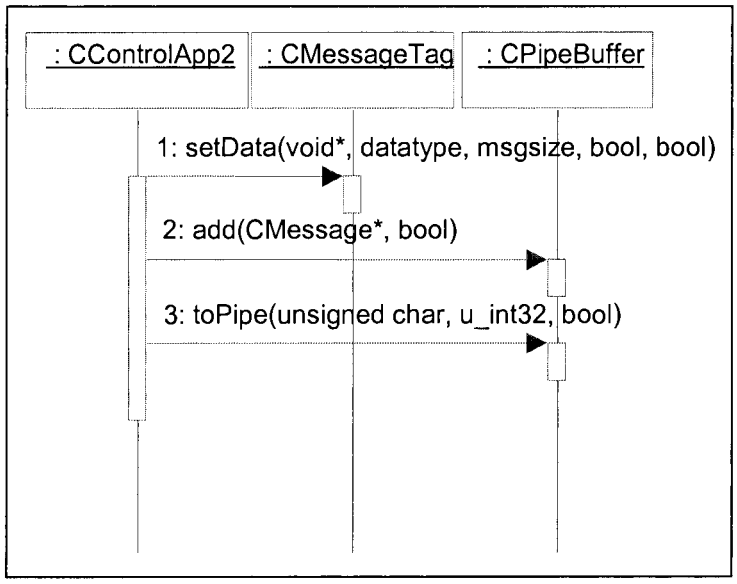


**Figure 8-17: Sequence Diagram of a Machine Control Process sending a Tag Message**

(The steps 1 and 2 are repeated for all tags whose value has changed)

1: The new value of the tag's data is written to the associated tag message.

2: The message holding the new value is added to the buffer that is used for communicating through the Priority Message Pipes.

3: All messages in the buffer are written to the pipe.

**Text 8-10: Description of the sequence diagram in Figure 8-17**

The message are read from the pipe by the OPC Server Process and forwarded to the OPC Server via the network. The sequence executed by the OPC Server Process to perform this task is the same as in Figure 8-15.

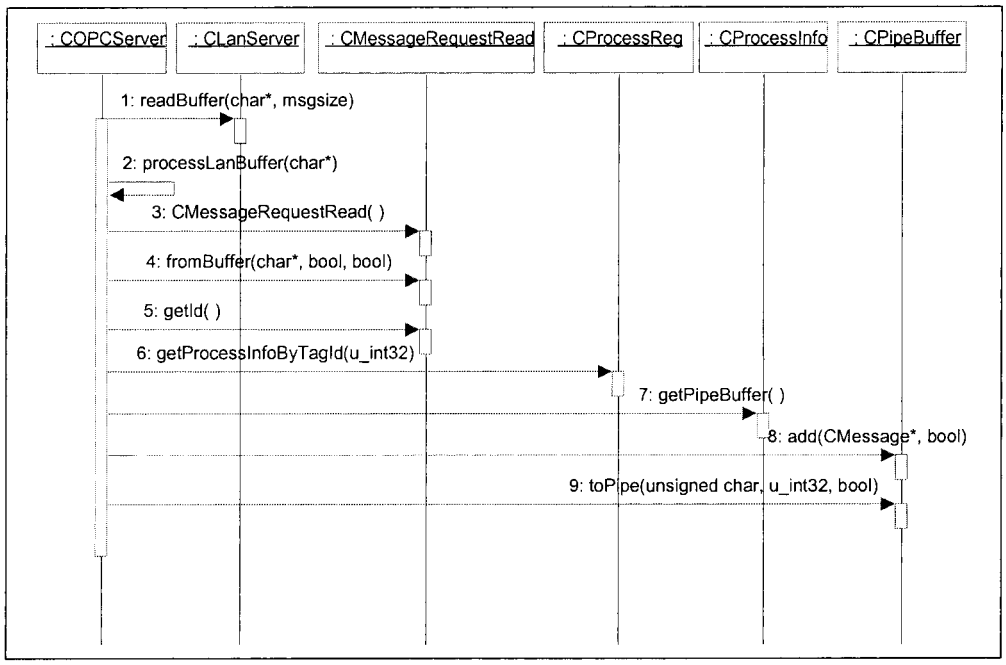


**Figure 8-18: Sequence Diagram of the OPC Server Process receiving a read request message**

| | |
|---|---|
| 1: | The OPC Server Process reads the arrived information from the network into a buffer. |
| 2: | The OPC Server Process starts processing the information in the buffer. |
| (The steps 3 through 8 are repeated for all messages that are read into the buffer) | |
| 3: | The type of the message is determined and a read request message is created. |
| 4: | The message reads itself the buffer. |
| 5: | The OPC Server Process gets the tag id from the message. |
| 6: | The tag id is used to get the process information record from the process registration list. |
| 7: | The buffer that is associated with the Message Pipe of the process that holds the destination tag is acquired from the process information record. |
| 8: | The message is added to the buffer. |
| 9: | When all messages are processed they are sent to the Message Pipes. |

**Text 8-11: Description of the sequence diagram in Figure 8-18**

When the OPC Server process receives a write request message the same sequence as shown in Figure 8-18 is executed. The only difference is that instead of a read request message a write request message is created.

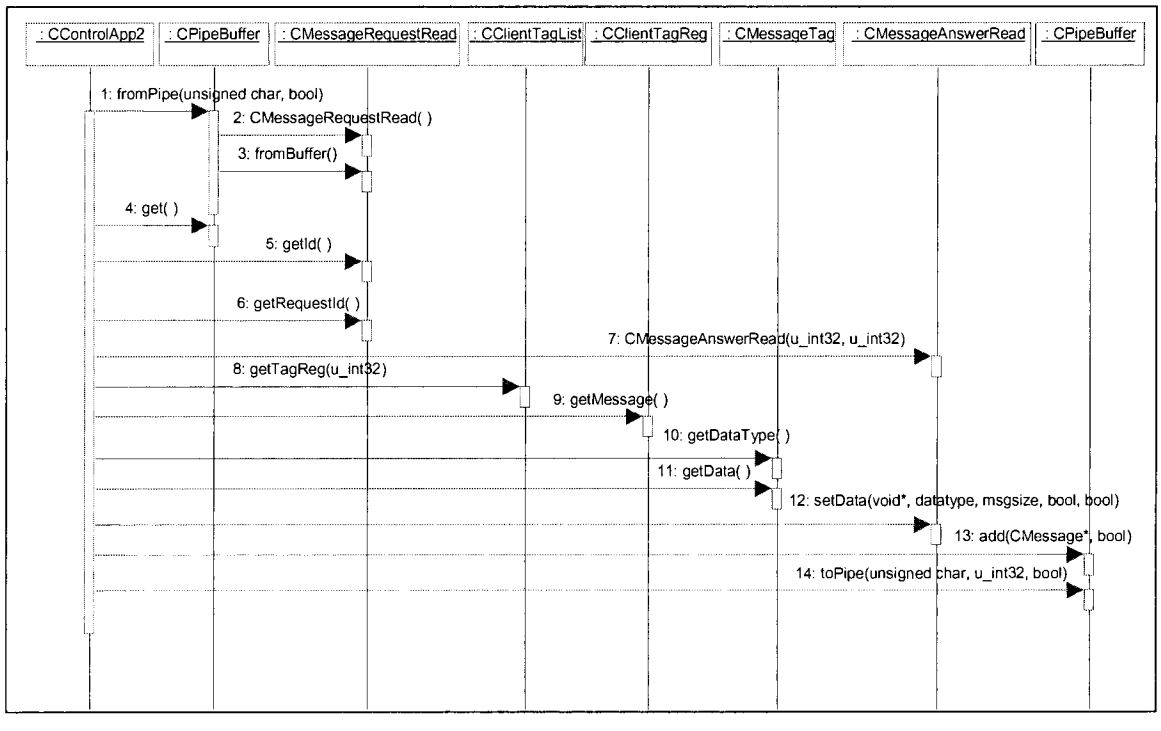


**Figure 8-19: Sequence Diagram of a Machine Control Process receiving a read request message**

1: The messages in the pipe are read into a buffer.

(The steps 2 through 13 are repeated for all read request messages in the pipe)

2: The message type is determined and a read request message is created.

3: The message reads itself from the buffer.

4: The Machine Control Process reads the first message from the queue in the buffer.

5: The tag id is acquired from the message.

6: The request id is acquired from the message.

7: The tag id and the request id are used to create a read answer message.

8: The tag registration record that is associated with the tag id is acquired from the tag registration object.

9: The tag message that holds the latest value of the tag is acquired from the tag registration record.

10: The data type is acquired from the tag message.

11: The data value is acquired from the tag message.

12: The data type and data value are written to the read answer message.

(The steps 9 through 12 can also be replaced by a call to a so-called call-back function that is responsible for writing the data to read answer message.)

13: The read answer message is added to the buffer associated with the Priority Message Pipes.

14: The messages in the buffer are written to the pipes.

**Text 8-12: Description of the sequence diagram in Figure 8-19**

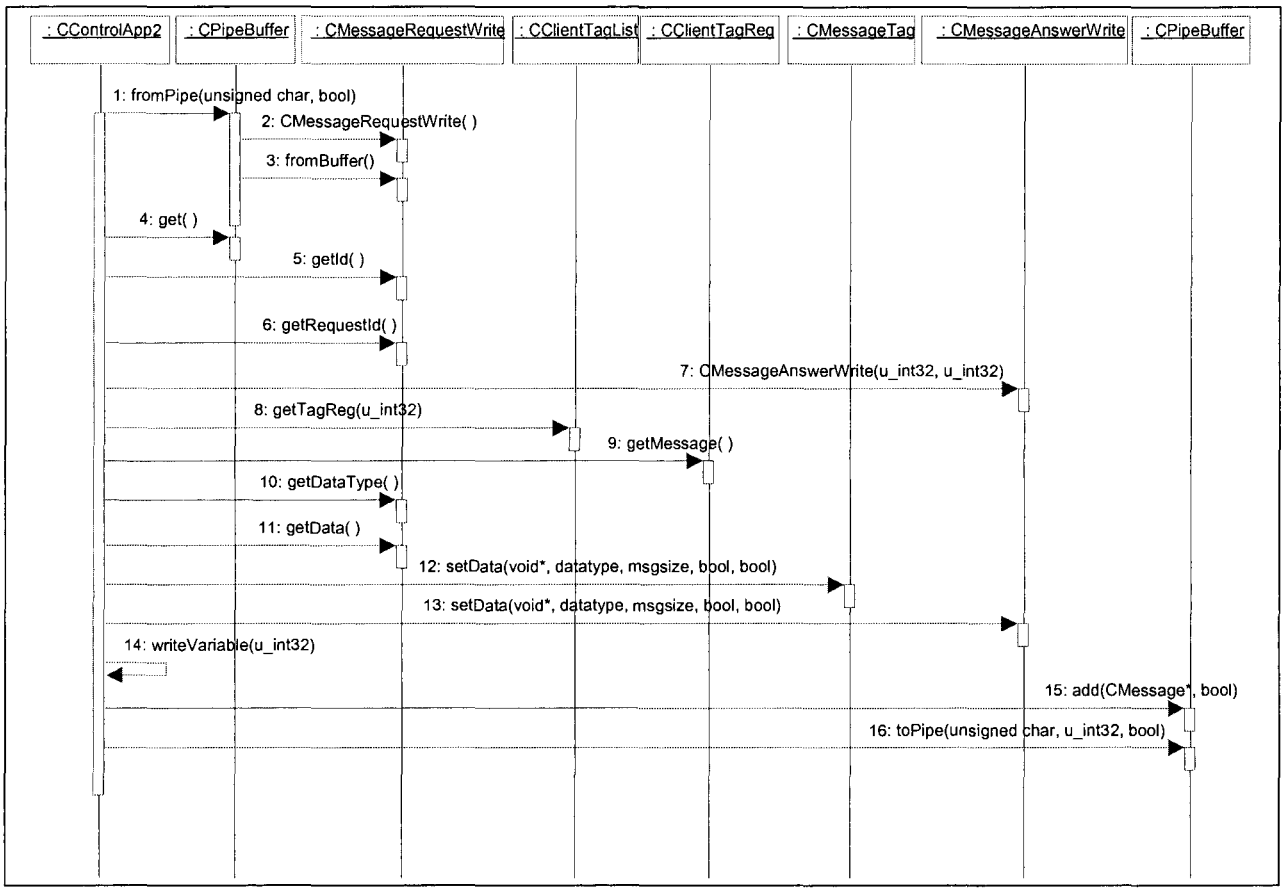


**Figure 8-20: Sequence Diagram of a Machine Control Process receiving a write request message**

1: The messages in the pipe are read into a buffer.

(The steps 2 through 15 are repeated for all write request messages in the pipe)

2: The message type is determined and a write request message is created.

3: The message reads itself from the buffer.

4: The Machine Control Process reads the first message from the queue in the buffer.

5: The tag id is acquired from the message.

6: The request id is acquired from the message.

7: The tag id and the request id are used to create a write answer message.

8: The tag registration record that is associated with the tag id is acquired from the tag registration object.

9: The tag message that holds the latest value of the tag is acquired from the tag registration record.

