Contract Number ICA4-CT-2002-10012
Efficient Operation of Urban Wastewater Treatment Plants

Software main frame able to integrate and manage the system Deliverable D8.3

Deliverable type: Report
Number: D8.3

Date of delivery: 1st March 2005
Task: WP8.3

Responsible:

Paolo Ratini
SPES scrl
Via Broganelli 84
60044 Fabriano (AN)
Italy
Email: paolo.ratini@spesonline.com

Other Contributors:

Massimiliano Pirani
SPES scrl
Via Broganelli 84
60044 Fabriano (AN)
Italy
Email: massimiliano.pirani@spesonline.com

Andrea Quintabà
SPES scrl
Via Broganelli 84
60044 Fabriano (AN)
Italy
Email: andrea.quintaba@spesonline.com

Abstract: This deliverable is the paper presentation of the first version of the software main frame and integrated systems that have been developed in the EOLI project. It presents a summary of the work done and decisions taken. The bulk of the work was in the design and implementation of the EOLI systems, of which this paper deliverable is a record and a reference document for the project.

A high-level conceptual presentation of the system's architecture and of the data flows between its components is given, relating to components within a single system. These flows are then described in more detail at a technical level in terms of the DATABASE and XML formalisations used for internal and remote system interfaces respectively. A summary of the work done in developing the system is presented with a brief scenario of the current implementation achievements.

TABLE OF CONTENTS

1 Introduction

This deliverable corresponds to the first complete and functioning EOLI system. It describes the systems developed in terms of the software main frame —that is, the EOLI developments required to integrate some functional modules such as software sensors, controllers and fault detection and isolation.

Other work packages are developing particular functional components such as smart (software) sensors and fault detection systems, controllers and isolation algorithms. The purpose of WP8 is to integrate them into a whole which is generic and customisable.

The purpose of this deliverable is to record the fundamental decisions that have been taken concerning the architecture of the systems to be produced within the EOLI project. Recall that the essence of the EOLI project is the control of a wastewater treatment plant (or multiple plants) both locally and from a remote centre. The architecture falls into three major divisions: hardware, software and interconnection and remote access. The key issues addressed within each of these divisions are:

- Hardware: the design of a digital platform to allow the coexistence and co-working of several sensors, probes and actuators as a whole system controlling a plant.

- Software: the design of the EOLI software to allow coexistence and co-working of the EOLI modules as a whole monitoring and control system.

- Interconnection and remote access: the nature of the connections between the local plant and the Internet, the remote control centre and the Internet, and the exchange of data between them.

A further purpose of this deliverable is to propose system management strategies for allowing a single remote control centre to manage multiple WWTPs in parallel.

The structure of the deliverable follows the sequence set out above.

There are two pilot plants in the EOLI project: the pilot scale plant at ENEA (Italy), located at the site of a full scale municipal wastewater treatment plant, and the pilot scale plant at INRA (France). The development work has focused mostly on the first of them. The validation and integration activity are planned to be implemented at the INRA pilot plant during the last six months of the project.
With the aim at testing the EOLI supervision system on different architectures and for the purpose of plant integration testing, UNAM and UU have developed their own systems for the monitoring and control of their lab plants and IbTech has built up a full scale plant: these plants have to be considered as examples of architectures compliant with the EOLI software supervision system, able to be managed in parallel with the two EOLI pilots plants by the same remote control centre. This will enforce the policy for exploitation and dissemination of the results of the EOLI project: the software main frame and the whole monitoring and control systems developed feature general capabilities and are customisable to many other types of EOLI compliant installations.

2 Hardware architecture

An off-the-shelf solution, used by most plant owners, is available: it consists of a PLC (Programmable Logic Controller) used to interface the sensors and actuators to the PC. The seeming advantages are the standardisation of components and the market availability, but it could be an expensive solution and not so easy to use and maintain for a small enterprise. There might not be a PC technician at a small enterprise. Consultancy from a PC technician is not a negligible cost; it might be accepted for installation and at start-up, but rejected during normal operation if too frequent.

In any case this solution could not really be plug-and-play and above all it is not as modular as it might appear: the addition of a new component to the system (a new sensor or actuator) produces a complexity increase which is different from one case to another and not predictable in advance, because it depends on the kind of the component.

In the EOLI project another approach is addressed: the developement of a WWTP digital platform. It means:

- To equip each sensor and actuator with an embedded electronic slave board that has the task to acquire and perform a first conditioning of the signal from the sensors and to operate the actuators and, if required, to carry out periodical operation of diagnosis, such as detection of faults, and auto-calibration.
  The Basic features of this electronic slave board are:
  - 16 bit microcontroller architecture
  - 12 A/D 24 bit resolution channels ,with galvanic isolation signal amplification and noise reduction implemented on board;
  - local area links and connections:
    RS232 serial link
    RS485 serial link
    RF (DECT standard or 2.45GHz) wireless link

- The embedded electronic main board of the platform is equipped with an integrated PSTN modem, with an ethernet card and with four serial RS232 interfaces to exchange data and commands with a PC and possibly a PLC. So the main board can be also linked to a server machine located at the remote control centre directly through a telephone line or by LAN or WWW network, or definitely using a serial or RF connected PC as a bridge to those networks.
  The communications between the main and other slave boards are made via serial RS485 or RS232 link or by RF wireless link. The information transmitted to and by slave boards which equip the sensors and actuators are gathered (for storing or logging) or sent by the main board exclusively on a digital support (viz. digital media).
  The summary of the basic features of the main board are:
  - 32 bit microcontroller architecture with embedded OS onboard
  - human interface: touch-screen
  - local area links and connections:
    RS232 serial link for PC and PLC connection
    RF (DECT standard or 2.45GHz) wireless link
    USB link
  - remote links: PSTN modem link
  - ethernet card
  - backup data storage: Compact Flash Card

This hardware architecture allows considerable costs reduction since the following commercially available components are no longer required:

- pH or ORP signal amplifiers
- DO signal amplifiers
- Pt100 signal amplifiers
- A/D converters

Moreover it dramatically simplifies the framework and the software management of the whole system. Actually for the EOLI set of sensors we have only one kind of wiring (RF or RS485) with only one protocol to manage at the EOLI level. From the communication point of view, the PC is the master for the main board of the plant: at fixed time interval the EOLI software requires data, via RS232 serial port, from the board. In this way the PC will only implement one communication protocol and will interface only one hardware board (Figure 1).

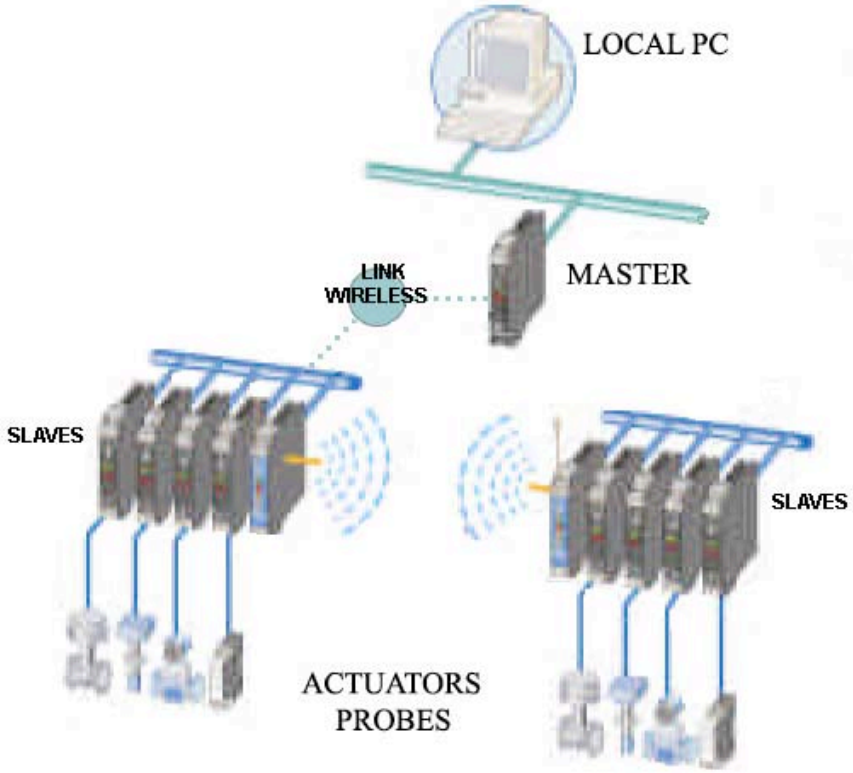

Figure 1

This solution is really modular, because any new component (sensor or actuator) of the system can be added by connecting a "cable" (see the "wireless link" with the slave board) and assigning it a code (the address) for the enrolment into the EOLI software. The increase of complexity so remains a priori defined as the same enrolment procedure applies to any kind of sensor. It is a real plug and play solution because the only installation requirement is that the slave boards have to be connected to the sensors and the main board to the PC via RS232 (or by the other links). Then the enrolment of the sensors is performed into the EOLI software, with no need to modify or to interact with the PC operating system or to run set-up or installation software packages.
Furthermore this solution is thus even simpler from the hardware point of view (the number and kind of devices involved are significantly less) and reasonably more robust.

The slave and the main boards can be enclosed in suited box to guarantee their performance and security issues. As far as reliability, standardisation and maintenance is concerning, these boards are developed and produced according to the international industrial requirements for standards and certifications.

The use of the digital platform could also offer the possibility -for example- to test remotely a new sensor on a plant without the use of a PC to store the data: they can be transmitted to a remote server by Internet, using the embedded modem and a telephone line; or to enrich the remote data base with data from sensors present on the plant, but not integrated into the local monitoring and control system at all.

