High Performance Hydraulic Simulations on the Grid using Java and ProActive

Guilherme Peretti-Pezzi

UNIVERSITÉ DE NICE - SOPHIA ANTIPOLIS

ÉCOLE DOCTORALE STIC
SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

THÈSE

pour obtenir le titre de

Docteur en Sciences

de l'Université de Nice - Sophia Antipolis

Mention Informatique

présentée et soutenu par

Guilherme PERETTI PEZZI

HIGH PERFORMANCE HYDRAULIC SIMULATIONS ON THE GRID USING JAVA AND PROACTIVE

Thèse dirigée par Denis Caromel,

au sein de l'équipe OASIS,
*équipe commune de l'*INRIA Sophia Antipolis*, du* CNRS *et du laboratoire* I3S

soutenue le 15 décembre 2011

Jury:

Président du Jury	Philippe GOURBESVILLE	Université de Nice - Sophia Antipolis
Rapporteurs	Jocelyne ERHEL	INRIA Rennes
	Pierre-Olivier MALATERRE	CEMAGREF Montpellier
Co-rapporteur	Olivier PILLER	CEMAGREF Bordeaux
Examinateur	Yann VIALA	Société du Canal de Provence
Directeur de thèse	Denis CAROMEL	Université de Nice - Sophia Antipolis

Contents

List of Figures

List of Tables

1 Motivation and positioning

Contents

This chapter gives a background on water distribution systems, its evolution and the necessity of using computer assistance in order to design modern distribution infrastructures. Then, it is presented an overview of some of the main tools used for designing these networks, followed by an introduction of the Société du Canal de Provence, their main activities and role in this project. Finally, the main projects used in the context of this work are presented: the *ProActive Parallel Suite* (developed by the OASIS research group at INRIA Sophia Antipolis and ActiveEon) and the AGOS project (developed by INRIA, HP, Oracle, ActiveEon and Amadeus).

1.1 Water distribution systems

This section introduces the main types of infrastructure used nowadays for distributing water around the world: free surface distribution and pressured water distribution networks (*WDN*).

1.1.1 Free surface/open channel

Water distribution systems have historically played a key role for supplying water to places with scarce resources. These systems have been traditionally built mostly over open channels, since the roman aqueducts for example, but still today these structures are widely used all over the world and they are fundamental for the development of cities and of agriculture.

Figure 1.1: Example of free surface channel (Boutre, Var department, France).

The flow in open channels (Figure 1.1) * is moved by gravity and thus is limited by the land relief. In these structures it is difficult to measure precisely the consumption, making it also difficult other maintenance task such as optimizing resources usage and reducing the loss. Also, when used for irrigation it is often necessary to impose a rotation criteria in order to guarantee equal rights to all consumers. As consequence of this compromise, some consumers might not be able to irrigate their crops when needed and this will therefore cause productivity losses. In order to avoid this issue it is also possible to feed the channel with more water than the actual demand. This can guarantee availability but will cause higher loss levels.

Although open channels can be adapted for transport, alternative distribution methods had to be developed due to the limitations when dealing with water distribution. One alternative that has been used in order to overcome most of these issues is the adoption of *WDN*.

1.1.2 Water Distribution Networks

Water distribution network is a pressurized system where the water flows due differences in the pressure, water flows from the point with higher pressure towards to the lower one. Today the most common ways to obtain a high pressures are the use of strategically located tanks, which are usually fed by a pump, or by using directly a pump in order to increase the network pressure.

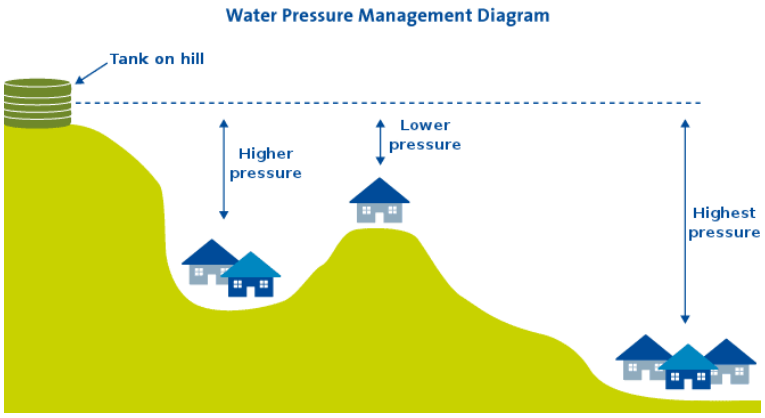

Figure 1.2: Diagram of a typical pressurized water network.