10: The data type is acquired from the write request message.

11: The data value is acquired from the write request message.

12: The data type and data value are written to the tag message.

13: The data type and data value are written to the write answer message.

14: The data is written to the real variable in the control system. This is done by a global procedure that knows what to do for every variable in the Machine Control Process when its value is written.

(The steps 9 through 11 and 14 can also be replaced by a call to a so-called call-back function that is responsible for writing the data to real variable and take specific action within the process.)

15: The read answer message is added to the buffer associated with the Priority Message Pipes.

16: The messages in the buffer are written to the pipes.

**Text 8-13: Description of the sequence diagram in Figure 8-20**

#### 8.3.2.4 Messages between the OPC Server and the OS-9 OPC Server Process when connected

When the OPC Server has connected to the control system and the OPC Server Process has forwarded all tag information messages from all Machine Control Processes to the OPC Server, the connection phase is over and the OPC Server and the OPC Server Process are said to be connected.

When the OPC Server and the OPC Server Process are connected, the OPC Server Process only forwards message from the Machine Control Process to the OPC Server and visa versa. The Machine Control Processes send tag messages that hold the current value of a tag. Each time the value of the data of a tag changes the new value is sent from the Machine Control Process to the OPC Server Process, which forwards it to the OPC Server.

Besides tag messages, there is one other stream of data in the system when connected. The OPC Server generates read and write requests for the tags in the Machine Control Process. The clients that are connected to the OPC Server initiate the read and write requests. The OPC Server sends the request messages to the OPC Server Process, which interprets the messages and forwards them to the right Machine Control Process. The Machine Control Process receives the request message and reads or writes the data to the tag. Subsequently the Machine Control Process creates an answer message and sends this back to the OPC Server Process. The OPC Server process forwards the message to the OPC Server. The OPC Server receives the answer message, writes the received data to the specific tag in the server's namespace and completes the request.

Table 8-6 and Table 8-7 show the messages that are sent between the OPC Server Process and the OPC Server when they are connected. Figure 8-21 through Figure 8-24 show the sequences that are executed when these messages are created or received.

**Table 8-6: Messages from the OPC Server to the OS-9 OPC Server Process when connected**

| Message type | Description | Data in message | Purpose |
|---|---|---|---|
| CMessageRequestRead | A read request for a specific tag | none | Used to perform a read request for a specific tag in the control system |
| CMessageRequestWrite | A write request for a specific tag | Data of the same type as the data type of the tag the message is sent for | Write a value to a tag in the control system |

**Table 8-7: Message from the OS-9 OPC Server Process to the OPC Server when connected**

| Message type | Description | Data in message | Purpose |
|---|---|---|---|
| CMessageTag | New value of a tag's data | Data of the same type as the data type of the tag the message is sent for | Send a changed value of a tag to the OPC Server |
| CMessageAnswerRead | The answer to a read request | Data of the same type as the data type of the tag the message is sent for | Used to send the value of the tag that received a read request back to the OPC Server |
| CMessageAnswerWrite | The answer to a write request | Data of the same type as the data type of the tag the message is sent for | The value written to a tag is by a write request is returned to the OPC Server to inform it that the request is completed |

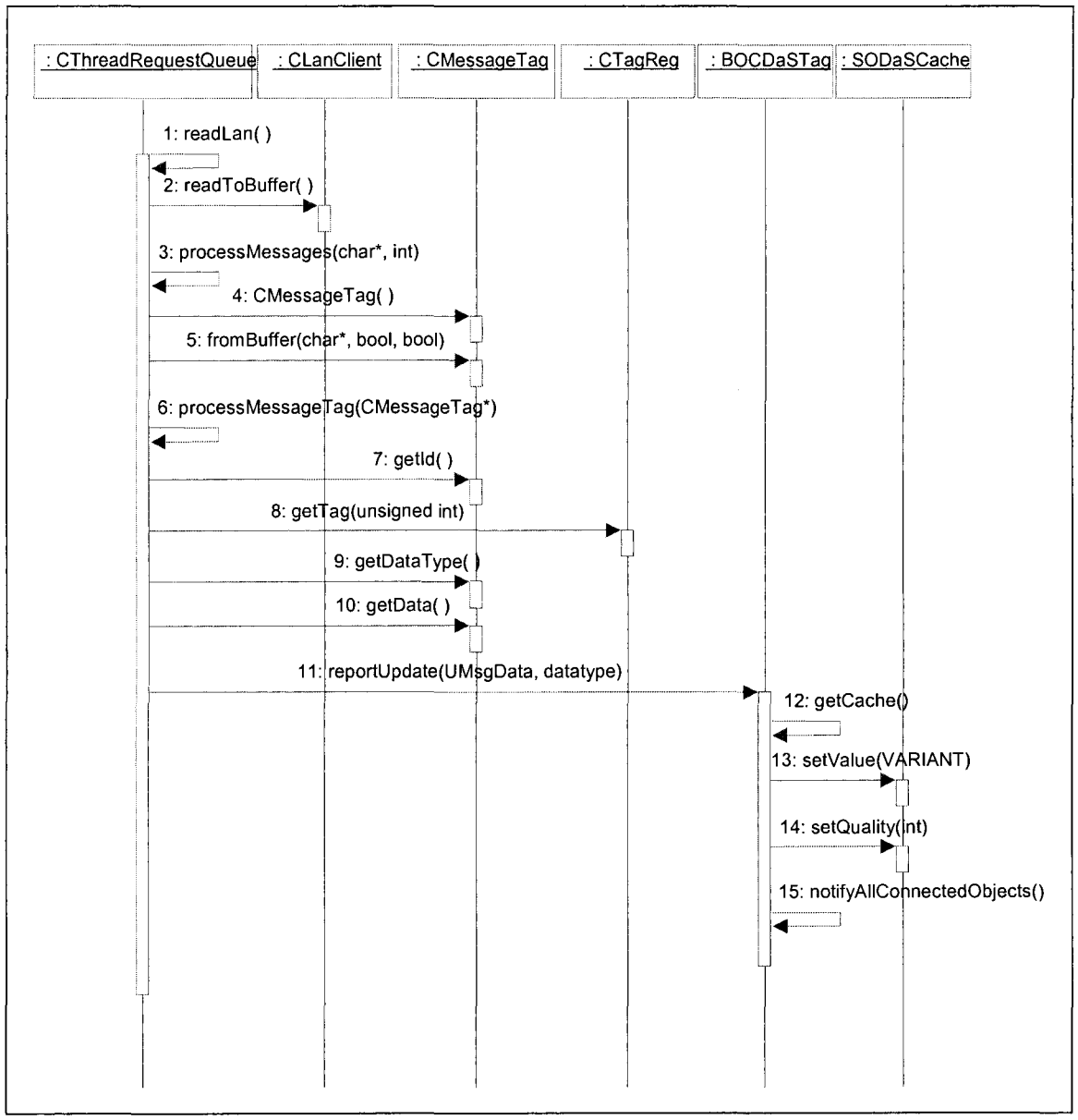


**Figure 8-21: Sequence Diagram of the OPC Server receiving a tag message**

1: The thread that manages all messages and communications with the machine that sent the tag message receives a message that there is data available from the network.

2: The data is read from the network into a buffer.

3: The thread starts processing the messages in the buffer.

(The steps 4 through 15 are repeated for all messages that are read into the buffer)

4: The type of the message is determined and a tag message object is created.

5: The tag message reads itself from the buffer.

6: The thread starts the handling of the tag message.

7: The tag id is acquired from the message.

8: The tag, that is located in the OPC Server's namespace, is acquired from the tag registration by use of the tag's id.

9: The data type is acquired from the tag message.

10: The data is acquired from the tag message.

11: The new data value is reported to the tag in the namespace.

12: The cache object that holds the last value of the tag's data is acquired.

13: The new data is written to the cache.

14: The quality of the data is set to 'good'.

15: All objects that are connected to the tag are notified of the new data value. The connected objects are the items that the OPC Server created for clients that want to connect to a certain tag.

**Text 8-14: Description of the sequence diagram in Figure 8-21**

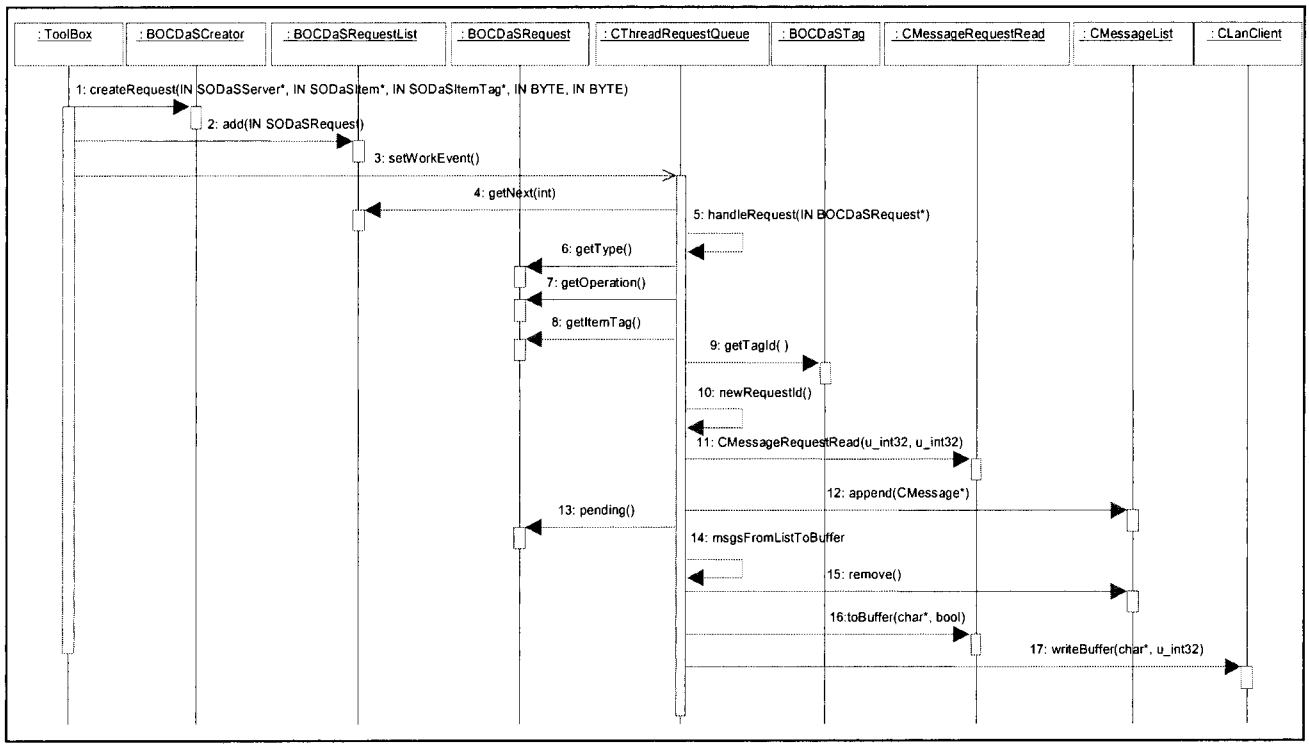


**Figure 8-22: Sequence Diagram of the OPC Server sending read request messages**

1: The internals of the toolbox call the request creator function of a creator object that is registered in the toolbox. This creator object is responsible for creating all objects that are used by the internals of the toolbox. The request is created as a result of a read request by one of the connected OPC Clients.

2: The internals of the toolbox place the created request in the request list that is associated with the destination machine. That is the machine that holds the tag the request is made for.

(The steps 1 and 2 can be repeated if a client performs more than one read request at the same time)

3: When all requests are added to the list, an event is set. This event activates the thread that is responsible for the handling of all messages for the destination machine.

(The steps 4 through 13 are repeated for all messages in the queue that are not yet pending, cancelled or completed)

4: The thread gets the next request from the request list.

5: If the request is not pending, cancelled or completed the handling of the request is started.

6: The request type is acquired from the request.

7: The operation is acquired from the request. The return value is a read or write operation.

8: The tag that is associated with the request is acquired from the request.

9: The tag id is acquired from the tag.

10: A new request id is generated. This request id is unique within the thread.

11: A read request message is created using the tag id and request id.

12: The read request message is added to the queue of messages that are ready to be sent to the OPC Server Process.

13: The request is set to the pending state.

14: The messages that are placed in the list are copied into a buffer.

(The steps 15 through 16 are repeated for all messages that are placed in the list)

15: The first message is removed from the list.

16: The message that is removed from the list is copied into the buffer.

17: When all messages are copied to the buffer or the buffer is full, the buffer is written to the OPC Server Process via the network. The OPC Server Process interprets and forwards the messages as shown in the previous section.