3 Software Architecture

The EOLI software environment has been designed and implemented according with the requirement specifications assessed in the deliverable D8.1. It provides all the functionalities needed for monitoring and control.

The EOLI software architecture is based on the client/server technology in which different modules implemented for different purposes, can share the same resources using a relational database management system (DBMS) that is introduced to replace the limitations of the file sharing architecture: the client/server technology is a versatile, message-based and modular infrastructure that is intended to improve usability, flexibility, interoperability, and scalability.
In addition, this architecture reduces network traffic by providing a query response rather than total file transfer and improves multi-user updating through a Graphical User Interface (GUI) front-end to a shared database. Finally the database server is also characterized by a multi-level system access with different grants for each different client that is connected to it.
In this communication system the Remote Procedure Calls (RPCs) or standard query language (SQL) statements are typically used to communicate between the client and server.
The modules can be developed in any programming language that is able to open a socket towards the database server and this assumes a wide number of possible developments.
In Figure 2 is reported a schema of the whole architecture.

The core, both at remote control centre and at the local plant, is a database implemented in MySQL technology and is used to store all the data gathered (measurements), generated (commands), transferred and managed by the EOLI system.

At the local plant level, the main frame software is organized in modules.
There is a low level layer, the LAYER0, developed in Microsoft Visual C++, based on a multithreading approach, to emulate parallel run of different tasks on the same machine, that serves the basic tasks of the system:

- interaction with Windows OS;
- communication via serial RS232 links with the master board (and with a PLC if needed);
- management of the database access;
- gateway implementation via LAN link for the connection with the remote web server;
- system manager administration.

Software modules can be enrolled into the system to implement the following features and functionalities:

- synoptic module, for an overall display of data and plant status;
- data plot module, for graphical data representation;
- plant console module, for manual management of the plant;
- modules developed for monitoring and control, such as software sensors, controllers, FDI.

They can be used as compiled (i.e. executable program) modules.

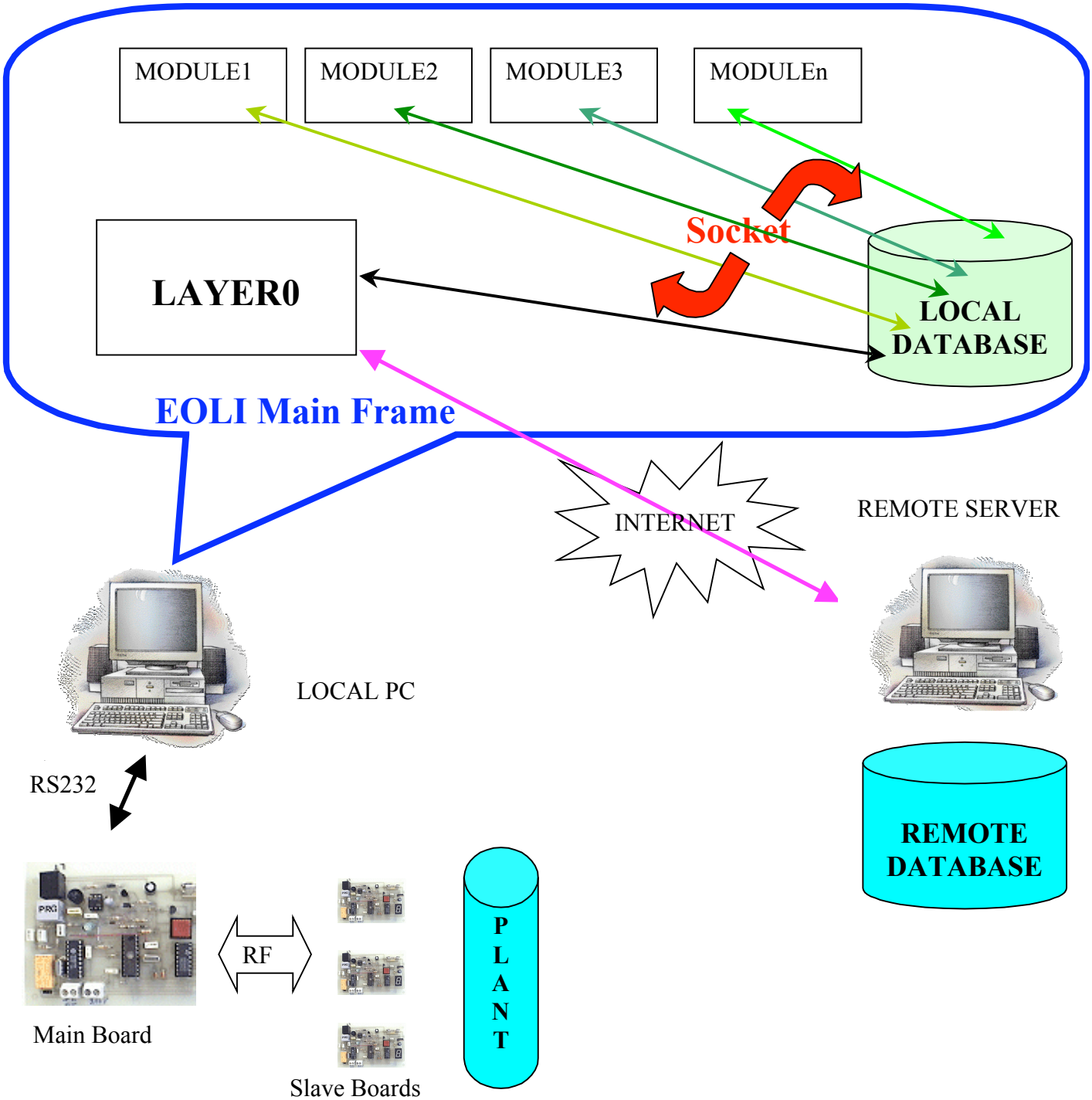

Figure 2

At the local plant level, the access to the local database is controlled by the special module named LAYER0, that constitutes also the interface with the hardware level composed by the electronic boards (the MASTER and the SLAVE ones) that are as the physical media to acquire data or to apply commands to the bioprocess plant.

The LAYER0 module communicates steadily with the MAIN board which monitors the plant, acquires the measurements from the probes and applies the control actions to the actuators. Moreover the LAYER0 is also able to send the data (XML FORMAT) to a remote machine, located at the remote control centre, in order to allow remote operators to view these data through internet technology, and in addition to process the remote command that must be applied to the local plant.
For the interaction with the local database, inside the EOLI software main frame, a module must obtain valid credentials. This operation is made by LAYER0 that is able to assign different grants for each table in the local database. So a module could be able only to recover data from a table while other modules can also concurrently write in this same table. In this way, different modules have different grants in relation to the role that the module has in the local plant control system.
The server database is based on a multithread architecture, in which multiple connections can be satisfied from different clients at the same time without any problem.
Each client doesn't know and doesn't care about the number of connections that are active because the server opens a single connection thread for each incoming request and eventually shuts down the ones that it finds corrupted.
An internal thread is responsible of checking multiple attempts of writing on the same table to prevent possible conflicts as a vigilant of the traffic: the first request that is arrived is the first to be satisfied without conflicts.
The server database is also multi-user and different modules with different grants can work concurrently for theirs purposes.
The LAYER0 reads all the tasks (control actions) to be applied at the plant, that the modules schedules in the local database and communicate them as soon as possible (the speed of the action queue depends on the setting of the fundamental update time for the current local software) to the hardware architecture: so the server database is the unique intermediary between the different collection of modules.
In the EOLI architecture all the information that a properly working module must handle, can be retrieved at any moment from the local database since any transferred data to and from the plant remains written in the database.

The general schema for the integration of the several modules in the software main frame is focused on how MySQL queries are used to implement a powerful method for the on-site integration of the EOLI system.
Because of MySQL is a server database, it is possible to take/put data using queries, through a TCP/IP socket, from/to several processes running on the same PC where the database is implemented or from/to processes running on a PC connected by LAN or Internet and located anywhere. This means that whatever MATLAB module built up to implement a software sensor, a controller and the FDI system - and in general any compiled modules, with no matter concerning the development environment developed to perform analysis or evaluation on the plant data - can run in parallel interfacing the database, for data exchange, using queries on a TCP/IP socket: no matter is concerning the location of the PC on which they run if a TCP/IP socket for the connection with the software main frame is available.
The connection and data exchange through MySQL queries is subjected to the use of an encrypted password.

The co-operation between different modules is really effective - even if they are resident on different machines and in different places - because they would interact in the same way they would do if they were implemented at the same site: by the way of the MySQL server database.
Moreover, in this way, a team of researchers or a consultant company could test and run MATLAB modules, providing them the input data and forwarding their outputs to the pilot plants by the way of queries to the MySQL server database with no need for sending and receiving and parsing data files. This would significantly simplify the data exchange and above all would reduce the time spent to transfer data, enabling an interaction very close to real-time.

In this way we could also imagine, for the future applications of the EOLI system, a consortium of different institutes or research centres that work and implement the MATLAB modules for several plants and companies from their own facilities.

Each module has to be equipped with routines able to load and manage the ODBC drivers used to implement the queries for data exchange with the MySQL server database. This is the interface each module needs to interact with the EOLI system.
Examples of these routines are presented for MATLAB and for other development environment in the Appendix D.
The structure of the EOLI database is presented in the Appendix C.
They have to be considered requirement specification of the hardware and software implementation for the integration.