**Text 8-15: Description of the sequence diagram in Figure 8-22**

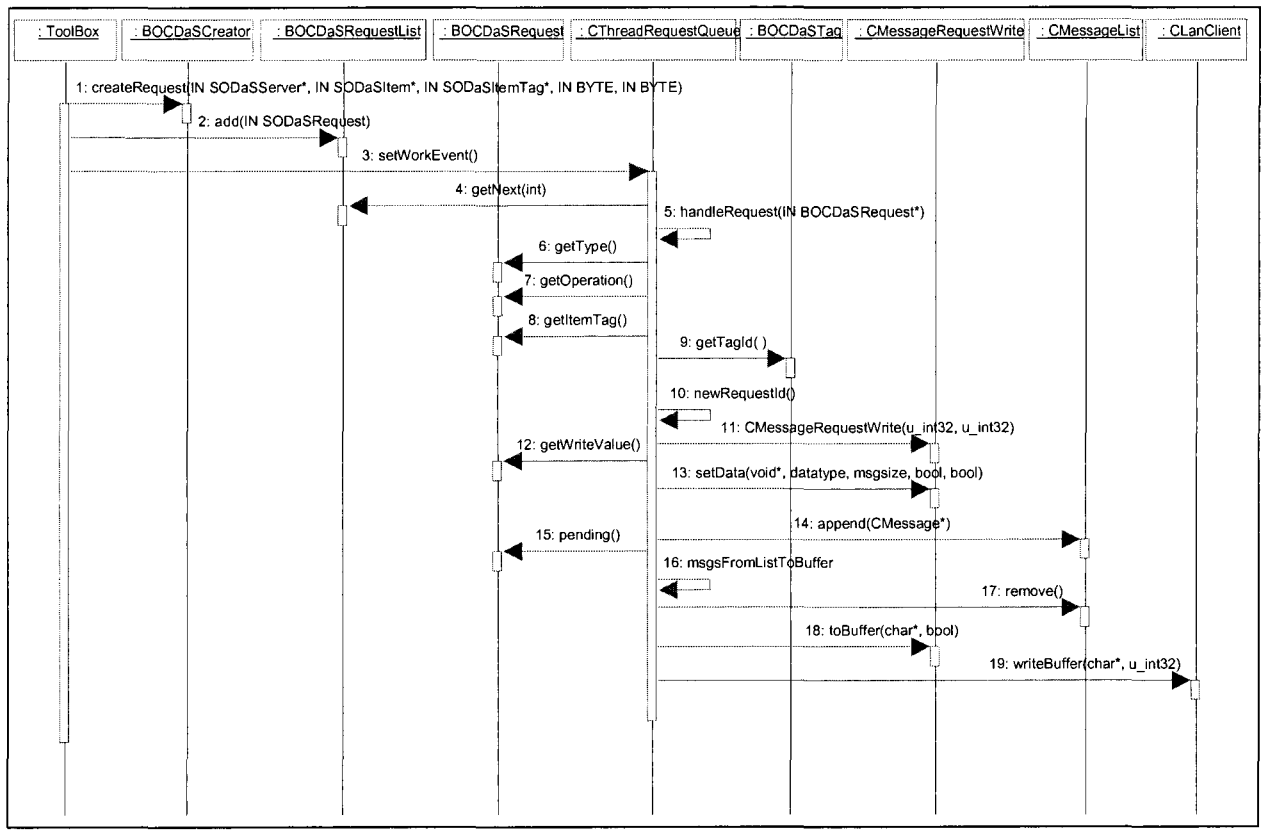


**Figure 8-23: Sequence Diagram of the OPC Server sending write request messages**

1: The internals of the toolbox call the request creator function of a creator object that is registered in the toolbox. This creator object is responsible for creating all objects that are used by the internals of the toolbox. The request is created as a result of a write request by one of the connected OPC Clients.

2: The internals of the toolbox place the created request in the request list that is associated with the destination machine. That is the machine that holds the tag the request is made for.

(The steps 1 and 2 can be repeated if a client performs more than one write request at the same time)

3: When all requests are added to the list, an event is set. This event activates the thread that is responsible for the handling of all messages for the destination machine.

(The steps 4 through 13 are repeated for all messages in the queue that are not yet pending, cancelled or completed)

4: The thread gets the next request from the request list.

5: If the request is not pending, cancelled or completed the handling of the request is started.

6: The request type is acquired from the request.

7: The operation is acquired from the request. The return value is a read or write operation.

8: The tag that is associated with the request is acquired from the request.

9: The tag id is acquired from the tag.

10: A new request id is generated. This request id is unique within the thread.

11: A write request message is created using the tag id and request id.

12: The data value an OPC Client wants to write to the tag is acquired from the request.

13: The data value is written to the write request message.

14: The write request message is added to the queue of messages that are ready to be sent to the OPC Server Process.

15: The request is set to the pending state.

16: The messages that are placed in the list are copied into a buffer.

(The steps 17 through 18 are repeated for all messages that are placed in the list)

17: The first message is removed from the list.

18: The message that is removed from the list is copied into the buffer.

19: When all messages are copied to the buffer or the buffer is full, the buffer is written to the OPC Server Process via the network. The OPC Server Process interprets and forwards the messages as shown in the previous section.

**Text 8-16: Description of the sequence diagram in Figure 8-23**

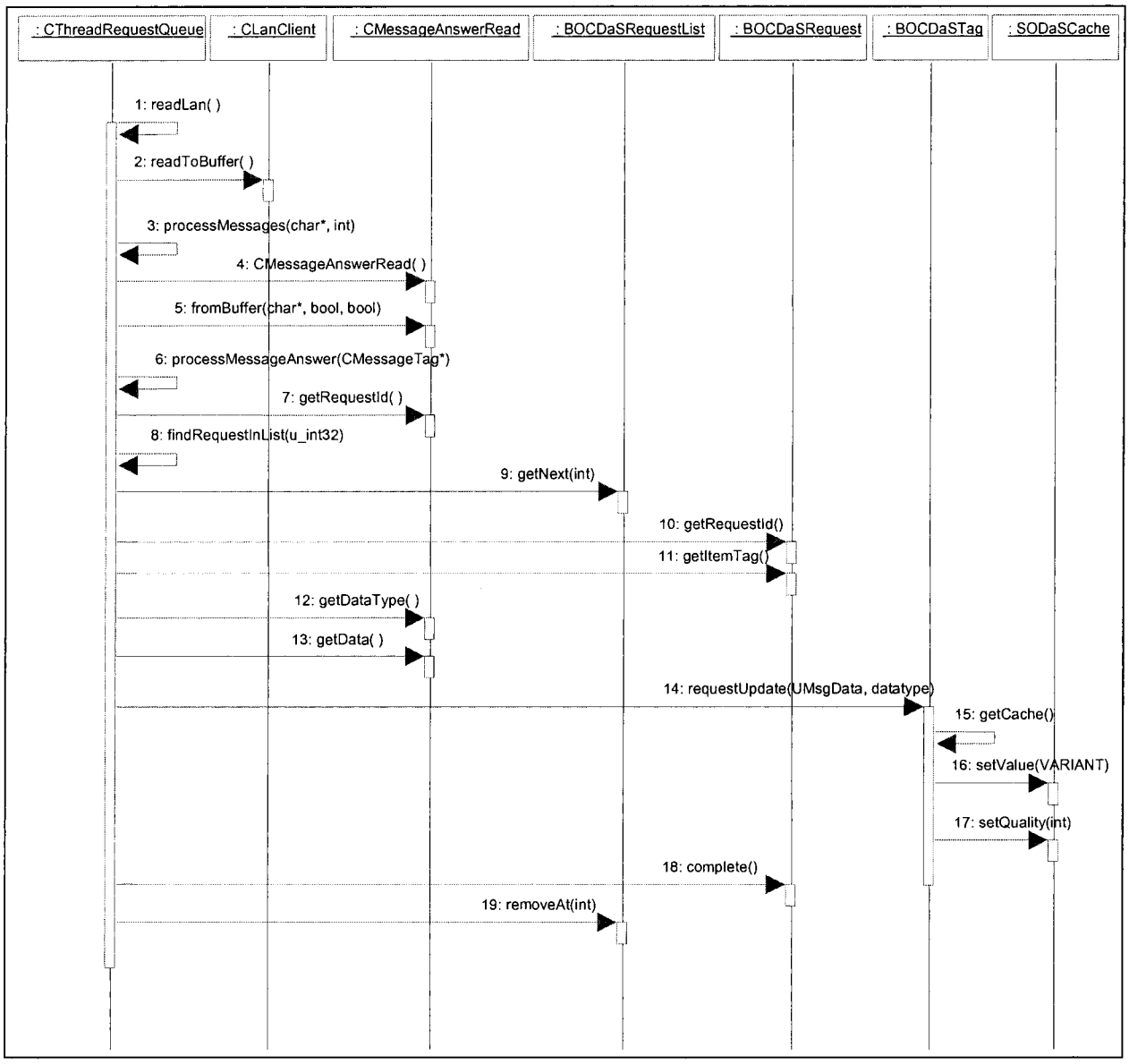


**Figure 8-24: Sequence Diagram of the OPC Server receiving a read answer message**

1: The thread that manages all messages and communications with the machine that sent the read answer message receives a message that there is data available from the network.

2: The data is read from the network into a buffer.

3: The thread starts processing the messages in the buffer.

(The steps 4 through 19 are repeated for all messages that are read into the buffer)

4: The type of the message is determined and a read answer message object is created.

5: The read answer message reads itself from the buffer.

6: The thread starts the handling of the answer message.

7: The request id is acquired from the message.

8: The thread starts looking in the request list for the request with the specific request id.

(The steps 9 and 10 are repeated until the request is found or the end of the list is reached)

9: The next request is acquired from the request list.

10: The request id is acquired from the request.

11: If the request id that is acquired from the request equals the request id from the read answer message, the destination tag is acquired from the request.

12: The data type is acquired from the read answer message.

13: The data is acquired from the read answer message.

14: The updating of the data value for the destination tag is started.

15: The cache object that holds the last data value for the tag is acquired from the tag.

16: The data is written to the cache object.

17: The quality of the data is set to 'good'.

18: The request is set to the completed state.

19: The request is removed from the request list.

**Text 8-17: Description of the sequence diagram in Figure 8-24**

When the OPC Server receives a write answer message, the same sequence as in Figure 8-24 is executed. The only difference is that instead of a read answer message a write answer message is created. A write answer message also holds the last value of the tag in the control system, just like a read answer message. The value of the data is normally the same as for the data that is written to the tag by the corresponding write request message.

### 8.3.3 Data types for messages in the system

Messages that contain data values for tags can contain various types of data. The types of data that are supported by the OPC standards are also supported by the toolkit. These data types are supported throughout the whole OPC Server for OS-9 Controlled Systems. The data types are compatible with the data types that are supported by Microsoft's VARIANT data type. Not all data type that are supported by Microsoft's VARIANT data type are supported by the OPC standards and the toolbox. The supported data types are shown in Table 8-8.

**Table 8-8: Supported data types**

| Name | Data type and size | Remarks |
|---|---|---|
| VT_EMPTY | empty | Contains no information |
| VT_I2 | short integer (2 bytes) | |
| VT_I4 | long integer (4 bytes) | |
| VT_R4 | real (4 bytes) | |
| VT_R8 | float (8 bytes) | |
| VT_DATE | OLE datetime | The datetime type is implemented as a floating-point value, measuring days from midnight, 30 December 1899. So, midnight 31 December 1899 is represented by 1.0. The fractional part of the number is the part of the day that has passed since midnight. |
| VT_BSTR | OLE string | A null terminated string |
| VT_ERROR | Error | |
| VT_BOOL | OLE bool | |
| VT_I1 | Byte | |

Within the OPC Server a VT_BSTR is a null terminated string. When a string is passed through the pipes in the control system or over the network, terminating null character is tripped and the string is prepended with a byte integer that holds the length of the string. This has the advantage that the receiver does not have to search for the terminating null character, but knows how much space must be allocated for the storage of the string before the string is copied.

The Boolean data type supported by Microsoft Visual C++ is a 2 byte integer. Because a Boolean can only take two values, true or false, a Boolean is sent as a single byte when sent through a pipe or over a network.

Besides the data types in Table 8-8 also arrays of these data types are supported, with the exception of an array of strings. When an array is sent through the pipes or over the network the data is prepended with a 2 byte integer that holds the size of the array. In the OPC Server a so-called SAFEARRAY has to be created to pass the array to the VARIANT data type that is used for setting a tag's data value. [MSDN]

All data that is sent through the pipes or over the network is sent in network byte order, which is big endian. Because the OPC Server Process and the Machine Control Processes are tested on a Motorola 68040 processor, which is also big endian, the transformation from host byte to network byte order has no effect. The network to host and host to network byte order transformation functions for the 68040 processor are empty macros. The big advantage that the byte order in the pipes is the same as the byte order on the network is that the OPC Server Process can forward messages from the pipes to the network without having to interpret the data in the messages, but just has to copy blocks of memory.

Because the OPC Server is running on an Intel Pentium Processor, which is big endian, the OPC Server has to change the byte order of the data in a message before it is sent or after it is received from the network. Each data type has its own byte order transformation functions.

### 8.3.4 Performance considerations concerning messages in the system

As can be seen in section 8.2.1.5 the achievable throughput is much larger if a few large messages are used instead of a lot of small messages. So combining messages into larger buffers before they are sent can significantly increase the achievable throughput.

Because each packet of data that is sent over the network is accompanied with headers and checksums, combining messages also has an advantage when sending messages over the network.

Because copying messages into a buffer also takes time and it is not advisable to interrupt this process, the Machine Control Process can not perform any machine control tasks while they are combining messages and copying the resulting buffer the a pipe. Because a Machine Control Process is not allowed to be busy with other tasks then machine control for a too long period of time, the maximum used buffer size may not be too large. A compromise has to be made between buffer size, and thus throughput and the maximum time the combining and sending may take.

## 8.4 Software design of the OPC server

The OPC Server is built of several types of objects that work together. The objects can be divided in four groups. The first group of objects starts the actual OPC Server and reads the configuration database that is used for configuring the OPC Server. The second group of objects is used to build the namespace of the OPC Server. The third group is used for the communication with the machines. These objects send and receive messages to and from the control system of a machine and handle requests. The fourth group of objects is used for the communication between connected OPC Clients and the OPC Server.

### 8.4.1 The objects and classes used in the OPC Server

#### 8.4.1.1 The objects that are used for starting the OPC Server and reading the configuration database

Several objects are used for starting the OPC Server and reading the configuration database. The following types of objects are used:

- **COPCServerApp**: Exactly one object of this type is created. It is the object that is running the application. It is responsible for creating the other objects and threads in order to start-up the OPC Server.
- **CThreadControl**: Exactly one object of this type is created. When the object is created, it created a new thread in the OPC Server. This thread is used for controlling the OPC Server application. If the user closes the application or the last OPC client disconnects from the OCC Server this thread is notified of these events. The thread reacts with stopping all other threads and destroying all objects and subsequently closing down the application.
- **CThreadOPCServer**: Exactly one object of this type is created. When the object is created, a new thread is created. This new thread is used for running the actual OPC Server.
- **BOCDaSCreator**: Exactly one object of this type is created. This object is registered to the internals of the OPC Server. It is can be used to create objects when some of the classes of the internals of the OPC Server are overridden. When the constructor of a class is called from within the internals of the OPC Server it would not be possible to override the classes from which the objects are created by the internals of the OPC Server. All objects that are created by the internals of the OPC Server are created by the registered creator object. BOCDaSCreator is a class that is derived from the SODaScreator class. This class is the default creator for the OPC Server's internals.
  In order to be able to use the classes that are derived from the default classes, a BOCDaSCreator object is created and registered globally. Now the overridden member functions of the BOCDaSCreator object are called instead of the default member functions from the SODaSCreator object. By overriding some member functions, objects from derived classes can be created instead of the default classes.
- **BOCSrvEventHandler**: Exactly one object of this type is created. It is used to override the default behaviour of the OPC Server when certain events occur.

- **BOCDaSEntry**: There is exactly one object of the class. The object holds the entry-point to the actual OPC Server. The class is derived from SODaSEntry. The entry-point is used to initialise the OPC Server, register the creator object, register the event handler and start the OPC Server.
- **SODaSServer**: There is exactly one object of this class. It runs the actual OPC Server.
- **CDaoDBConfig**: Exactly one global object if this type is created. It is used to open the configuration database and to maintain the connection with that database. All other database objects use this object to connect to the database.
- **CDaoRecordsetLocalConfig**: Exactly one global object of this type is created. It is used to read data from the local configuration database. The local configuration database is not used during the experiment, but can later on be used for storing all types of local configuration parameters.
- **CDaoRecordsetNameSpace**: Exactly one global object of this type is created. It is used to read data from the namespace tables of the configuration database. This object is not used during the experiment, but can be used later on, when for example functionality is added to the OPC Server for managing the configuration from the application.
- **CDaoRecordsetTag**: Exactly one global object of this type is created. It is used to read all tag information from the configuration database. This information is used for building the namespace of the OPC Server.

#### 8.4.1.2 The objects that are used for building the namespace

The namespace of the OPC Server is build out of several types of objects that have a pre-defined relation with each other. Figure 8-25 and

Figure 8-26 in the next section display an example of a namespace.

- **BOCDaSNameSpaceRoot**: Only one object of this class is created. It holds the entry-point of the namespace. All other namespace objects are located 'underneath' this object. The class is derived from SODaSNameSpaceRoot.
- **BOCDaSNode**: Objects of this type are used as the nodes in the namespace tree. Their parent can be the BOCDaSNameSpaceRoot object or another BOCDaSNode object. The class is derived from SODaSNode.
- **BOCDaSTag**: Objects of this type are the actual tags in the OPC Server. They are children of the BOCDaSNameSpaceRoot object or a BOCDaSNode object.
- **BOCDaSProperty**: Objects of this type are used to create properties for a tag or node. Their parent is a BOCDaSTag or BOCDaSNode object. Within the experiment each tag has a property with the description of the tag as its value.

#### 8.4.1.3 The objects that are used for communicating with the machines

A separate thread per machine performs the communication between the OPC Server and the control systems of the machines. Several objects of different types work together to fulfil their tasks.

- **CThreadRequestQueue**: Objects of this type are used to perform all communication between the OPC Server and the control systems of the machines. One object per machine is created. Each object creates a new thread that is used for handling the tasks that are needed for the communication with a machine.
- **CThreadRequestQueueHandler**: Exactly one object of this type is created. It is used to register all CThreadRequestQueue objects and the threads that are created by these objects.
- **CLanClient**: Exactly one object of this type is created per CThreadRequestQueue object. The CLanClient object is used for the actual communication between the OPC Server and the control system of the machine. It is used to set-up a connection to the machines control system and to send and receive message across the network.

- **CMessageList**: Exactly one object of this type is created per CThreadRequestQueue object. It is used to store all messages that are ready to be sent over the network to the machine's control system.
- **CTagReg**: Exactly one object of the type is created per CThreadRequestQueue object. It is used to maintain a sorted array of all registered tags for the machine.
- **BOCDaSRequestList**: Exactly one object of this type is created per CThreadRequestQueue object. The internals of the OPC toolbox place read and write requests in this list. The thread that is created by the CThreadRequestQueue gets the requests from the list and sends request messages to the control system of the machine. The machine's control system responds with sending an answer message. When the answer message is received the thread finds the request in the list and completes the request.
- **BOCDaSRequest**: Objects of this type are created by the internals of the OPC toolbox. The created objects are placed in the BOCDaSRequestList. The thread that is created by a CThreadRequestQueue object gets the requests from the list and sends request messages to the machine's control system accordingly.
- **CMessageTagInfo**: Objects of this type are received from the machine's control system. The control system sends these messages during the connection phase. They hold information about a specific tag. Objects of this type are used to update the information in the CTagReg object.
- **CMessageTag**: Objects of this type are received from the machine's control system when the data value of a tag in a Machine Control Process has changed. The thread that is created by a CThreadRequestQueue object updates the data value of the specific tag in the namespace according to the information in the CMessageTag object.
- **CMessageRequestRead**: Objects of this type are created by the thread that is created by a CThreadRequestQueue object when the internals of the toolbox placed a read request in the request list associated with the thread. The object is then send to the machine's control system.
- **CMessageRequestWrite**: Objects of this type are created by the thread that is created by a CThreadRequestQueue object when the internals of the toolbox placed a write request in the request list associated with the thread. The object is then send to the machine's control system.
- **CMessageAnswerRead**: Objects of this type are received from the machine's control system. The control system sends these objects as a response to a read request.
- **CMessageAnswerWrite**: Objects of this type are received from the machine's control system. The control system sends these objects as a response to a write request.

#### 8.4.1.4 The objects that are used for communication with OPC Clients

The communication between OPC Clients and the OPC Server is completely managed by the internals of the toolbox. The most important classes and objects are:

- **SODaSItem**: Objects of this type are created when an OPC Client wants to access a tag. For each connection to a tag that an OPC Client creates an object of this type is created.
- **SODaSGroup**: Each SODaSItem object is placed in a group. For each group a SODaSGroup object is created.
- **SODaSTransaction**: SODaSTransaction objects are created for when more than one request message needs to be sent between the OPC Server and a OPC Client. Objects of this type combine more than one message into one transaction unit.

### 8.4.2 Relations between various parts of the OPC Server

When the OPC Server is started, the first thing it does it to read the configuration database and create a namespace. This namespace holds all available tags in a tree representation. This tree is build as follows. The first node in the tree is the root. The nodes under the roots are the machine names. In the configuration database the name of a machine is associated with the IP number of the machine it identifies. The leaves of

the machine nodes are nodes with the names of the processes in the machines. The process nodes can have leaves that are tags or leaves that are nodes and hold a path. When a tag leave is created, the corresponding tag object is created too. Within the namespace tree only a pointer to the actual tag object is used.

When the OPC Server has received all tag information messages from a machines control system, it also holds a sorted list of available tags. For each message that is received, the corresponding tag has to be found in the namespace. The machine name from the configuration, the process name, the path, and the tag name are used to uniquely identify a tag in the namespace. A pointer to the found tag is placed in a resizable array of pointers. When a certain amount of tag information messages is receive, the pointers in the array are sorted with the identification number of the tag, the pointer points to, as the value that is used for sorting. When the tags are ordered in the array by their identification number the corresponding tag can be found very efficient when a message arrives for specific tag. In Figure 8-25 you can see a schematic overview of a possible situation with some tags and their organisation in a namespace and a registration array.

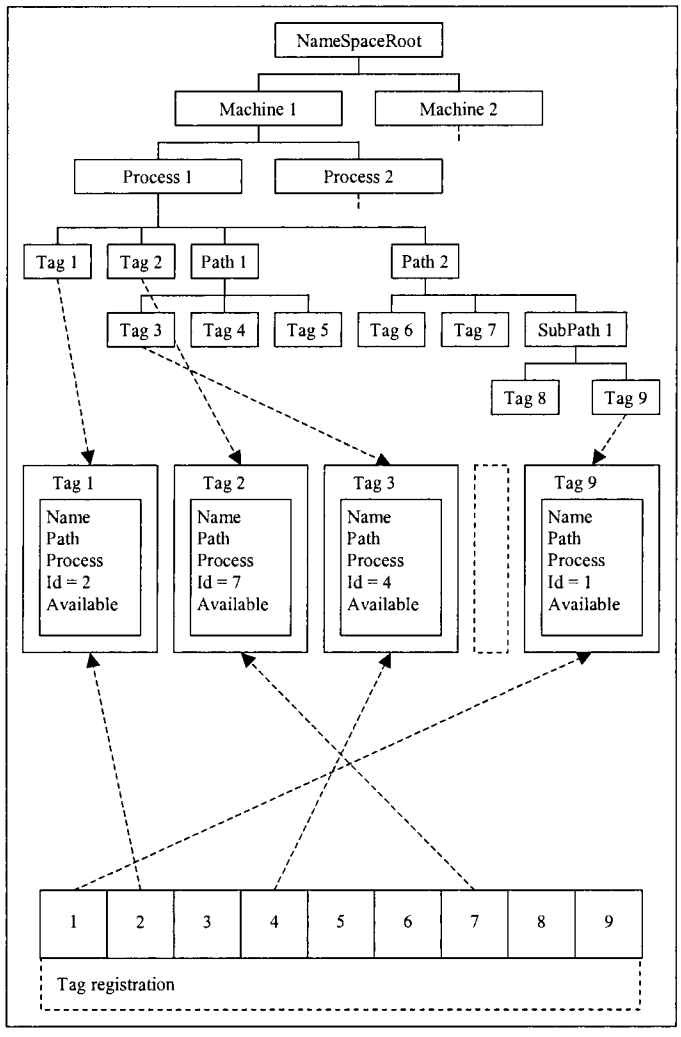


**Figure 8-25: Schematic overview of the relation between the tag registration and the namespace**

OPC Clients that connect to the OPC Server can access the tags in the namespace. A client creates one or more groups and places items in these groups. Each item is associated with a tag from the namespace. More than one item can be associated with a specific tag. An item can only be associated with one tag. A client can enable and disable a group. When a group is disabled, no data updates are sent to the client. When a group is enabled and de data value of a tag that is associated with an item in the group changes, the

new value is sent to the client. A client can have more than one enabled group. In Figure 8-26 you can see an example of the relation between tags in the namespace, groups, and items in the OPC Server.