This approach for the integration makes easier the development, the implementation and the update of the modules devoted to monitoring and control: this software architecture supplies the developer the possibility to use the most suited environment for the task to implement.
Really each module could be developed in any software environment, once equipped with the interface for the data exchange, and then used as compiled modules. This of course simplifies also the use of the Matlab modules for the implementation of the supervision system because of this software architecture removes the need of the presence of the Matlab run time engine.

4 Interconnection and remote access between Remote Server and WWTP

The EOLI project is based on the idea that each remote WWTP is connected to a Remote Server Machine – located at a remote control centre - in order to upload measured data and alarms and download control actions. This Remote Server is essentially a supervisor system which implements a distributed control architecture. It has been chosen to use Internet as the main connection infrastructure between the Remote Server and each plant; this is because of the flexible, inexpensive and widespread nature of this medium. The use of Internet as a communication media raises a certain number of issues to be solved:

- the physical and logical interconnection of the plant to the Internet;
- the physical and logical interconnection of the Remote Server to the Internet;
- the nature of data exchange between the plant, the Remote Server and a remote user (the expert at a remote site);
- data formats;
- robustness issues;
- hardware and software platforms;
- security issues.

We will analyse in further detail these points.

4.1 Plant connection to the Internet

If a PC located at a wastewater treatment plant needs to access the Internet, a unique address (IP address) must be assigned to it, in order to identify it on the network. This address can be a static or dynamic one. The most common solution for an end-user is a dynamic address, which is assigned on dial-up by the Internet Service Provider (ISP) among the pool of addresses it has already registered. A static IP address means that the ISP must assign this address only to a specific end-user, and it is more expensive and not all ISPs are willing to do that (or have enough registered addresses). The most important factor in this choice is that plants could be located in remote places of the earth, where the ISP cannot have the possibility or the technical resources to assign a stastic address. This would be an important limitation of the usability of the EOLI software that we cannot accept. Security issues are also more complex to fulfil for a static IP address.

Another important choice we have to make is about the physical connection to the ISP; there are also different possibilities:

- **xDSL connection with fixed IP address.** It is an always on connection: the Remote Web Server can thus access every remote plant at any time, imposing control actions and reading in real time all the data from the sensors. This is, of course, the best solution from the flexibility of the architecture point of view, but there are many important drawbacks we have to consider:
  - xDSL are not widespread on the territory and are usually found in big towns and not in rural areas.
  - Every plant owner (e.g. distilleries, diary industries etc.) would need to buy a pool of IP addresses.
  - A xDSL router is needed.
  - This kind of connection is difficult to manage from an end-user with little expertise about the Internet.

- **xDSL connection with dynamic IP address.** In this case, the ISP assigns the IP address, with a nearly instantaneous connection time, simulating an always-on connection. The drawbacks are the same as before, with a slightly less cost.

- **Dialup connection.** This is not an always-on connection, but it is the logic on the plant which decides when to make a phone call to the ISP and then, through the Internet, connect to the TCC. In other words, the Remote Web Server has not the possibility to access every remote plant at any time, but only when the plant decides to call. This can be done regularly with a fixed frequency or when the logic on the plant detects a dangerous situation (like an alarm) and decides to call the Remote Web Server to warn it.

  The external hardware needed is just a modem, which is very cheap and easy to use: as from the end-user point of view he/she needs only a telephone plug to the local Public Switched Telephone Network (PSTN).

The EOLI project has decided to opt for a dynamic IP address assigned on the fly by the ISP.

4.2 Remote Server connection to the Internet

The Remote Server is the server of the entire system. It is based on a database: it collects data from every remote plant and sends specific control actions to the plants. By its nature, it must be an always-on server and have a fixed Internet address (and optionally it would be nice also to register a name by the Domain Name System), in order to be accessed by every plant at any time and also by every authorised user intending to access its database. From a physical point of view, an xDSL connection or an Ethernet link to an already Internet connected network could represent feasible solutions.

4.3 Data exchange

We will distinguish two kinds of accesses to the Remote Server: data access from one of the plants and web access from remote users authorised to do it.

- **Data access.** The PC on the plant will make a dial-up call with a fixed frequency to the ISP and then, once logged on the Internet, connect to the server (this is possible because we know its unique IP address). Data are then uploaded to the server in XML format, which is becoming a de-facto standard for data transmission thought the Internet. The Remote Server gets this data, checks for its correctness (data integrity is an important issue to fulfil), extracts the useful information from the XML format and sends all them directly to the Remote Database, implemented using MySQL technology.

  At this point, once the upload has been terminated correctly, the client asks for a database table from the Remote Server where all the actions that must be performed on the specific plant are previously queued. Once the client has got and executed it, it can send the results of these actions to the Remote Server. If no other actions are necessary, the connection will be shut down.

- **Web access.** The easiest and most effective way a user can access the databases and interact with a remote plant, is using a conventional Web browser (like Internet Explorer or Netscape Navigator); this will thus be an HTTP access through an http port of the Remote Server. The main advantage of this solution is that we avoid installing a specific program on the remote PC to connect. Using the Web browser it will be possible, by means of tables and other formatting elements, to access to the requested data. Again, a server scripting language, will extract and present this data in a Web Page.

4.4 Hardware and software platform

From a hardware point of view, the server can be a PC with suitable resources. The operating system will be one of the Microsoft Windows family; we will decide later on, depending on the developments and requirements of the project. The Web server will be again a widespread and standard piece of software: This software is developed and implemented using an open source environment: Apache is used as Web Server, the XML parser and the Internet pages are implemented using PHP technology.

There are no particular special requirements on the PC to be used for connecting to the Remote Web Server as a remote expert.

4.5 Security issues

It is appropriate to make a few remarks here. Security issues are aimed to prevent an unauthorised user to perform harmful actions on the plant and also protect data stored in the remote database at the Remote Server. There will be different requirements for the two different kind of accesses to the Remote Server: from the remote expert and from the remote plant.
In the first case, it is very important to protect the connection from the outside world to the Remote Server; there must be a robust access verification procedure and, if thought useful, an encryption of the connection.

Less important is the security between the local plant and the Remote Server; having a dynamic IP address, it will be very difficult to trace this connection: it would be like find a needle in a haystack and, honestly, we do not see any reasons to do that.

5 System management strategies

An important feature of the EOLI project is the prospect of managing multiple WWTPs from a single Remote Server. This has several advantages:

- Efficiency and cost reduction.
- Concentration of expertise at a single centre; the remote experts do need to travel, and can share their expertise equally across the multiple WWTPs.
- The possibility of taking advantage of the data and knowledge accumulated from multiple WWTPs to improve the management of each of them individually.

Some issues arise concerning the management of multiple plants in parallel:

- Application of knowledge acquired from the managing of multiple plants.
  There is the possibility that the larger amounts of data obtained from monitoring multiple plants will also give rise to general conclusions about plant behaviour.
- Prioritisation and presentation of the handling of multiple plants.
  Prioritisation refers to the situation that several plants require urgent attention simultaneously: it is possible that there might be contention for processing power or for remote access, if many plants are attempting to make contact simultaneously. Clearly these risks are serious only if there are a considerable number of plants being managed simultaneously: if there are only two, it will not be a problem; if there are 20 or more, it might well be.

These are illustrated schematically in Figure 3
At this stage, the aim is to point out the options and approaches that will be considered during the life of the project for dealing with these issues. It will not be possible to be more definite until some empirical experience is gained in the performance of the EOLI system.

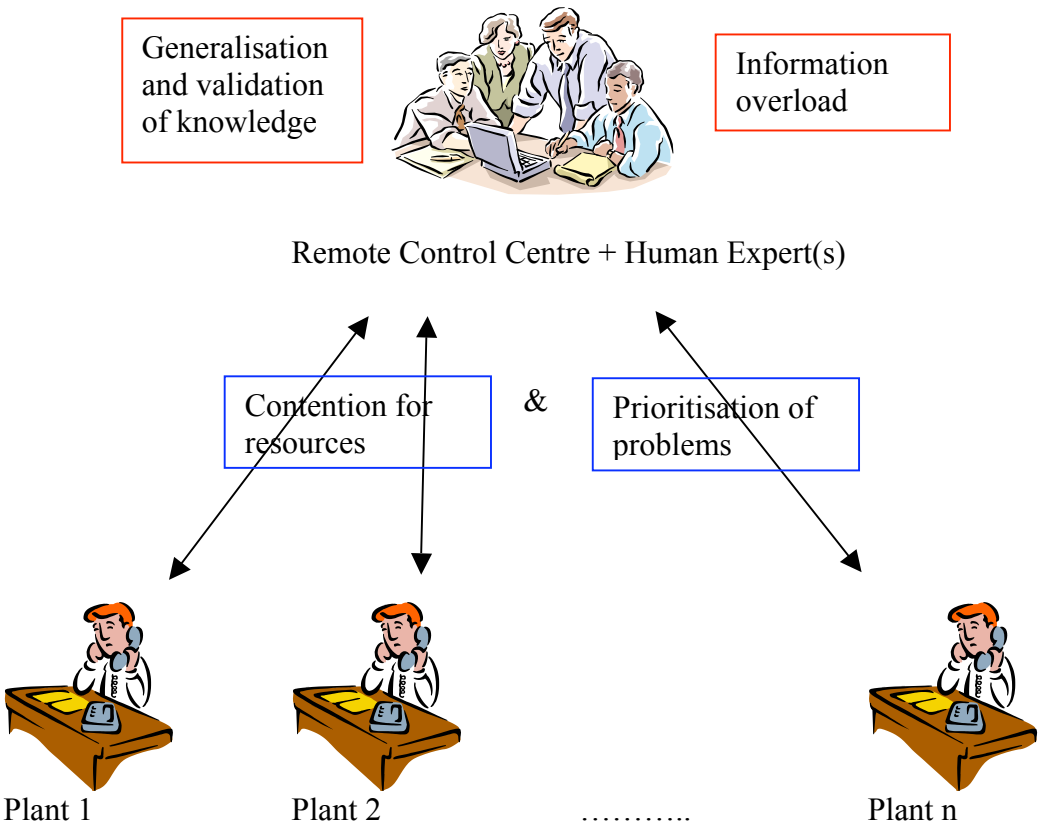

Figure 3

6 High-level specifications of data flows

Figure 4 shows an overall view of the components of the EOLI integrated system and the existence of data flows between them. The system it depicts is the most complete envisaged by EOLI; at any particular installation some of the components might be absent. The installation may be a subset of that shown in the diagram, and for that reason the diagram can be taken as a definitive, maximal representation of what the EOLI system is.