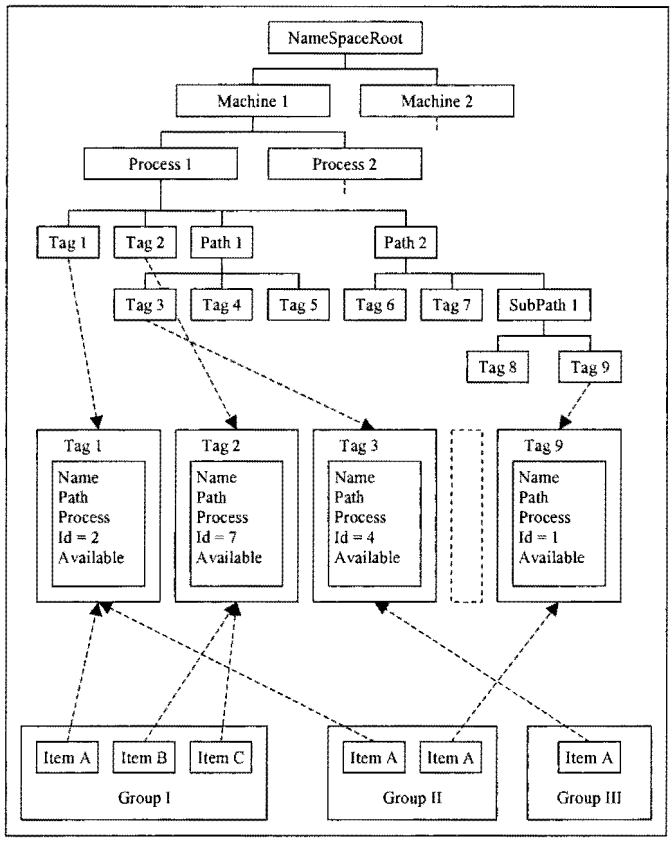


**Figure 8-26: Schematic overview of the relation between the namespace and items and groups**

### 8.4.3 Threads in the OPC server

Within the OPC Server several threads are running. Each thread performs its own specific tasks. The threads that are running in the OPC Server are:

- **Control Thread**: The control thread is started just after the application is started. It receives events when the last OPC Client has disconnected from the OPC Server or a user has closed the application. If one of these events has occurred, the control thread stops all other threads and terminates the application.
- **OPC Server Thread**: The OPC server thread runs the actual OPC server that is internal of the toolbox. The thread is started by the control thread. When the thread is started the entry point of the OPC Server is acquired and the namespace of the OPC Server is created.
- **Request Queue Threads**: When the namespace of the OPC Server is being build, the OPC server thread creates a request queue thread for each machine in the namespace. A request queue thread is responsible for handling all requests that are made for tags in the control system of the machine the thread is created for. The thread sends and receives all messages to and from the specific machine. The thread also updates the values of tag data when tag messages arrive from the control system.
- **Local Update Thread**: Besides the request queue threads that handle requests and update values from machines, another thread is created that is used to update some locally created tags. Within the experiment the local time, the local date and the server start-up time are generated locally.

All these threads are so-called worker threads. They are not associated with a visible windows component. Besides the worker threads, some so-called MFC-threads are created. These threads handle the functions

that are initiated or related to graphical components. Within the experiment there are three dialog windows: the application's main window, a window that can be used for displaying logging and debugging messages, and an about box. These windows have no specific function for the OPC Server. Their only purpose is that the main window can be used to close the application.

Within the internals of the OPC Server toolbox several threads are running for performing various tasks. More threads are created when OPC Clients connect to the OPC Server.

## 8.5 Software design of the OPC Server Process

The OPC Server Process is built of several types of objects that work together. The objects can be divided into four groups of objects that together perform a certain task in the OPC Server Process. Four functional groups can be found. One group of objects is co-ordinating all tasks in the OPC Server Process. A second group of objects is used to maintain the registration of all connected Machine Control Processes. The Third group takes care of all communication between the Machine Control Processes and the OPC Server process through pipes. The fourth group manages the communication with the OPC Server over the network.

### 8.5.1 The objects and classes used in the OPC Server Process

#### 8.5.1.1 The objects that are used for co-ordinating all tasks within the OPC Server Process

The co-ordination of the tasks is performed by one object. A few other objects are used by this object.

- **COPCServer**: Exactly one object of this type is created. It is created immediately when the OPC Server Process is started. The object creates all other objects that are used in the other groups. The object also contains the main function of the application. This main function is triggered by several signals when certain events occur. The main function reacts to these signals and performs the related tasks. It uses objects from other groups to perform these tasks.
- **CMessageRequestRead**: Objects of this type are received from the OPC Server. The OPC Server Process forwards them to a Machine Control Process. They contain a read request for a specific tag. COPCServer extracts these objects from the buffers that are received from the network communication objects.
- **CMessageRequestWrite**: Objects of this type are received from the OPC Server. The OPC Server Process forwards them to a Machine Control Process. They contain a write request for a specific tag. COPCServer extracts these objects from the buffers that are received from the network communication objects.

#### 8.5.1.2 The objects that are used for registration of the connected Machine Control Processes

Two types of objects, together maintain the registration of Machine Control Processes.

- **CProcessInfo**: One object of this type is created for every connected Machine Control Process. It contains information about a Machine Control Process. This information includes then process's name, the minimum and maximum assigned tag id's and a reference to the object that is used for sending messages to the Machine Control Process.
- **CProcessReg**: Exactly one object of this type is created. It holds a list of CprocessInfo objects. The object is used to find information about a Machine Control Process.

#### 8.5.1.3 The objects that are used for communication through pipes

The communication through pipes is done by the use of objects of one type. This class type is derived from two other classes and uses some other classes to perform the communications.

- **CPipeBuffer**: Objects of this type are used for the communication through pipes. Initially two objects of this type are created. One object is created for receiving messages through the Connect Pipe and one

object is created for receiving messages through the Priority Message Pipes. When a Machine Control Process connects to the OPC Server Process, an object of the CPipeBuffer type is created for sending messages to the Message Pipe of the specific Machine Control Process.
The object has internal FIFO queues that are used for storing messages that are ready to be sent or for storing received messages. The object takes care of combining messages into a buffer, which in turn is written to a pipe. The object is capable of reading messages from a pipe into a buffer or it can interpret the data in the buffer and place the received messages in its internal FIFO queues.

- **CPipes**: This is one of the base classes for CPipeBuffer. It is actually creating and deleting the actual pipes. It contains size and path information about the pipes.
- **CPipeEvents**: This is one of the base classes for CPipeBuffer. It is used for creating, checking and deleting the events that are used to synchronise the write operations to the pipes between the Machine Control Processes.
- **CArrayFifo**: One object of this type is created for each pipe. It is used internally by CPipeBuffer objects. Objects of this type are used to store messages that are ready to be sent or are received. This class is implemented as a template class, which makes it usable for multiple data types. This class implements a FIFO queue be using a fixed size array, see section 8.2.2 for more information about the use and implementation of FIFO queues under OS-9.
- **CMessageConnect**: Objects of this type are received through the Connect Pipe. A Machine Control Process sends an object of this type when it is setting up a connection to the OPC Server Process. It contains information about a process and the number of available tags in the Machine Control Process.
- **CMessageDisconnect**: Objects of this type are created when a disconnection from the OPC Server has been detected. The objects are sent to all connected Machine Control Processes.
- **CMessageReg**: Objects of this type are created when the OPC Server connects to the OPC Server Process via the network. The objects are then send to the Machine Control Processes. This informs the Machine Control Processes that they should start sending tag information to the OPC Server.
- **CMessageTagId**: An Object of this type is created when a connect message is received. The object is then sent to the Machine Control Process that sent the connect message. The object contains information about the tag ids that are assigned to the tags in the Machine Control Process by the OPC Server Process.
- **CMessageRequestRead**: Objects of this type are received from the OPC Server. The OPC Server Process forwards them to a Machine Control Process. They contain a read request for a specific tag.
- **CMessageRequestWrite**: Objects of this type are received from the OPC Server. The OPC Server Process forwards them to a Machine Control Process. They contain a write request for a specific tag.

#### 8.5.1.4 The objects that are used for communication with the OPC Server

The communication with the OPC Server is performed by one object.

- **CLanServer**: Exactly one object of this type is created. It creates a server for TCP/IP communications over the network with the OPC Server. This object is used to set-up and maintain the connection. All communications with the OPC Server are performed by using this object.

### 8.5.2 Relations between the various parts of the OPC Server Process

When the OPC Server Process is started an OPC Server Object is created that internally creates all other objects that are necessary for setting up the various communication channels. The OPC Server object co-ordinates the tasks that are performed by all other objects. It forwards messages and buffers from one communication object to another. It provides the process registration objects with information about newly registered processes and interprets certain messages and takes action according to the information in those messages. In Figure 8-27 you can see the relations between the groups of objects in the OPC Server Process.

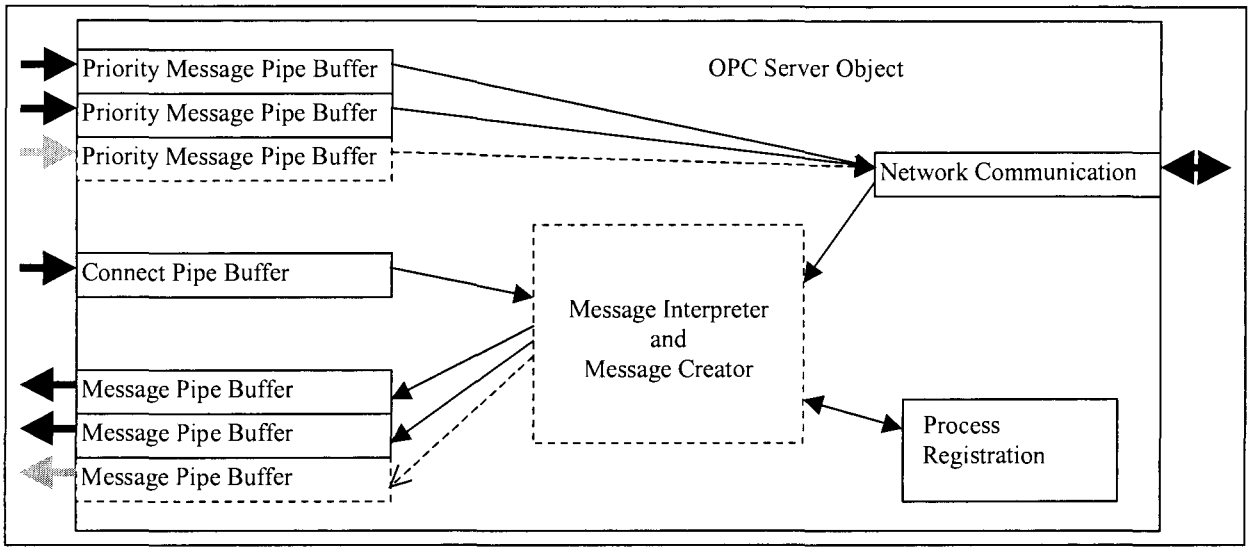


**Figure 8-27: Relations between the groups of objects in the OPC Server Process**

### 8.5.3 Execution of the several tasks

The main function of the OPC Server Object sets the process to idle when no tasks have to be performed. The process can be woken up by several signals. A signal can be installed on several events. The events that are used to generate signals are shown in Table 8-9.

**Table 8-9: Signals used in the OPC Server Process**

| Signal | Event | Remark |
|---|---|---|
| SIG_PIPE_PRIx | Data is available in the Priority Message Pipe with priority x. | A signal is installed on each Priority Message Pipe |
| SIG_PIPE_CON | Data is available in the Connect Pipe. | A signal is installed on the Connect Pipe |
| SIG_LAN_CON | A connection over the network is waiting to be accepted. | A signal is installed on the server socket |
| SIG_LAN_IO | Data is available from the network. | A signal is installed on an accepted network socket |
| SIG_ALM_SENDREQ | Alarm timer | An alarm timer is created when not all data could be written to the network or pipe. The timer is used to wait for a certain time and then perform a retry. |

If data is available in one of the pipes or from the network, the kernel of the OS-9 operating system generates a signal. The OPC Server Object's main function reacts with reading the data from the pipe or network and forwards or interprets the messages it read. If the signal indicates a connection attempt to the network server, the connection is accepted and thus the network connection is established.

If the OPC Server Process could not write data to a pipe of to the network, an alarm timer is installed. If this timer times out, a signal is generated. This signal is used as a time-out. If the signal is set the OPC Server Object's main function retries to send all data that is still in the buffers and not yet sent. If not all data could be sent, a new alarm timer is generated.

### 8.5.4 Handling the messages according to their priority

All messages in the OPC Server have a member variable that indicates their priority. The pipe buffer objects acquire this variable from the message and place in message that is added to the buffer in the FIFO queue that is related with pipe that is meant for sending message with that priority. The only pipe buffer objects that send data through more than one pipe are the pipe buffer objects that send messages through the Priority Message Pipes. The priorities of all messages that are send through the other pipes should be set to the highest priority, which is 0. This is also the default priority a message gets when the default constructor is used to create a message.

The Machine Control Processes send messages through the Priority Message Pipe. When the tag messages are created during the start-up sequence of a Machine Control Process the priorities are assigned to the messages. The priority of a message is never changed.

Tag information messages are also send through the Priority Message Pipes. These messages are given the same priority as the priority of the tag message that is associated with tag, whose information is in the tag information message. The tag information messages are always sent before any tag messages are sent. Giving the tag information messages the same priority as the corresponding tag messages ensures that the OPC Server receives a tag information message for a specific tag, and thus can register a tag, before any data is received for the specific tag.

Answer messages are also send through the Priority Message Pipes. Answer messages send the answer to a read or write request. One priority is reserved for sending answer messages. No tag messages or tag information messages are send through the pipe that is associated with the priority of answer messages.

In order to handle messages with a higher priority faster than messages with a lower priority, the OPC Server Process has to act differently for messages of different priorities. When a signal is set that indicates that data is available in a certain Priority Message Pipe, the OPC Server Object has to forward data from that pipe to the network. In order to handle messages with a higher priority faster, pipes of a higher priority are checked for available data first. If data is available in a pipe with a higher priority then the pipe associated with the set signal, the higher priority pipe is handled first. For example, the following situation occurs:

- The signal for the pipe with priority 3 is set.
- Data is available in the pipes with priority 0 and 2.
- No data is available in the pipe with priority 1.

Then the data from the pipe with priority 0 is forwarded to the network first, followed by the data from the pipe with priority 2. Finally the data from the pipe with priority 3 is forwarded.

### 8.5.5 Size of the buffers

There are several buffers in the OPC Server Process that are implemented as fixed array size FIFO queues. Because these queues have a fixed maximum size, a good choice for the size of these queues has to be made.

The buffers are used to receive message from the Connect Pipe do not receive large amounts of messages. The only messages the OPC Server Process receives through the Connect Pipe are connect messages. No more connect messages than the number of Machine Control Process can be received. So setting the size of the buffer to the number of machine Control Processes is a good choice.

The Priority Message Pipes are not used for sending messages to the OPC Server Process that are interpreted by the OPC Server Process. Therefore no FIFO queues are used to store received messages in the buffers that receive data from the Priority Message Pipes. Their size is set to zero.

The Message Pipes are used for sending request messages to the Machine Control Processes. It is not useful to have to pending requests for the same tag at the same time. It is possible that the OPC Server performs a read request for every available tag. Thus the size of the buffers is set to the number of tags that are available in the destination Machine Control Process.

## 8.6 Software design of the machine control processes

Many of the classes, the objects in the OPC Server Process are created from, are also used in the Machine Control Processes. The objects in the Machine Control Processes can be divided in three groups. The first group is co-ordinating all tasks of the Machine Control Process. The second groups takes care of all communication with the OPC Server Process through the pipes. The third group maintains the registration of all tags in the Machine Control Process.

### 8.6.1 The objects and classes used in the Machine Control Processes

#### 8.6.1.1 The objects that are used for co-ordinating all tasks within a Machine Control Process

The co-ordination of the tasks in a Machine Control Process is performed by one object.

- **CControlApp**: Exactly one object of this type is created, the Machine Control Object. It is created immediately when the Machine Control Process is started. The object creates all other objects that are used in the other groups. The object also contains the applications main function. This function is triggered by several signals when certain events occur. The main function reacts to these signals and performs the related tasks. It uses the objects from the other groups to execute these tasks.

#### 8.6.1.2 The objects that are used for registration of the tags

Two types of objects, together maintain the registration to the tags in the Machine Control Process.

- **CClientTagReg**: One object of this type is created for each tag in the Machine Control Process. They are created at start-up. Each registration object holds the information about a tag. This information includes the tag's name, the tag's path, and the tag id that is assigned to the tag.
- **CClientTagList**: Exactly one object of this type is created. It holds all tag registration objects. The Machine Control Object uses this object to find a tag by its tag id.

#### 8.6.1.3 The objects that are used for communication through pipes

The communication through pipes is done by the use of objects of one type. This class type is derived from two other classes and uses some other classes to perform the communications.

- **CPipeBuffer**: Objects of this type are used for the communication through pipes. Three objects of this type are created. One object is created for sending messages through the Connect Pipe, one object is created for sending messages through the Priority Message Pipes and one object is created for receiving messages from the Message Pipe.
  The object has internal FIFO queues that are used for storing messages that are ready to be sent or for storing received messages. The object takes care of combining messages into a buffer, which in turn is written to a pipe. The object is capable of reading messages from a pipe into a buffer or it can interpret the data in the buffer and place the received messages in its internal FIFO queues.
- **CPipes**: This is one of the base classes for CPipeBuffer. It is actually creating and deleting the actual pipes. It contains size and path information about the pipes.
- **CPipeEvents**: This is one of the base classes for CPipeBuffer. It is used for creating, checking and deleting the events that are used to synchronise the write operations to the pipes between the Machine Control Processes.
- **CArrayFifo**: One object of this type is created for each pipe. It is used internally by CPipeBuffer objects. Objects of this type are used to store messages that are ready to be sent or are received. This class is implemented as a template class, which makes it usable for multiple data types. This class implements a FIFO queue be using a fixed size array, see section 8.2.2 for more information about the use and implementation of FIFO queues under OS-9.

- **CMessageConnect**: Objects of this type are send through the Connect Pipe. A Machine Control Process sends an object of this type when it is setting up a connection to the OPC Server Process. It contains information about a process and the number of available tags in the Machine Control Process.
- **CMessageDisconnect**: Objects of this type are created when a disconnection from the OPC Server has been detected. The objects are sent to all connected Machine Control Processes. They are received by the Machine Control Process through the Message Pipe.
- **CMessageReg**: Objects of this type are created when the OPC Server connects to the OPC Server Process via the network. The objects are then send to the Machine Control Processes. This informs the Machine Control Processes that they should start sending tag information to the OPC Server. They are received through the Message Pipe.
- **CMessageTagId**: An Object of this type is received through the Message Pipe when and is sent by the OPC Server Process. The OPC Server Process sends this object after it has received a connect message. The object contains information about the tag ids that are assigned to the tags in the Machine Control Process by the OPC Server Process.
- **CMessageRequestRead**: Objects of this type are received from the OPC Server Process. The object contains information about a read request.
- **CMessageRequestWrite**: Objects of this type are received from the OPC Server Process. The object contains information about a write request.
- **CMessageTagInfo**: Objects of this type are created when information about a tag needs to be sent. The object holds information about a tag. This information includes the name of the process's name, the tag's path, the tag's name, the tag's data type, and the id that is assigned to the tag.
- **CMessageAnswerRead**: Objects of this type are created when a read request is received. The object is filled with the response data to the read request and sent to the OPC Server Process.
- **CMessageAnswerWrite**: Objects of this type are created when a write request is received. The object is filled with the response data to the write request and sent to the OPC Server Process.
- **CMessageTag**: One object of this type is created for each tag in the Machine Control Process. The Machine Control Object creates these objects immediately when the process is started. Each object is reused every time the data value of a tag changes. The specific object is sent to the OPC Server Process.

### 8.6.2 Relations between the various parts of a Machine Control Process

When a Machine Control Process is started a Machine Control Object is created. This object creates all other objects that are used in the Machine Control Process. The Machine Control Object co-ordinates all tasks that are performed by all other object. The main function of the Machine Control Object reacts to the occurrence of several signals that are generated by certain events. The Machine Control Object handles the messages that are received by the communication objects. The Machine Control Object sends messages when a tag's data value changes, or as a response to request messages. The Machine Control Object also sets up the connection with the OPC Server Process and updates the information in the tag registration object.

Besides handling all messages and performing all tasks that are related the OPC Server, each Machine Control Process has to execute several functions for controlling the machine. The tag values throughout the entire OPC Server are generated inside the machine control functions. The machine control functions are not part of the Machine Control Process. They are normally global functions that can be called from the process's main function. The task of the process's main function has to be implemented in the main function of the Machine Control Object. Thus the main function of the Machine Control Object is an integration of the original main function of the Machine Control Process without any OPC Server tasks and the main function of the Machine Control Process without any machine control tasks. When the data value of a tag changes, the machine control functions can place the message that is related with that tag in the pipe buffers of the Message Priority Pipes. It is also possible to write one or more member functions of the Machine Control Class that updates the tag messages and places them in the buffers. Figure 8-28 shows the relations between the different groups of objects in a Machine Control Process and the machine control functions.

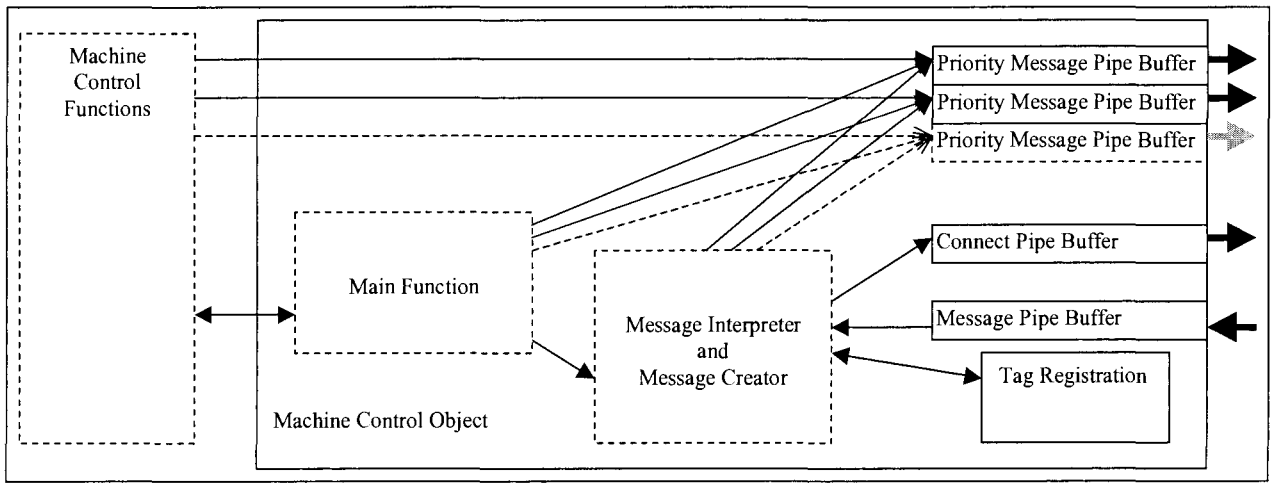


**Figure 8-28: Relations between the groups of objects in a Machine Control Process**

### 8.6.3 Execution of the several tasks

The main function of a Machine Control Process sets the process to idle when no tasks have to be performed. The process can be woken up by several signals. A signal can be installed on several events. The events that are used to generate signals are shown in Table 8-10.

**Table 8-10: Signals used in a Machine Control Process**

| Signal | Event | Remark |
|---|---|---|
| SIG_PIPE_MSG | Data is available in the Message Pipe | A signal is installed on the Message Pipe |
| SIG_ALM_MSG | Alarm timer | An alarm timer is created when there are still messages in the receiving buffer of the Message Pipe |
| SIG_xxx | A machine control function has to be performed | The signal can be set by an alarm timer, a hardware interrupt, another process, etc. |

When data is available in the Message Pipe the Machine Control Process reads the messages from the pipe into the related buffers. Subsequently it interprets the messages and takes action accordingly. When a lot of messages have arrived, the process could be busy handling all these messages for a relatively large amount of time. In this time, no machine control functions can be executed. A maximum number of messages that may be handled at a time is defined. If the maximum number of messages is handled and there are still messages available, an alarm timer is created. If the time the timer runs has elapsed, the according signal is set. When this signal is received the Machine Control Process restarts handling of the messages that are still in the buffer until all messages are handled of the maximum number of messages that may be handled in one time is reached again. In the latter situation, again an alarm timer is created.

Many more signals can be received. On the reception of these signals certain machine control processes are executed. The Machine Control Object starts these functions when the related signal arrives.

### 8.6.4 Handling messages according to their priority

The Machine Control Processes do not receive any messages with different priorities. All messages that are sent through the Priority Message Pipes are send through the pipe associated with their priority. When the data value of a tag has changed, the related tag message is placed in the FIFO queue of the priority pipe buffer object. When all message that need to be sent are placed in the buffer, the messages are sent. The messages with the highest priority are sent first, followed by the messages with the second highest priority.

### 8.6.5 Size of the buffers

There are several buffers in the Machine Control Processes that are implemented as fixed array size FIFO queues. Because these queues have a fixed maximum size, a good choice for the size of these queues has to be made.

No more than one message at a time is send to the Connect Pipe. Thus size of queue in the buffer associated with the Connect Pipe can be set to 1.

The queue in the buffer that is used for receiving messages from the Message Pipe is set to the same size as the size of the queue in the buffer that is used for writing message to the Message Pipe in the OPC Server Process. This size is equal to the number of tags in the Machine Control Process. The Machine Control Process is then able to receive a request message for each tag at same time. Normally the OPC Server does not make that amount of requests, but it is possible. A badly configured OPC Client can cause to OPC Server to make even more requests. In that case the queue size may be too small and should be enlarged.

The queue sizes in the buffer that is used for sending messages to the Priority Message Pipes must be set according to the number of available tags of the corresponding priority. The size of the buffer that is used for messages with priority 0 is set to the amount of available tags with messages of priority 0. The same applies for the other priorities. This size is large enough, because it is not useful to have more than one message per tag in the queue at the same time. If this should be the case, the value from the first message would immediately be overwritten be the data from the second message. In addition to this, only pointers to the actual messages are stored in the queue. All tag messages are reused in the Machine Control Process. Changing the data value of a message overwrites the old value. If the message is already in the queue the data value is overwritten, too. In this case the old value is already lost. Adding the message to the queue again places the same message with the same data value in the queue twice. This not useful. The tag messages have a data member indicates if the buffer is already in a queue. Thus, no more tag messages than the number of tags of the specific priority need to be in the queue. The only situation where the Machine Control Process may want to place more messages than tags in the queues is after the sending of the tag information messages. These messages are also sent through the Priority Message Pipes. Each tag information message is sent through the same pipe as tag messages for the corresponding tag go through. As many tag information messages as tags of a certain priority are available are placed in the corresponding queue in the buffer. When not all tag information messages are send and the Machine Control Process wants to send tag messages, to buffers may be too small. This only occurs just after the OPC Server has connected to the control system. So it is not necessary to enlarge the buffers. In order to make sure that all tag messages can be sent, the Machine Control Process should wait with adding tag messages to the buffer until all tag information messages are written to the Priority Message Pipes.