The aim of the diagram is to show the functional units of EOLI and the ways in which they interact.

The diagram shows three human actors—the local technician, the remote control centre technician and the remote expert[1]. It is worth summarising the decisions that the project has taken about the responsibilities and interactions of these three.

1. The remote experts are delocalised both from the plant and the control centre: they will always be able to access the data from the Remote Server at the remote control centre, and may or may not be able to take actions on the plant(s) - this will require different levels of authorisation, controlled by username and password. They will interact using a Web interface. Reconfiguration of the plant will not be enabled for the remote experts.

[1] In fact there may be multiple remote experts; they will then form a 'network of expertise' available to assist with problems on a plant.

2. In addition, there may be technicians located at the remote control centre. They will interact through a graphical user interface at the Remote Server. Then it is the responsibility of any particular installation of EOLI to decide on its policies for the responsibilities of remote control centre technicians and remote experts, bearing in mind issues such as the reliability of connections to the plant.

3. In order to avoid problems of time delays as far as possible, there must be careful control of the frequency of connection from the plant to the Remote Server.

4. The validation of any action by the local plant must be possible. However, in some cases if the local expertise is very weak, the local experts will not be asked to validate any action. On the contrary, most of the actions that the local expert can undertake may be disabled.

The arrows in Figure 4 indicate the existence of a flow of data between two components of the EOLI system.
Some of the arrows can be dealt with in general terms. Obviously the arrows that link the plant itself to the sensors and actuators are outside the scope of the EOLI project.

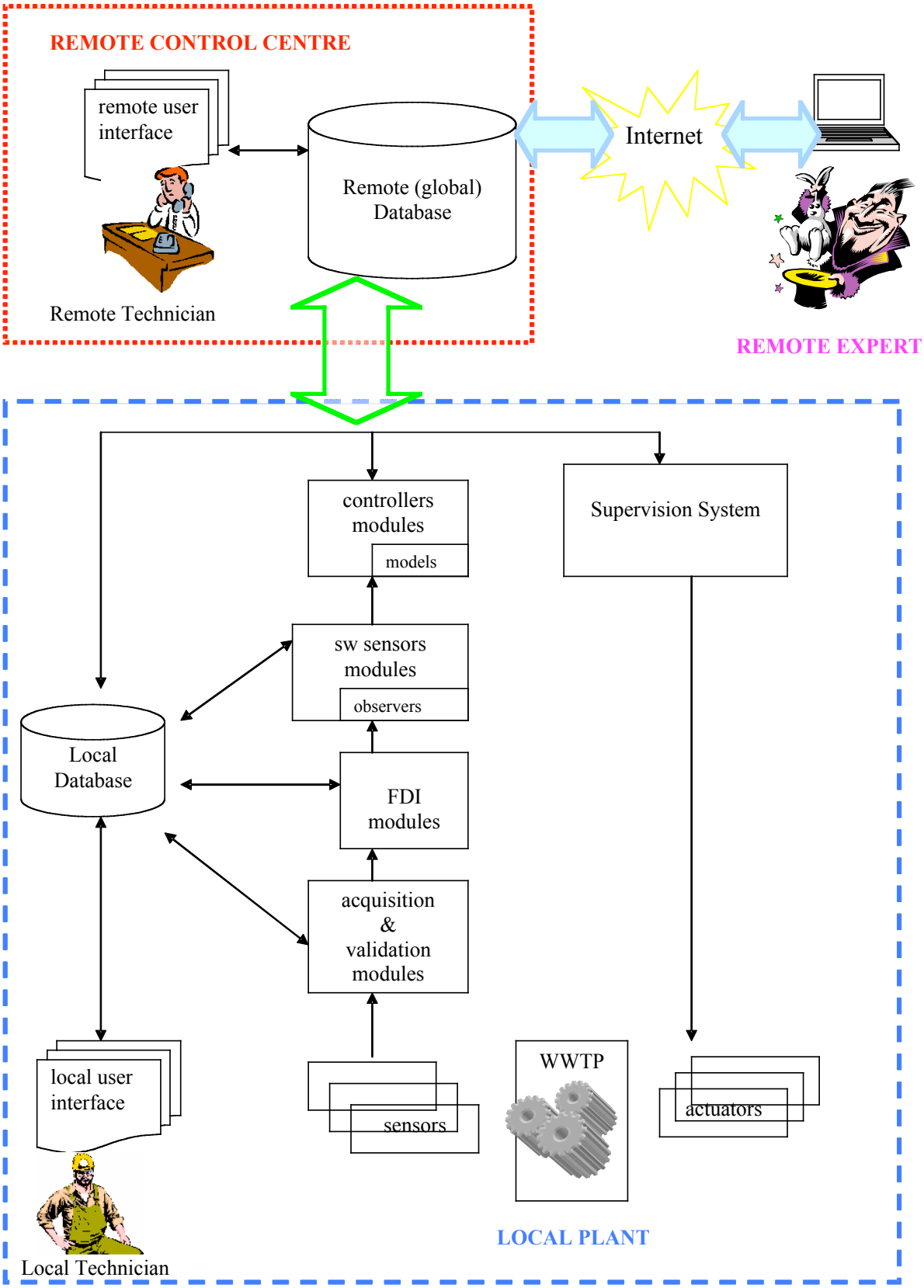
REMOTE CONTROL CENTRE
remote user interface
Remote (global) Database
Internet
Remote Technician
REMOTE EXPERT
controllers modules
models
Supervision System
sw sensors modules
observers
Local Database
FDI modules
acquisition & validation modules
local user interface
sensors
WWTP
actuators
LOCAL PLANT
Local Technician

Figure 4

7 Appendix A: XML format

Introduction
Here is the full and final definition of PlantML, the ad-hoc XML language used to exchange data between distant modules or modules based on different software environments. An earlier version was presented in D5.2a; the present deliverable specifies the final version after careful study and initial experience of its use. The language is described in a file called a DTD, which defines the form of the mark-up, and is included in an annex to this report.

PlantML is a language which can be used in the domain of bioprocess or more widely chemical engineering; it can be used to describe information about acquisition, diagnosis, decision, control and exchange this information between distant modules or operators.

Syntactical convention : a mark-up is written in lower case, except first letter of second word constituting the mark-up if it exists.
Example : <**row**>, < **bioProcessinfo** >
A PlantML document is structured as illustrated below; the body of the message is located between mark-up <**plantML**> and </**plantML**>.
There are a plantMLUpload (Figure A1) from local machine to remote web server and a plantMLDownload (Figure A2) from remote web server and the local machine.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plantML SYSTEM "PMLFull.dtd">
<plantMLUpload>


</plantMLUpload>
```

Figure A1: Basic structure of a PlantMLUpload file

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plantML SYSTEM "PMLFull.dtd">
<plantMLDownload>


</plantMLDownload>
```

Figure A2: Basic structure of a PlantMLDownload file

The content is organised in five main sections:

- Administrative information: **<header> </header>**
- Plant architecture (physical configuration) **<bioProcessinfo> </ bioProcessinfo >**
- Data characterizing the process (sensors, off-line measures, observers results, simulations): **<acq> </ acq>**
- Actions concerning the process (control,command,setpoint,param): **<action></action>**
- Events concerning the process (calibration,software upload) : **<maintenance></maintenance>**

7.1.1 Common mark-up

<actor></actor>
An actor is a physical person who schedules an action or a maintenance. According to situation, all attributes are not necessary, typical uses are shown in Figure A3.

```
<!ELEMENT origin (nameActor, namePartner, address, email, phone, role, priorityMax)>
<!ELEMENT nameActor (#PCDATA)>
<!ELEMENT namePartner (#PCDATA)>
<!ELEMENT priorityMax (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT role (#PCDATA)>
```

Figure A3: Definition of an *actor*

Priority level is dynamically evaluated according to status and age of last information when decision was taken:

- A local actor has a maximal priority of 4. He has a direct knowledge of the plant and can have to stop the plant.
- A remote actor has a maximal priority of 3.
- Local software has a maximal priority of 2.
- Remote software has a maximal priority of 1.

<comment></comment>
This mark-up is used to add free comments.

<address></address>
It provides the description of a postal address

<date>
A date is characterized by a standard ISO format: yyyy-mm-dd HH:MM:SS
Figure A4 gives an example.

```
<date>2005-04-01 00:00:00</ date >
```

Figure A4 : an example of date definition

<**row**>
The mark-up row is used to wrap a set of data that are referred to the same information. Figure A5 gives an example.

```
<row>
        <idVariable>S0</idVariable>
        <idType>V2</idType>
        <unit>°C</unit>
        <min>0</min>
        <max>0</max>
        <nameEquipment>T Input</nameEquipment>
        <shortName>T Inp</shortName>
</row>
```

Figure A5 : an example of date definition

7.1.2 <header></header>

This section provides generic information:

- **<plantID>** an alpha-numeric identifier, build with international phone code of the country, ZIP code of town and an arbitrary number. Example : 33.11100.200 for a process situated in France (33, in Narbonne (11100).
- **<plantName> :** usual name
- **<address>** :
- **<actor>**: authorised actors (physical or software)
- **<lastUpdate>** : last modification of physical or administrative configuration
- **<startUp>** : date of start-up of the process.

7.1.3 <bioprocessInfo></ bioprocessInfo >

This mark-up is used to describe the physical architecture of a plant that is stored in the table of the database with the same name.
Each element is stored inside a <row> </row> mark-up, which is made by this elements:

```
<!ELEMENT bioprocessInfo (numRow, row+)>

<!ELEMENT row (idVariable, idType, unit, min, max, nameEquipment, shortName)>
```

- **<idVariable>** : the id of the equipment in the plant.
- **<unit>** : unit of the device.
- **<min>** : min value.
- **<max>** : max value.
- **<nameEquipment>** : The current name of the device in the plant. It can be a sensor, an automate, an actuator,….
- **<shortName>** : the short name of the device.

An example is illustrated in Figure A6.

```
<bioprocessInfo>
<row>
        <idVariable>S0</idVariable>
        <idType>V2</idType>
        <unit>°C</unit>
        <min>0</min>
        <max>0</max>
        <nameEquipment>T Input</nameEquipment>
        <shortName>T Inp</shortName>
</row>
<row>
        <idVariable>S1</idVariable>
        <idType>V10</idType>
        <unit/>
        <min>0</min>
        <max>0</max>
        <nameEquipment>ARM</nameEquipment>
        <shortName>ARM</shortName>
</row>
</bioprocessInfo>
```

Figure A6 : an example of bioprocessInfo definition

This informations are always sent with the plantMLUpload.

7.1.4 <acq></acq>

This mark-up is used to describe any information characterizing the dynamical state of the process. All the measure done are stored in this section of the plantMLUpload.

Each <acq> mark-up represents a single acquisition linked to the process, and all the data are wrapped inside this tag (Figure A7).

```
<!ELEMENT acq (date, dev)>
<!ELEMENT dev (devType, met?, devName, chan)>
<!ELEMENT devName (#PCDATA)>
<!ELEMENT devType (#PCDATA)>
<!ELEMENT chan (lbl, unit, val)>
<!ELEMENT lbl (#PCDATA)>
<!ELEMENT unit (#PCDATA)>
<!ELEMENT val (#PCDATA)>
<!ELEMENT met (#PCDATA)>
```

Figure A7: Definition of acquisition

Example of the <acq> of probes (Figure A8):

```
<acq>
        <date>2005-04-27 12:55:20</date>
        <dev>
            <devType>SENS</devType>
            <devName>S2</devName>
            <chan>
                <lbl>PH Input</lbl>
                <unit>uPh</unit>
                <val>3.499000</val>
            </chan>
        </dev>
</acq>
<acq>
        <date>2005-04-27 12:55:20</date>
        <dev>
            <devType>SENS</devType>
            <devName>S4</devName>
            <chan>
                <lbl>Ph Low</lbl>
                <unit>uPh</unit>
                <val>48.820400</val>
            </chan>
        </dev>
</acq>
```

Figure A8: On-line sensors

Example of the <acq> of probes (Figure A9):

```
<acq>
        <date>2005-04-27 12:55:20</date>
        <dev>
            <devType>OFFLINE</devType>
            <devName>O4</devName>
            <met>Metodo1</met>
            <chan>
                <lbl>Ph Low</lbl>
                <unit>uPh</unit>
                <val>48.820400</val>
            </chan>
```

```
        </dev>
</acq>
```

Figure A9: Example of an off-line measure

7.1.5 <action></action>

This mark-up is used:

- to activate and tune a controller.
- to send commands to actuators.
- to change the parameters of a controller(PID).

Tuning of a controller (Figure A10)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plantML SYSTEM "D:\PMLFull.dtd">
<plantMLUpload>
    <action>
        <numRow>1</numRow>
        <row>
            <nameObject>controllore Matlab</nameObject>
            <nameEquipment>pompa 1</nameEquipment>
            <date>2005-05-04 18:05:49</date>
            <startDate>2005-05-04 18:05:49</startDate>
            <stopDate>2005-05-04 18:05:49</stopDate>
            <actionType>PARAM</actionType>
            <comment>PID</comment>
            <value>12@13@56</value>
            <status>off</status>
        </row>
    </action>
</plantMLUpload>
```

Figure A10: tuning a controller

Sending commands to actuators (Figure A11)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plantML SYSTEM "D:\PMLFull.dtd">
< plantMLUpload >
    <action>
        <numRow>1</numRow>
        <row>
            <origin>
                <nameActor>Andrea Quintabà</nameActor>
                <namePartner/>
                <address>Broganelli</address>
                <email>a@b.it</email>
                <phone>12345678</phone>
                <role>administrator</role>
                <priorityMax>5</priorityMax>
            </origin>
            <nameEquipment>Pompa di Carico</nameEquipment>
            <date>2005-05-03 10:59:38</date>
            <startDate/>
            <stopDate/>
            <actionType>COMMAND</actionType>
            <comment/>
            <value/>
            <status>on</status>
        </row>
    </action>
```

```
</ plantMLUpload >
```

Figure A11: Actions on the plant

Change a setpoint of a controller (Figure A12)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plantML SYSTEM "D:\PMLFull.dtd">
<plantMLUpload>
    <action>
        <numRow>1</numRow>
        <row>
            <origin>
                <nameActor>Andrea Quintabà</nameActor>
                <namePartner/>
                <address>Broganelli</address>
                <email>a@b.it</email>
                <phone>12345678</phone>
                <role>administrator</role>
                <priorityMax>5</priorityMax>
            </origin>
            <nameEquipment>Pompa di Carico</nameEquipment>
            <date>2005-05-03 10:59:38</date>
            <startDate/>
            <stopDate/>
            <actionType>SETPOINT</actionType>
            <comment/>
            <value>100</value>
            <status>on</status>
        </row>
    </action>
</plantMLUpload>
```

Figure A12: change a setpoint on the plant

7.1.6 <event></event>

This mark-up (Figure A13) is used:

- to transfer the events.
- to signalize maintenance to do from remote operator to the local plant.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plantML SYSTEM "D:\PMLFull.dtd">
<plantMLUpload>
<maintenance>
        <numRow>1</numRow>
        <row>
            <origin>
                <nameActor>Andrea Quintabà</nameActor>
                <namePartner/>
                <address>Broganelli</address>
                <email>a@b.it</email>
                <phone>12345678</phone>
                <role>administrator</role>
                <priorityMax>5</priorityMax>
            </origin>
            <nameClassEvent>Pulizia</nameClassEvent>
            <nameEquipment>Pompa 1</nameEquipment>
            <date>2005-05-05 10:05:00</date>
            <startDate>2005-05-05 10:05:00</startDate>
            <stopDate>2005-05-05 10:05:00</stopDate>
            <symptoms/>
            <solutions/>
            <consequences/>
            <comment/>
            <startEvent>2005-05-05 10:05:00</startEvent>
```

```
            <stopEvent/>
            <executedEvent/>
        </row>
    </maintenance>
</plantMLUpload>
```

Figure A13: Maintenance operations

8 APPENDIX B: Document Type Definition (DTD)

8.1 Upload

```
<?xml version="1.0" encoding="UTF-8"?>
<!--DTD generated by XMLSPY v5 rel. 4 U (http://www.xmlspy.com)-->
<!ELEMENT acq (date, dev)>
<!ELEMENT action (numRow, row+)>
<!ELEMENT actionType (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT bioprocessInfo (numRow, row+)>
<!ELEMENT chan (lbl, unit, val)>
<!ELEMENT comment (#PCDATA)>
<!ELEMENT consequences EMPTY>
<!ELEMENT date (#PCDATA)>
<!ELEMENT dateFormat (#PCDATA)>
<!ELEMENT dateGMT (#PCDATA)>
<!ELEMENT destID (#PCDATA)>
<!ELEMENT dev (devType, met?, devName, chan)>
<!ELEMENT devName (#PCDATA)>
<!ELEMENT devType (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT executedEvent (#PCDATA)>
<!ELEMENT header (protocolVersion, sourceID, sourceVersion, destID, dateFormat,
dateGMT, plantID, plantName, plantPurpose, influentType, plantLayout,
plantTechnology)>
<!ELEMENT idType (#PCDATA)>
<!ELEMENT idVariable (#PCDATA)>
<!ELEMENT influentType (#PCDATA)>
<!ELEMENT lbl (#PCDATA)>
<!ELEMENT maintenace (numRow, row+)>
<!ELEMENT max (#PCDATA)>
<!ELEMENT met (#PCDATA)>
<!ELEMENT min (#PCDATA)>
<!ELEMENT nameActor (#PCDATA)>
<!ELEMENT nameClassEvent (#PCDATA)>
<!ELEMENT nameEquipment (#PCDATA)>
<!ELEMENT nameObject (#PCDATA)>
<!ELEMENT namePartner (#PCDATA)>
<!ELEMENT numAcquisition (#PCDATA)>
<!ELEMENT numRow (#PCDATA)>
<!ELEMENT origin (nameActor, namePartner, address, email, phone, role,
priorityMax)>
<!ELEMENT phone (#PCDATA)>
<!ELEMENT plantID (#PCDATA)>
<!ELEMENT plantLayout (#PCDATA)>
<!--                          root element                                 -->
<!ELEMENT plantMLUpload (header, bioprocessInfo, numAcquisition?, acq*,
maintenance?, action?)>
<!ELEMENT plantName (#PCDATA)>
<!ELEMENT plantPurpose (#PCDATA)>
<!ELEMENT plantTechnology (#PCDATA)>
<!ELEMENT priorityMax (#PCDATA)>
<!ELEMENT processed (#PCDATA)>
<!ELEMENT protocolVersion (#PCDATA)>
<!ELEMENT role (#PCDATA)>
<!ELEMENT row (idVariable?, idType?, unit?, min?, max?, origin?,
nameClassEvent?, nameObject?, nameEquipment, shortName?, date?, startDate?,
```

```
stopDate?, symptoms?, solutions?, consequences?, actionType?, comment?,
startEvent?, stopEvent?, executedEvent?, value?, status?, processed?)>
<!ELEMENT shortName (#PCDATA)>
<!ELEMENT solutions EMPTY>
<!ELEMENT sourceID (#PCDATA)>
<!ELEMENT sourceVersion (#PCDATA)>
<!ELEMENT startDate (#PCDATA)>
<!ELEMENT startEvent (#PCDATA)>
<!ELEMENT status (#PCDATA)>
<!ELEMENT stopDate (#PCDATA)>
<!ELEMENT stopEvent (#PCDATA)>
<!ELEMENT symptoms EMPTY>
<!ELEMENT unit (#PCDATA)>
<!ELEMENT val (#PCDATA)>
<!ELEMENT value (#PCDATA)>
```