One of the Priority Message Pipes is reserved for sending answer messages. The size of the queue that is used for this is set to the same size as the queue size of the Message Pipe, which is used for receiving request messages. The size of the Message Pipe queue is set to the number of available tags in the Machine Control Process. Thus the size of the queue that is used for answer messages is set to the same size.

## 8.7 Preventing the occurrence of blocking writes

All Machine Control Process sent message to the same pipes. When a process is writing data to a pipe, this write operation must be finished before any other process can write data to the pipe. If a process tries to write data to a pipe while another process is already doing the same, the process blocks and has to wait until the other process has finished its write operation.

Another blocking occurs when a process tries to write more data to a pipe than the amount of free space in the pipe. In this case, the write operation blocks until another process has read data from the pipe and enough free space is available. Then the write operation is completed.

Machine Control Processes are not allowed to block for a long time, because no machine control tasks can be performed when the process is blocked. When a lot of processes try to write to the pipe and the process is the last one that is activated, it has to wait until all processes are finished. If at the same time the pipe gets full, first another process has to read data from the pipe, which increases the time the process is blocked even more.

As a result of the stated problem, a protection mechanism has to be built in to prevent the Machine Control Processes from blocking. Only checking for available free space is not enough. There could still be a lot of processes waiting to be allowed to write data to the pipe.

Because the combining of messages from the queue and placing them in the actual buffer that is used for writing the messages to the pipe also can take considerable time, it is advisable that it is always possible to write the resulting buffer to the pipe immediately after it is filled. If this is possible, it results in predictable behaviour and the a good estimation for the maximum time the process can be busy combining messages and writing them to the pipe can be made.

The combine and write process can be made exclusively accessible be introducing an event with two values, a so-called binary event. A process can wait for an event. If it gets access to the event, its value is raised. No other process can get access to the event at the same. When the process has access to the event releases it, another process can get access to it. This mechanism can be used to protect a resource from being accessed by more than one process at the same time.

If this mechanism is used to ensure that only one process is writing to a pipe, a process can have to wait a long time, because a lot of processes can be waiting for the event. If the combining of the messages is encapsulated in the protected code block, the waiting time can be even larger than with only blocking writes.

It is possible to check the state of an event without waiting for it. A process can first check the state of an event and if it is available decide to wait for it. If the event is not available the process can decide from the function and send the messages later on. This method works fine most of the time. However there is always a risk that another process is enabled between the check for the state of the event and the wait for the event. If this happens and the enabled process sets the event and is scheduled out before the event is reset, the first process will still block on wait function for the event. The result is that the combining and sending of the messages can take twice the time it is allowed to take.

To prevent the last stated problem to occur a second event could be used. The wait and release functions of the second event are placed around the check and wait functions for the first event. Each process waits for the second event. If it gets access, a process checks the state of the first event and if its available gets access to it. Whether access is allowed or denied, the release function of the second event is called. Now the next process can enter the same check and wait loop. This process will find out that the first event is not available. The code between the wait and release functions of the second event is very small en takes a very short time to execute. As a result the waiting processes do not have to wait a long time before they can enter the piece of protected code.

# 9. TESTING THE OPC SERVER FOR OS-9 CONTROLLED SYSTEMS

After the different parts of the OPC Server for OS-9 Controlled systems were created according to the hardware and software descriptions in the chapters 7 and 8, I tested the working of the OPC Server. The first tests should show that the OPC Server has the required functionality and that the chosen design can work. Next the OPC Server had to be tested with the performance requirements from section 6.5 through 6.8 in mind. The OPC Server should be able to handle the amount of tags that is calculated in section 6.6 with the accompanying update rates and data types. In addition to the constant data stream the OPC Server is also tested for its burst behaviour. The OPC Server should be able to handle the various types of burst data as calculated in section 6.8. Finally the jitter of the data stream in the OPC Server is tested in several situations.

After the functional tests are completed and the OPC Server has been tested against the performance requirements, the maximum throughput of the OPC Server is determined.

## 9.1 Testing environment

The following hardware and software was used to test the software of the OPC Server.

### 9.1.1 Control system

The OS-9 operating system was running on a PG2056/60 from Philips/Nyquist. This is the same board as is used as the main controller board of the ILF and NCCW. The processor on this board is a Motorola 68040, running at 33 MHz. The board is equipped with 8 MB DRAM. The operating system on this board is OS-9 version 3.03. No machine control functions are performed during the tests. The only running processes during the tests are the OPC Server Process and the test processes provide the tags and generate the data.

### 9.1.2 Windows system

The PC that is running the OPC Server is an IBM PC 300 XL with an Intel Pentium II processor at 233 MHz. This PC is equipped with 80 MB DRAM. The PC is running Microsoft Windows NT 4 with service pack 6a installed. The OPC Client that is used is Softing OPC Toolbox Demo Client version 3.04. This client is used to see if the data generate in the control system is available to an OPC Client. The client is not connected to the OPC Server during all tests, because the OPC Client takes a lot processor time when it has to display a large amount of tags.

### 9.1.3 Network

The control system and PC are both connected to the office network of BOC Edwards PS. The control system is equipped with a 10 Mbit Ethernet network card and the PC with a 100 Mbit Ethernet network card.

## 9.2 Functional tests

For the first functional tests, two identical processes are created. Both processes provide 40 tags of different data types. All data types from Table 8-8 are used, except VT_EMPTY and VT_ERROR. The update rate of the tags is set to 1 Hz. For each tested data type 5 tags are created. 5 priority levels are used for the messages that are being sent. Priority 2 is reserved for answer messages that are sent as a response to a read or write request. Later the update rate is increased to 10 updates per second.

To test the functionality of read and write requests, the update rate is set to 0 updates per second. The test processes do not send any tag messages. They only react to read and write request messages by sending answer messages. The test client is used to generate read and write requests.

### 9.2.1 Results of the functional tests

The OPC Server receives all data updates. Each time a data value is updated in the OPC Server a timestamp is added. This timestamp is made visible in the test client. When the update rate is set to 1 per

second, every second a new data value should arrive in the test client for each tag. This can easily be observed visually.

All tags are visible in the client. When the update rate is set to 1 per second, the data of every tag is updated every second. To check if the software that runs in the control system can keep up with the data stream and does not get behind, the size of the Priority Message Pipes can be checked. The size of the pipes can be checked by using the simple 'dir' command from the command line. The size of the pipes should always be near 0. Maximally one block of messages may be in each pipe. If this is the case, the OPC Server Process is capable of forwarding all messages from the pipes to the OPC Server. The pipes are almost empty all of the time.

When the update is raised to 10 per second, it is much more difficult to see if all updates are received by the test client. Optically everything seems to work fine. When looking at the size of the pipes in the control system, they are almost empty all of the time.

When no tag messages are sent and the test processes only react to request messages, the answers to the requests are received in the test client.

### 9.2.2 Conclusion of the functional tests

The OPC Server is capable of handling all data types the messages and is capable of handling request, answer and tag messages. So all messages are supported. The messages that are used inside the control system also work well, because if there was something wrong with the registration procedures inside the control system, the communication with the OPC Server could not work as a result of the failure.

## 9.3 Performance tests

### 9.3.1 Maximum throughput

Bepalen maximale throughput

Hoe gedraagt het systeem zich onder grote load

Testprocessen met veel data

Met teveel data, wat gaat er mis?

Hoe is het burst gedrag

## 9.4 Jitter tests

Hoe is de jitter in het systeem onder normale omstandigheden

Wat verandert er bij hoge load en bursts

Gebruik timestamp in OPC Server en schrijf die naar een logfile

## 10. CONCLUSIONS

10.1

10.2

## 11. RECOMMENDATIONS

During the experiment not all aspects of the OPC Server have been implemented yet. Some handy functionality is missing and some improvements can be made. This chapter describes some of the improvements that can be implemented.

### 11.1 Sending keep-alive messages between the several parts of the OPC Server

During the experiment no functionality has been build in to check if all communication channels are still working. So-called keep-alive messages could be added to check this. If keep-alive messages are sent at certain times an answering message is received within a certain time, the connection is still working. If the answering message is not received within a certain amount of time, the connection is considered broken. If a connection is broken, certain action could be taken to re-establish the connection or the exclude the specific channel from further communications.

### 11.2 Using array data types

The use of arrays for reducing the overhead in messages, because data blocks are relatively large and headers relatively small when the use of arrays is compared with the situation without the use of arrays. A disadvantage of using arrays is that not all OPC Clients and SCADA systems are able to handle arrays. This reduces the possibilities of the use of arrays. A solution where arrays can still be used in parts of the system is when the OPC Server transforms an array variable in multiple variables.

Adding functionality to the OPC Server to effectively split arrays into single variables can be done in the following way. If a variable is defined as an array variable in the configuration database, the item created for the variable is not a tag object, but a node object. An extra field must be added to the configuration database. This field holds the number of items that are in the array. For each variable in the array a tag is created. All tags have the created node as their parent. The tags are numbered according to the variable's position in the array the represent. The data from the variables in the array is copied to the created tag, where each tag receives the data from one single position in the array.

### 11.3 Adding a learning mode the OPC Server

If the OPC Server received tag information messages from the control system with information for a tag that is not in the namespace, the message is deleted and nothing is done with the information. A learning mode can be added to the OPC Server. In the learning mode, the OPC Server sends a learning mode message to the control system. The OPC Server Process forwards this message to all Machine Control Processes. Each Machine Control Process reacts with sending tag information messages for all tags in the Machine Control Process. The OPC Server does not delete the messages, but uses the information in the message for adding a record to the configuration database. This way the configuration database can be acquired from the control system and does not need to be entered by hand.

### 11.4 Disabling tags

OPC Client most of the time do not display all tags. Normally only visible items are enabled. If no enabled items are related to a tag, data updates for the tag are send to the tag, but nothing is done with the new data. If the SODaSItem class of the toolbox is overridden, it is possible to determine that no items that are related to a certain tag are enabled. In this case a disable message can be send to the control system to disable the tag in the control system. When an item is enabled, an enable item can be send to the control system and the tag in the control system enabled. By default all tags in the control system can be disabled.

references

[CMP EVA] *Comparison between VxWorks x86 5.3.1, QNX 4.25 and pSOS x86 2.2.6, Comparison between QNX RTOS v6.1 and Windows CE 3.0, QNX 6.1 Evaluation Report, pSOS x86 2.2.6 Evaluation Report, VxWorks x86 5.3.1Evaluation Report, Windows CE 3.0 Evaluation Report,* Dedicated Systems, Brussels, 2001

[IMP FBA] *Implementing Functional Block Adapters*, T. Heverhagen, R. Tracht, University of Essen, Essen, 2001

[INT UML] *Integrating UML-RealTime and IEC 61131-3 with Function Block Adapters, IEEE Int. Symposium on Object Oriented Real-time Distributed Computing*, T. Heverhagen, R. Tracht, 2001

[MS DCOM] *The Component Object Model Specification*, Microsoft, 1995

[MSDN] *Microsoft Developers Network Library*

[OPC DAC] *OLE for Process Control; Data Access Custom Interface Standard Version 2.05*, OPC Task Force (www.opcfoundation.com), 2001

[OPC OVW] *OPC Overview, Version 1.0,* OPC Task Force (www.opcfoundation.com), 1998

[OS9 INS] *OS-9 Insights; an advanced programmer's guide to OS-9, second edition,* P. Dibble, Des Moines (Iowa), 1992

[RT SS] *Real-time software systems: an introduction to structured and object-oriented design*, J.E. Cooling, London, 1997

[RTOS EV] *Real Time Operating Systems: Evaluation results*, Philips CFT and High Tech Automation, Eindhoven/Bilthoven, 1997

[RTOS MS] *RTOS Market Survey: Preliminary Results*, Martin Timmerman, Dedicated Systems, Brussels, 2000

[SDM RTS] *Software Design Methods for Concurrent and Real-Time Systems*, H. Gomaa, New York, 1993

[STR RTS] *Strategies for real-time system specification*, D.J. Hatley and I.A. Pirbhai, New York, 1987

[UML MT] *UML Modeling Tools: executive summary*, Yphise, Lewisville, 2000

[UML RM] *The unified modeling language reference manual*, I. Jacobson, G. Booch and J. Rumbaugh, Amsterdam, 1999

[UML UG] *The unified modeling language user guide*, I. Jacobson, G. Booch and J. Rumbaugh, Amsterdam, 1999

[UNI SDP] *The unified software development process*, I. Jacobson, G. Booch and J. Rumbaugh, Amsterdam, 1999

[UST UML] *Using Stereotypes of the Unified Modelling Language in Mechatronic Systems*, T. Heverhagen, R. Tracht, University of Essen, Essen 2001

## APPENDIX A: CLASS DIAGRAMS

## APPENDIX B: USE CASES

**SUMMARY**

In September 2001 I started my graduation project at BOC Edwards Pharmaceutical Systems. My project comprehended a contribution to a new generation of control systems.