8.2 Download

```
<?xml version="1.0" encoding="UTF-8"?>
<!--DTD generated by XMLSPY v5 rel. 4 U (http://www.xmlspy.com)-->
<!ELEMENT action (numRow, row+)>
<!ELEMENT actionType (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT comment (#PCDATA)>
<!ELEMENT consequences EMPTY>
<!ELEMENT date (#PCDATA)>
<!ELEMENT dateFormat (#PCDATA)>
<!ELEMENT dateGMT (#PCDATA)>
<!ELEMENT destID (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT executedEvent (#PCDATA)>
<!ELEMENT header (protocolVersion, sourceID, sourceVersion, destID, dateFormat,
dateGMT, plantID, plantName, plantPurpose, influentType, plantLayout,
plantTechnology,serverResp)>
<!ELEMENT influentType (#PCDATA)>
<!ELEMENT serverResp (#PCDATA)>
<!ELEMENT maintenance (numRow, row+)>
<!ELEMENT nameActor (#PCDATA)>
<!ELEMENT nameClassEvent (#PCDATA)>
<!ELEMENT nameEquipment (#PCDATA)>
<!ELEMENT nameObject (#PCDATA)>
<!ELEMENT namePartner (#PCDATA)>
<!ELEMENT numRow (#PCDATA)>
<!ELEMENT origin (nameActor, namePartner, address, email, phone, role,
priorityMax)>
<!ELEMENT phone (#PCDATA)>
<!ELEMENT plantID (#PCDATA)>
<!ELEMENT plantLayout (#PCDATA)>
<!ELEMENT plantMLDownload (header, maintenance?, action?)>
<!ELEMENT plantName (#PCDATA)>
<!ELEMENT plantPurpose (#PCDATA)>
<!ELEMENT plantTechnology (#PCDATA)>
<!ELEMENT priorityMax (#PCDATA)>
<!ELEMENT processed (#PCDATA)>
<!ELEMENT protocolVersion (#PCDATA)>
<!ELEMENT role (#PCDATA)>
<!ELEMENT row (origin?, nameClassEvent?, nameObject?, nameEquipment, shortName?,
date?, startDate?, stopDate?, symptoms?, solutions?, consequences?, actionType?,
comment?, startEvent?, stopEvent?, executedEvent?, value?, status?, processed?)>
```

```
<!ELEMENT shortName (#PCDATA)>
<!ELEMENT solutions EMPTY>
<!ELEMENT sourceID (#PCDATA)>
<!ELEMENT sourceVersion (#PCDATA)>
<!ELEMENT startDate (#PCDATA)>
<!ELEMENT startEvent (#PCDATA)>
<!ELEMENT status (#PCDATA)>
<!ELEMENT stopDate (#PCDATA)>
<!ELEMENT stopEvent (#PCDATA)>
<!ELEMENT symptoms EMPTY>
<!ELEMENT value (#PCDATA)>
```

9 APPENDIX C: Database structure

Introduction

The database structure (**Figure C1**) is a revised and improved version of the Telemac database structure which has been tested from SPES for a sufficiently long period.

The experiences conducted have revealed some flaws concerning the effective manageability of the possibly huge amount of data and its maintenance across the network of controlled plants. The flaws were mainly regarding some technical aspects of efficient implementation and utilization of the database, though only from a very practical and product-oriented point of view.

Indeed the original database structure proposed has been found to be a very good one under theoretical arguments, but with quite awkward implementation on low cost mainframe architectures.

Bearing in mind that, low-cost and efficiency is our constant main goal, here we propose some (almost everywhere minor) modifications of the original Telemac database structure.

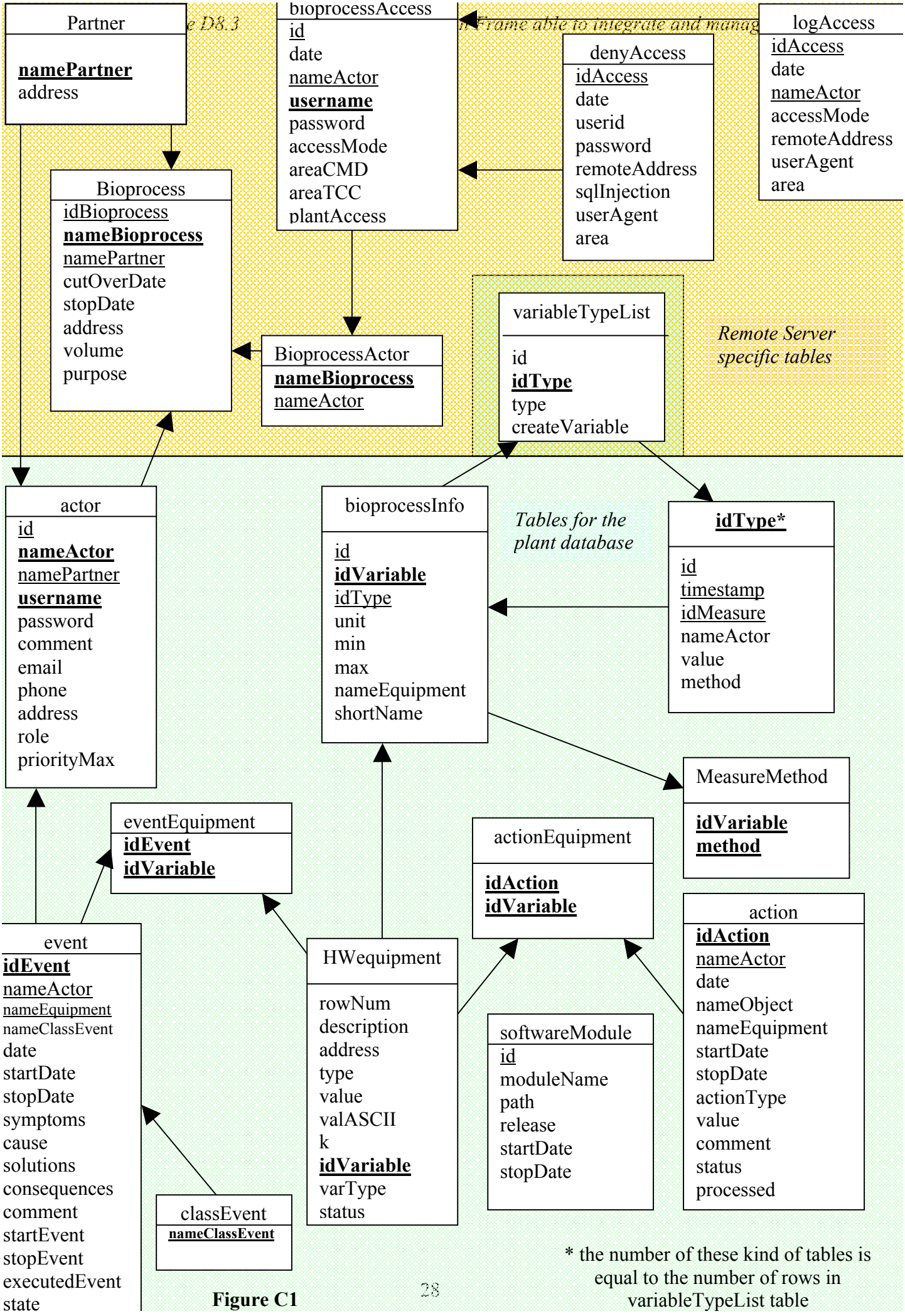
Partner
namePartner
address
bioprocessAccess
id
date
nameActor
username
password
accessMode
areaCMD
areaTCC
plantAccess
logAccess
idAccess
date
nameActor
accessMode
remoteAddress
userAgent
area
denyAccess
idAccess
date
userid
password
remoteAddress
sqlInjection
userAgent
area
Bioprocess
idBioprocess
nameBioprocess
namePartner
cutOverDate
stopDate
address
volume
purpose
BioprocessActor
nameBioprocess
nameActor
variableTypeList
id
idType
type
createVariable
Remote Server specific tables
actor
id
nameActor
namePartner
username
password
comment
email
phone
address
role
priorityMax
bioprocessInfo
id
idVariable
idType
unit
min
max
nameEquipment
shortName
Tables for the plant database
idType*
id
timestamp
idMeasure
nameActor
value
method
MeasureMethod
idVariable
method
eventEquipment
idEvent
idVariable
actionEquipment
idAction
idVariable
action
idAction
nameActor
date
nameObject
nameEquipment
startDate
stopDate
actionType
value
comment
status
processed
event
idEvent
nameActor
nameEquipment
nameClassEvent
date
startDate
stopDate
symptoms
cause
solutions
consequences
comment
startEvent
stopEvent
executedEvent
state
HWequipment
rowNum
description
address
type
value
valASCII
k
idVariable
varType
status
softwareModule
id
moduleName
path
release
startDate
stopDate
classEvent
nameClassEvent
* the number of these kind of tables is equal to the number of rows in variableTypeList table

Figure C1

Database structure overview

A key issue for the comprehension of the database structure is the exact definition we now give to the term "variable".

A **variable** here is intended as "a quantity that can assume any set of values" (and not intended as "the symbol associated to a variable quantity").

The tag which is associated to a variable (i.e. the symbol) and used to refer to it will be composed of two parts: the name and the method. Moreover, to refer to a certain variable tag, which may be quite long, is has been provided the room for associating a shorthand tag to the long variable tag. It will come in handy for example for variable reference in graphs, control panels, synoptic graphs or whatever needs a short label to be displayed.

To a variable it is associated also a "type" and possibly a measurement unit and a scaling constant. With **type** is intended the measure or quality of the variable. For example, a variable can be obtained by a measurement of pH, so the type will be "pH". Other types may be temperature, frequency, pressure etc.

But there can be types which are not directly related to a physical measurement or quantity, for example the status of a finite state machine or a switch device or whatever concerns with actuator devices in general. Indeed, a variable can be also the status of an electrovalve or a compressor which are very common devices in bioprocess control plants. In this case the type has to be intended as a general quality associated to a variable.

Then it can be provided a measurement unit to a variable and a scaling constant which can be used to post-process acquired data before their insertion in the database. That constant is used, for example, when the acquired data is digital and ranges between 0-255 while the physical quantity ranges between 0 and 1 in the actual measurement unit. So it can be viewed also as a minor signal conditioning capability.

Note that a type as "pH" can be at same time the measurement unit as well. So it must be paid attention to misleading definitions of types which overlaps with the names of measurement units.

The remote server database structure is showed in **Figure C1**. It composed by the tables depicted in the two different coloured main areas plus a different coloured small area containing only one table.

A remote server is the main node of a network of plant nodes, connected by internet protocol.

The tables which appear in the orange area are present only on the remote server database. The tables in the blue area are in common with each of the databases (i.e. local databases) on the network plants. So we can consider this part of the database as the shared one from the remote

server point of view. Actually the remote server database will contain a copy of those local databases, one copy for each plant, and they are going to be synchronized remotely by means of internet connections.

The only exception is the **variableTypeList** table. It can be considered a real shared table as it exists unique on remote server database and can only be changed and written by the remote server (i.e. by a remote expert operator which has authorization to full access to the remote server database).This restriction has been conceived to let the network have a variable type standardization of its own. When a change of this table is made it has to be dispatched to all the plants of the network.

This kind of databases separation is related to the different purposes of the local and remote software: the remote one must manage all the plants of the network, while the local software must supervise only its own plant process. So, having copies of the local databases on the remote server avoids the need to conceive a structure translation scheme for the local data uploaded for their storage in the remote central database. Intermediate software parsers are not required from the communication between local and remote databases.

Now, some other useful information about **Figure C1** need to be added.

The field's name that is represented in a bold underlined font means that in the current table it is a Primary Key (index and unique).

The field's name that is represented in underlined font means that in the current table it is an index.

The database structure is compliant with referential integrity requirements, typical of the DBMS (Database Management System). That is, if a modification affects a field that contained referential keys, this modification should be imperatively manually propagated in all the tables where this field appeared, otherwise the database becomes inconsistent.

Database table description

Partner
namePartner (varchar)(30)(pk) : name of the partner company registered in the network
address (text) : address of the company

Partner table is used to register the partners which are involved in the controlled plant network.

Bioprocess
idBioprocess int (index) : identification index of the plant bioprocess
nameBioprocess (varchar)(30)(pk) : identification name of the plant bioprocess
namePartner (varchar)(30)(index) : name of the partner company registered in the network
startDate (dateTime) : start date of joining the plant network
stopDate (dateTime) : stop date of joining the plant network
address (text) : address of the plant
volume (float) : information on plant size or production volumes
purpose (text) : kind of production

Bioprocess this table retains the list of the plants in the network.

bioprocessActor
nameBioprocess (varchar)(30)(pk) : identification name of the plant bioprocess
nameActor (varchar)(60)(pk) : identification name for a plant operator or user

bioprocessActor table traces the relationships between the following **actor** table and the **bioprocess** table.

actor
Id (int**)** (autoincrement): identification index of the Actor
nameActor (varchar)(60)(pk) : identification name for a plant operator or user
namePartner (varchar)(30)(index) : name of the partner company registered in the network
username (varchar)(30)(pk) : username of the operator or user for software access
password (varchar)(50) : password
comment (text) : a slot for added information
email (varchar)(30) : e-mail address
phone (varchar)(30) : phone address
address (text)(30) : address
role (text) : charge or role in the company
priorityMax (int) : a number used for the priority level of actions 1-5

actor table holds the profile of the plant operator or user with related information for access to local software.

variableTypeList
id int (index) : identification number of variable type
idType (varchar)(5)(pk) : identification tag of variable type
type (text) : name of variable type
createTable(set('Y','N')): establish if must be create a data table for this type.

variableTypeList table lists all the variable types that can be measured in all the plants of the network.

The **idType** field is assigned in automatic and unique way by the remote expert and it is used for the referential relation in the database structure for each single plant.

The **createTable** establishes if the type owns a data table or not.

Example:

id	**idType**	**type**	**createTable**
1	V1	PH	Y
2	V2	H2	Y
3	V3	CO2	Y
4	V4	Temperature	Y
5	V6	Inverter	N

This table is the same for all the plants of the project, and after an initial filling done in the software configuration, it can be updated through an XML file by the remote administrator (where the internet connection is available otherwise with a manual update).

bioprocessAccess
id int index : index of the table and identification for remote access
date (datetime) : date and time of the remote access to central database
nameActor (varchar)(60)(index) : name of the accessing person
username (varchar)(30)(pk) . username
password (varchar)(50) : password
accessMode (enum('user,'administration')) : authorization level group profile
areaCMD (enum('allow','deny')) : remote action command on plant authorization
areaTCC (enum('allow','deny')) : remote event scheduling on plant authorization
.
.
.
area???
plantAccess (text) : list of names of the plants to which the access is allowed (;name1;name2;...)

bioprocessAccess table manages the remote access to the network and plant database, stating which plants the user is allowed to enter in.

In the **plantAccess** field, the allowed plants are registered with this structure:

;idBioprocess1;idBioprocess2;……

The **areaXXX** fields are used to define the command or action areas allowed for the current user.

Example:

id	**date**	**Name Actor**	**username**	**password**	**access Mode**	**area CMD**	**area TCC**	**area Pan**	**Plant Access**
1	2004-04-09 16:32:21	Paolo Ratini	pratini	2ffe4e77325d9a7152f7086ea7aa5114	user	deny	deny	allow	;plant1;plant2;plant3;

denyAccess
idAccess int (index) : identification number of failed access
Date (datetime) : date and time of failed access
Userid (varchar)(30) : name used for accessing as user name
Password (varchar)(30) : password used for accessing
remoteAddress (varchar)(100) : address of the remote client who tried to connect
sqlInjection (enum('yes,'no')) : whether an hacking technique was used
userAgent (varchar)(100) : name of the browser used for the connection
area (text) : area tried to access

denyAccess table registers the attempts of accessing that were failed.

logAccess
idAccess int (index) : identification number of access
Date (datetime) : date and time of access
nameActor (varchar(60)) : name of operator or user
remoteAddress (varchar)(100) : address of the remote client who connected
userAgent (varchar)(100) : name of the browser used for the connection
area (text): area of access

logAccess table registers the accesses which succeeded.

bioprocessInfo
id int(index) : identification index for a variable of the plant
idVariable (varchar)(5)(pk) : identification tag for a variable on the plant
idType (varchar)(5)(index) : type name of the variable
unit (varchar)(10) : measurement unit of the variable
min (float) : min value allowed
max (float) : max value allowed
sampleTime int(11): a specific sample time for the probe.
nameEquipment (varchar)(30)(unique) : name of the device used for measuring or the first name

in general **shortName** (varchar)(30) : shorthand name

bioprocessInfo table stores information on the plant variables. So it contains the actual configuration of the process with its online (measures from a device/probe) and offline variables (manual inserted measurements from a local operator).

Example:

id	idVariable	idType	unit	min	max	sampleTime	nameEquipment	shortName
1	S1	V1	upH	0	14	60	MyName	MyShortName
2	S2	V4	°C	0	100	30	MyName	MyShorName
3	O1	V1	Mg/l	0	50	Null	MyName	MyShorName
4	O2	V2	°C	0	100	Null	MyName	MyShorName

The **idVariable** field is the primary key that links this table with the other ones which are needed to define uniquely a variable. The **idType** field is the joint between this and the **variableListType** table. The **sampleTime** field establishes a specific sample time for the probe, and its value is at least equal to the main sampleTime of the process. So if the plant has a general sample time of 30 seconds, a probe with a sampleTime equal to 60, will have a data measure every two sample. If the field is null, the main sample will be used. Only multiple value will be accepted for this specific time.