I started with making an inventory of the machines that BOC Edwards Pharmaceutical Systems produces. The control systems of these machines vary a lot. The simplest machines can be controlled by relays. Most of the machines are controlled by PLC's and an industrial computer controls the most complex machines.

One of the problem areas that was identified is the software design method. With such a variety of control systems it is almost impossible to use one software development method that is optimal for all control systems. On the other hand it is not advisable to use more than one software design method, because this would require a lot of training and schooling of software engineers.

In the ideal situation one modular control system would be used for all types of machines. This control system must be capable of controlling the most complex machines. Thus a PLC is not sufficient. This implies that such a modular control system should be based on an industrial computer. Customers often demand the use of a certain brand of PLC on the machines they order. This combination of requirements hinders the use of one modular control system.

The main board of the newest control system that is based on an industrial computer is already 8 years old. The company that delivers the main board is not doing any development according to controller boards for industrial computers any more. This could lead to problems with the availability and support in the future. Therefore I investigated the possibilities to upgrade the current control system and together with that investigated which operating system to use.

One of the big problems with control systems that are based on an industrial computer is that there is no standard solution to integrate them with SCADA systems. I investigated several solutions. One of these solutions is to develop an OPC Server, which is capable of communication with current SCADA systems, for control systems that are running OS-9. During my graduation project I developed such an OPC Server for OS-9 Controlled Systems.

I first investigated the context in which such an OPC Server would have to function. After I had finished that investigation, I set up the requirements the OPC Server would have to satisfy. These requirements are based on a future machine, that is an integration of two of the current machines that are produces by BOC Edwards Pharmaceutical Systems. The requirements include functional and performance requirements.

The OPC Server for OS-9 Controlled systems consists of several parts. One part is running on computer that is running Microsoft Windows. This application communicates with the SCADA systems. The other parts of the OPC Server are running in the control system. They are OS-9 processes. The communication between the Windows application and OS-9 takes places over TCP/IP over Ethernet.

The software development of the various parts of the OPC Server is done with the use of UML and the case tool Rational Rose. The software of all parts is constructed using classes and is programmed in C++. The use of UML is new to BOC Edwards Pharmaceutical Systems. It is used in the experiment to demonstrate that is can be used for developing software for their control systems.

The Machine Control Processes generate all data that is provided to a connected SCADA system. The OS-9 OPC Server Process functions as a communication buffer between the Windows application and the Machine Control Processes.

After I developed the various parts of the OPC Server for OS-9 Controlled Systems I did some tests to check if it is functioning according to the requirements. The OPC Server for OS-9 Controlled Systems meets the functional requirements. Due to the fact that the full version of the toolbox did arrive only one week before the end date of my graduation project, the performance, burst behaviour and jitter tests could not yet be performed. These tests need to be done in the future.

## 1. INTRODUCTION

BOC Edwards Pharmaceutical Systems is a company that produces machines for the pharmaceutical industry. Their product range varies from washing machines and sterilisation tunnels to fluid fillers to loading systems. All these machines have their own control system. Together with the variance of the complexity of the different machines the complexity of the control systems also varies a lot.

BOC Edwards Pharmaceutical Systems manufacturers their machines on a project-oriented base. Customers have great influence on design decisions that have to be made. This also involves the decisions that have to be made for the control systems of the machines. This brings about that for every new machine that is being build new problems can rise and have to be solved.

I started my graduation project in September 2001. My task was to make an inventory of the current situation concerning the control systems that are used to control the machines, to study trends in de development of the control systems and to propose some improvements. Subsequently I implemented one of the proposed improvements. This graduation report describes this various stages of my graduation project at BOC Edwards Pharmaceutical Systems.

## 2. PROBLEM STATEMENT

I started my graduation project at BOC Edwards Pharmaceutical systems with generating an inventory of the current situation concerning the control systems of the various machines they produce. BOC Edwards Pharmaceutical Systems produces a variety of machines. Some have very simple control systems, some very complex.

The most complex machines are controlled by an industrial computer that is running a real-time operating system. This operating system is OS-9. All machines that are controlled by a PLC can be integrated in a SCADA system.

SCADA systems are used to control and monitor production lines at a higher control level. The control systems in a machine are used to actually control the machine. If more machines together form a production line, the control systems are not directly coupled. Where each machine's control system is responsible for controlling the machine, a SCADA system is used to control the machines at a higher level. A SCADA system can control and monitor all machines in a production line. All PLC manufacturers provide a standard solution for integrating their PLC with current SCADA systems. Such a solution is not available for machines that are controlled by an industrial computer that is running OS-9.

During my graduation project an experiment that shows a solution for integrating control systems that are based on OS-9 with SCADA systems was performed. The requirements for a possible solution are set up, based on a future machine that will be developed by BOC Edwards Pharmaceutical Systems in the near future. Most current SCADA systems can communicate with an OPC Server. A possible solution for integrating the machines that are controlled by an industrial computer is to design an OPC Server for these control systems. Such a server enables BOC Edwards Pharmaceutical Systems to integrate their most complex machines with SCADA systems. After I designed an OPC Server for OS-9 Controlled Systems, I also performed some tests to show that the OPC Server provides the required functionality.

The chapters 3 and 4 describe the different machines BOC Edwards Pharmaceutical Systems produces and identify the problem areas. The chapters 5 through 8 describe the design of the OPC Server for OS-9 Controlled Systems. The chapters 9 through 11 describe the results of the tests that were performed to test the design and give conclusions and recommendations.

## 10. CONCLUSIONS

### 10.1 Software Design Method

Currently BOC Edwards Pharmaceutical Systems is using the software design method of Hatley and Pirbhai for the development of all software designs for the control systems of their machines. The software design method of Hatley and Pirbhai separates the data and processes. Performing software design, using the method of Hatley and Pirbhai results in a lot of parallel processes and data stores. In PLC software also a separation of data and processes is visible. Thus the method of Hatley and Pirbhai is very well suited for developing software for PLC's.

More complex machines are not controlled by a PLC, but by an industrial computer. This industrial computer is running a real-time operating system, OS-9. A more complex machine also has a more complex software design. As a result a lot of processes are generated using the software design method of Hatley and Pirbhai. Running many processes and using many inter-process communications can dramatically decrease the performance of a control system based on an industrial computer and a real-time operating system. Thus, the results of the software design method of Hatley and Pirbhai need to be transformed into processes and communications in an efficient way.

Software design methods that are better suited for developing software for control systems that are based on an industrial computer and a real-time operating system are object-oriented software design methods. The best parts of several object-oriented software design methods are brought together in the Unified Modelling Language, UML. UML has become a standard for complex software development. The method creates classes and objects for real world objects during developing the software design. Communication between the objects is represented in various diagrams as well as the execution order of several tasks.

PLC software does not support the use of objects that hold data and operations on that data. Implementation of object-oriented programming is not a problem on industrial computers. There are several object-oriented programming languages available. This is nor the case for PLC's. When using UML for software development, this results in an object-oriented design. However, UML can be used for developing PLC software. During the design phase, two types of objects need to be created. One type of objects is used for storing data. These objects only store data and do not contain any functions. The second type of objects only holds functions and no data. The results of such a design can be implemented on a PLC.

Where UML is better suited for developing complex software for real-time operating systems, the method of Hatley and Pirbhai is better suited for PLC software. It is not advisable to use different software design methods for different control systems. If this is done, it would mean that software engineers need to now how to implement both methods. This would require a lot a training and schooling. Most machines that are produced by BOC Edwards Pharmaceutical Systems are controlled by PLC's. Thus it is a good choice to stay with the method of Hatley and Pirbhai. If in the future more machines are equipped with a control system based on an industrial computer, if is advisable to introduce UML.

### 10.2 Control Systems

In the ideal situation all control systems would have the same base design. It should be modular. More complex machines require more complex control systems than relatively simple machines. The base of the control system should be usable on the most complex machines. This means that the control system should be based on an industrial computer and not on a PLC, because a PLC does not have the possibility to control the most complex machines.

Customers often require that a machine is controlled by a certain brand of PLC. They do often not accept the use of a control system that is based on an industrial computer for controlling a relatively simple machine. The customer's demands keep BOC Edwards Pharmaceutical Systems from using the same control system for all machines.

### 10.3 Hardware and Software for Complex Machines

Nowadays, there are a lot of real-time operating systems. All of these operating systems have their advantages and disadvantages. Performing a technical comparison of real-time operating systems does not point out a clear winner. VxWorks is the most used real-time operating systems worldwide. Almost all hardware manufacturers provide support and/or drivers for VxWorks. In the world of machine control applications OS-9 is still one of the large players. Thus, there is no need to change to another operating system than OS-9.

The newest control system that is based on an industrial computer is based on a 68040 processor from Motorola. The main board of this control system is already 8 years old. The company that produces these boards does not do any new development of controller boards for industrial computers. This might lead to problems with the availability of the product in the future.

Therefore it is advisable to look for a new control system, to use on new machines. There are a lot of hardware manufacturers that produce parts for an industrial computer. Most of the current controller boards for industrial computers are bases on a Motorola Power PC or an Intel Pentium processor. Although there are many different boards on the market with a great variety of features and performance differences, technical arguments are not the most important arguments for selecting the parts of an industrial computer. The control systems for machines that are produces by BOC Edwards Pharmaceutical Systems do not need the newest available hardware and the highest possible performance. Decisions made on product life cycle and the number of suppliers are much more important.

I investigated 4 different solutions. Two of those are completely based on products supplied by MEN. The rack, the controlled board and all interface cards are manufactured by Men. The other two solutions are based on product from Motorola and Tews. Various suppliers, supply components from both manufacturers. Choosing one of these solutions results in a limited number of suppliers and in a design of the control system that is up to date.

### 10.4 Supervisory Control and Data Acquisition

Besides the man machine interfaces on the machines themselves, machines can also be monitored and controlled by a higher-level control system, a so-called SCADA system. SCADA systems are organised as a client-server applications. One or more servers provide one or more clients with data. Clients can be used to control or monitor one or more machines in a production line. The integration of several machines often takes place via a SCADA system.

All PLC manufacturers provide a solution to integrate their PLC's with SCADA systems. Such solutions are not available for control systems based on an industrial computer running a real-time operating system. The most complex machines that are produced by BOC Edwards Pharmaceutical Systems are equipped with a control system that is running OS-9. Several solutions are possible to integrate such a control system with a SCADA system. During my graduation project I performed an experiment with developing an OPC Server for OS-9 Controlled Systems. OPC is an interface standard. Most SCADA systems can communicate with an OPC Server. They acquire their data from an OPC Server. During the experiment I developed an OPC Server that is running on a computer that is running Microsoft Windows. The OPC Server communicates with the control system based on OS-9 and uses it as its data source. This solution give BOC Edwards Pharmaceutical Systems the possibility to integrate machines with control systems based on industrial computers with SCADA systems.

### 10.5 OPC Server for OS-9 Controlled Systems

Before the design of the OPC Server for OS-9 Control Systems could be started I first set up the requirements that the design would have to meet. These requirements are based on a future machine, that is an integration of two of the current machines that are produced by BOC Edwards Pharmaceutical Systems. The requirements include functional and performance requirements.

The OPC Server for OS-9 Controlled Systems that was developed during the experiment consists of several parts. One part is running on a computer that is running Microsoft Windows. This part of the OPC Server is developed using a toolbox. This toolbox is Softing OPC DA SDK toolbox 3.04. It consists of several libraries

that provide the OPC interface. The application that is generated with the use of the toolbox provides an interface to OPC Clients. On the other side the application communicates with OS-9 control system.

In the control system, one process, the OPC Server Process, is responsible for communicating with the OPC Server application that is running on a Windows computer. This process communicates with all Machine Control Processes in the control system. These Machine Control Processes generate the data that is provided to the OPC Server application. The OPC Server Process functions as a communication buffer between the OPC Server and the Machine Control Processes.

All parts of the OPC Server for OS-9 Controlled Systems are programmed in C++. All code is implemented using classes. The software development is done using UML, with the use of the case tool Rational Rose Enterprise Evaluation Edition.

During the design of the OS-9 processes several design decisions needed to be made. One of the main decisions was the use of pipes for interprocess communication. Another design decision was the implementation of FIFO queues on OS-9. Because problems can rise when the standard implementation of FIFO queues is used, an alternative had to be found. A working solution is the implementation of FIFO queues with a fixed size array. Because the main purpose of Machine Control Processes is to control the machine, these processes are not allowed to block or to be busy with other tasks for a too long time. A special protection mechanism is built in to prevent the Machine Control Processes from blocking. This mechanism uses two events to protect a pipe and its related writing routines from being accessed by more than one process at a time.

After the several parts of the OPC Server were designed I performed some tests with it. The results of the first tests show that the OPC Server is working and provides all required functionality.

Unfortunately the full version of the toolbox did arrive only one week before I write this. Thus not all tests are performed yet. Performance, burst behaviour and jitter tests still need to be performed.