The nameEquipment and shortName fields constitute the name representations that the process operator has assigned to a variable and they can be modified at any moment: this procedure avoids the introduction of a variable name in different languages because now the operator chooses a familiar name for the variable that he would acquire; in addition the offline measure can be inserted without any care about the previous existing set of variables. At any time a new one can be added with no difficulties, because they are created on the fly as the operator adds them to local database.

Note that in order to obtain the complete tag of the variable we need also information about the method. This information resides in the table **measureMethod**.

measureMethod
idVariable (varchar)(5)(pk) : identification tag for a variable on the plant
method varchar(30)(pk) : measurement methodology or technique name

The measureMethod table registers the relation between the method and the variable of the process. The relational use of bioprocessInfo and measureMethod identifies uniquely every variable of the plant.

Example: S1 is associated with “pH high” as equipment name and “online” as the method. Now the operator can introduce S2 with “pH low” as equipment name and “online” as the method, or S3 with “pH low” and “automatic” method. But he cannot use the same equipment name and method.

Note: it is the database itself which signals the error, whatever the software used to access it! This avoids error checkings at higher level.

idType*
id int(index) : identification index
Timestamp (datetime)(index) : date and time of acquisition
idVariable (varchar)(5)(index) : identification tag for a variable on the plant
value (float) : numerical quantity
method (varchar)(30) : measurement methodology or technique name

The **idType** tables are numerous as the **variableTypeList** states. The type identifier is the name for each table.

On the plant database there is a number of these tables equal to the different types of variables that in the plant are actually measured.

All the values for all the pH-type variables will be stored in the table named (for example) V1, because this is the primary key of the **variableListType** for the type “pH”.

Example 1: **Table V1 data**

id	**timestamp**	**nameActor**	**idVariable**	**value**	**method**
1	2004-12-22 00:00:01	Localsoftware	S1	7.0	online
2	2004-12-22 00:00:01	Paolo Ratini	O1	6.2	Method1

Example 2, if I have a process in which I can do measurements of pH, the type of the variable will be "pH", while the possibly variable names can be 'High pH' with method 'online' or 'low pH' with method 'method1' , but they both belong to the type "pH" and so their data will be stored in the same table V1.

Finally we can say that all the measurements of the variables in **bioprocessInfo** with the same **idType** are stored in the same table which name is the idType itself.

HWequipment
RowNum (int(index)) : identification index of hardware slot
Description (varchar)(30) : field for a brief description of hardware slot
Address (varchar)(12) : a field used only some type of hardware (reserved9
Type (int) : type of data acquired : 1-int 1 bytes;2-int 2 bytes; 3_float; 4-int 4 bytes
idVariable (varchar)(5)(PK): identification tag for a variable on the plant
value (varchar)(10): exadecimal value acquired
valASCII (varchar)(10) : string containing the number representing the value
k (float) : scaling constant
note (text): room for notes
varType(int): 1 for actuators – 0 for probes
status (set('Active','NotActive','Removed')): status of the variable and/or device

The **HWequipment** table is made up with a number of rows equal to the maximum number of slaves that can be enrolled by the hardware architecture that sends to the local software the data and through which pass the actions toward the plant.

The motivation for this table is to keep trace of the communication between the masterboard and the local software and for this reason its size remains established once the hardware stuff has been configured. So the order of the rows can be relevant or not depending on the hardware.

The software will assign to each row a primary key (**idMeasure** field) that establishes the relation between this table and the other ones.

If a device or probe is removed/stopped from the plant for some time, the corresponding row remains in the table but it is flagged with a status flag in the **status** field. The possible values for this field are:

1. Active
2. NotActive
3. Empty
4. Removed

The empty condition is set when the hardware is installed for the first time and the row must be assigned to a probe.

Example:

Row Num	description	address	type	Id Variable	value	Val ASCII	k	Var Type	status
1	Pump 1	0203020100000000	1	S1	72E53A406C5A2904	8.5219	1	1	Active
2	PH Input	0201010100000000	1	S2	5B3D3B407E5704FF	27.2397	1	0	Active

The introduction of a record in this table assumes the addition of a new record also in the **bioprocessInfo** table, which realizes an instant photo of all the variables measured in the plant both online (measure from the probe) and offline measures(measure from an operator of the process) and in the **measureMethod** table, that stores all the methods through which the variables are measured.

softwareModule
id (int) : identification number of the module
moduleName (varchar)(30) : extended name of the module
path (varchar)(200) : path to the .exe or .dll file.
release (varchar)(15) : release identifier
startDate (datetime) : date and time of enrolment
stopDate (datetime) : date and time of module shut down

The softwareModule table registers the software modules eventually enrolled by the local software.

Event
idEvent int(PK) : identification number of the event
nameActor (varchar)(60)(index) : name of operator or user
nameEquipment (varchar)(30) : name of the device used for action
nameClassEvent (varchar)(30) : name of the event class
date (datetime) : date and time of event insertion
startDate (datetime) : date and time of the start of scheduled event
stopDate (datetime) : date and time of the stop of scheduled event
symptoms (text) : symptoms that cause an alarm
cause (text) : cause of the alarm
solutions (text) : solution to problem
consequences (text) : consequences on the process
comment (text) : room for comments
startEvent(datetime**) :** beginning date of the event
stopEvent(datetime) : stop/pause date of the event
ExecutedEvent(datetime) : date of execution of the event
state (enum('Y','N')): state of the event

The **Event** table registers the events of the bioprocess. The state field establishes if the event must be sent to the local or remote database.

The **startEvent**,**stopEvent** and **ExecutedEvent** trace the history of the maintenance, that is when the event is started, or stopped for short time and finally finished. The **startDate** and **stopDate** fields are used for the events sent by the remote user, as a suggested time for the maintenance.

eventEquipment
idEvent int (pk) : identification number of the event
idVariable (varchar)(5) (pk): identification tag for a variable on the plant

The **eventEquipment** table registers the relation between the HWequipment and event tables.

Action
idAction int(pk) : identification number of action
nameActor (varchar)(60)(index) : name of operator or user
date (datetime) : date and time of event insertion

nameObject (varchar)(50) : name of the possibly controller
nameEquipment (varchar)(30) : name of the device used for action
startDate (datetime) : date and time of the start of scheduled action
stopDate (datetime) : date and time of the stop of scheduled action
actionType (enum('SETPOINT','PARAM','OUTPUT','COMMAND','INPUT')) : type of action
value (text) : value to send
comment (text) : room for comments
state (set('on','off') : state of the device
processed (set('Y','N')) : flag to indicate the actual execution of actions

The **Action** table registers the command and setpoints applied to the actuators. Some info about the fields:

1. **nameObject** : if the action comes from a Matlab controller, this field isn't null because It is filled with the name of the controller.
2. **nameEquipment** : the name of the actuators on which the action is applied. This field is the same of the **bioprocessInfo** table.
3. **actionType**: the kind of action to do(Setpoint,Param,Output,Command).
4. **status**: if the action is a command the field can be switched on or off.
5. **value**: in the case of a setpoint or similar action it is the value to apply. In the case of a param contains the formatted string with the params to apply:
   Ex **;P;I;D;**
6. **processed**: establishes if the action has been processed by the software.

actionEquipment
idAction int (pk) : identification number of action
idVariable (varchar)(5) (pk) : identification tag for a variable on the plant

The actionEquipment table registers the relation between the HWequipment and action tables.

10 APPENDIX D: ACCESSING DATABASE WITH MATLAB

The connection to the Mysql database server through Matlab is realized by the use of a dll (mysql.dll) that is provided by the development team of the same database server (http://www.mmf.utoronto.ca/resrchres/mysql/).

The connection in made with this instruction:

>> mysql('open','192.168.1.171','eoli','eoli')
Uptime: 20781 Threads: 2 Questions: 1963 Slow queries: 0 Opens: 53 Flush tables: 1 Open tables: 2 Queries per second avg: 0.094

After that we choose the database to use:
>> mysql('use','eoli')
Current database is "eoli"

Now we can do our queries on the database:
>> mysql('select * from bioprocessinfo')

id	idVariable	idType	unit	min	max	sampleTime	nameEquipment	shortName
2	S1	V10		0	0	0	ARM	ARM
3	S2	V1	uPh	0	0	60	PH Input	PH In
4	S3	V2	°C	0	0	60	T Low	T Low
5	S4	V1	uPh	0	0	60	Ph Low	Ph Lo
6	S5	V2	°C	0	100	60	T High	T Hi
7	S6	V2	°C	0	100	60	T Ambiente	T Amb
8	S7	V1	uPh	0	100	60	PH High	PH H
9	S8	V6	gas	0	1e+006	60	VGas	VGas
10	S9	V4	upp/123	0	12.12	60	H2	H2
11	S10	V5	CO2	0	0	60	Co2	Co2
12	S11	V3	flusso	0	0	60	Flussimetro 1	Flux1
13	S12	V3	flusso	0	0	60	Flussimentro2	Flux2
16	S13	V7		0	0	0	Pompa di Carico	Pompa
17	S14	V7		0	0	0	Pomap di Scarico	Pompa
18	S15	V7		0	0	0	Pompa Prelievo Fanghi	Pompa
19	S16	V7		0	0	0	Elettrovalvola 1	Elett
20	S17	V7		0	0	0	Elettrovalvola 2	Elett
21	S18	V11		0	0	0	Inverter1	Inv1

Finally we close the connection to the server:
>> mysql('close')

In these few lines of code we have shown as easy is the interaction with the Mysql server in Matlab.