WDN have many advantages over traditional open channel systems. They are able to provide water on-demand, meaning that they can provide services with better quality and with a higher

*Photo credits: Camille Moirenc.

distribution efficiency. They can also overcome topographic constraints and therefore they can cover more area when compared to open channels. The mechanisms for measuring the water usage are more precise, they can also help to detect water leaks and this has both economic and environmental impacts.

One key issue especially when dealing with irrigation networks is the problem of predicting the number of simultaneous clients consuming the resources. When designing these networks the objective is to provide on-demand delivery schedule to all clients while minimizing the cost by optimizing the pipes diameters. The original approach developed by *SCP* for predicting client's demand is called *Clément*'s Formula [Clé66], more details can be found in Section 3.2.

WDN can also be difficult to be designed as they become bigger and more complex. For example, some *WDN* equipments can have dynamic behaviors depending on a monitored parameter such as pressure, flow direction or tank level. Their actions may affect other equipments and trigger other behavior changes. The possible scenarios can quickly become too complicated and therefore it is crucial to use an adapted tool when designing these networks.

1.2 Computer aided network system design

With the growth of the *WDN*, the engineers began to rely on computer simulation in order to design and optimize them. The use of computers allowed to design much more complex networks, calculating information that otherwise couldn't be obtained with the same level of precision. This happened many years ago when computers were very limited in terms of processing power, storage and availability of proper tools for writing software for performing complex calculations. For example, the *SCP* began developing a software for simulating meshed water networks in 1977 based on an existing code formerly created by the CEMAGREF [cem11].

This section presents some of the existing tools used for designing *WDN*, including the software developed by the *SCP* called IRMA.

1.2.1 Epanet

Epanet is a software for modeling *WDN* that was developed by the United States Environmental Protection Agency (EPA). Epanet is an open source tool, it first appeared in early 90's and nowadays it has become a standard reference for designing *WDN*.

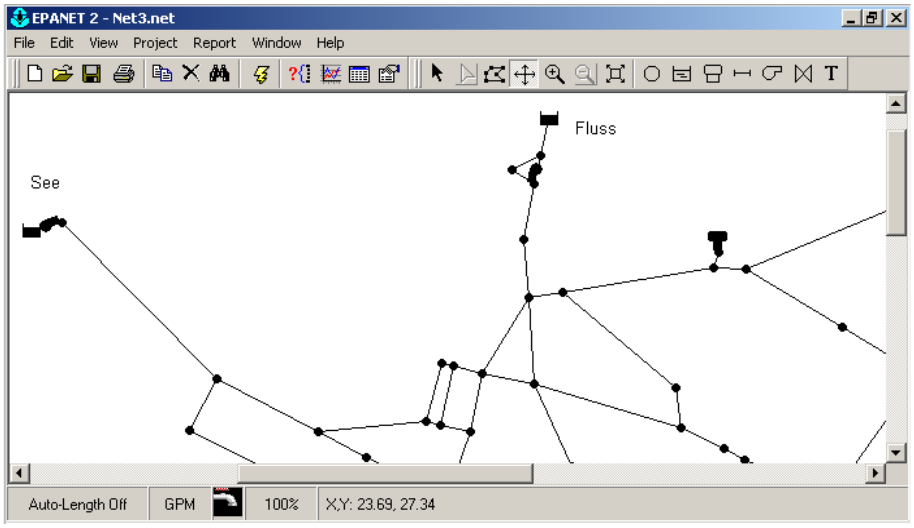

Figure 1.3: Screenshot of Epanet 2.

It models several types of structures, such as pumps (with constant or variable speed), valves

(standard, pressure/flow regulating or shutoff/security), tanks (with different shapes) and reservoirs (with infinite capacity). More details about these equipments can be found at Section 2.6. Epanet can perform steady state simulations and also simulation for extended periods of time.

Epanet provides an integrated environment for editing networks, running hydraulic and water quality simulations, and viewing the results. Figure 1.3 shows an example network being edited in Epanet. The icons for most commonly used operations can be found on the top menu bar: file handling, edition, results viewing, table input edition, scrolling and selection tools, and finally the icons for adding equipments. The network can be edited in the white pane below where pipe, junctions and equipments are represented.

1.2.2 MIKE URBAN

MIKE URBAN [mik11] is a commercial software solution developed by DHI [dhi11] for urban water modeling. Besides treating *WDN*, it covers also sewer and water drainage systems.

The underlying engine used by MIKE URBAN for simulating *WDN* is Epanet based and being a commercial solution it offers other interesting features, such as support for Geographical Information Systems (GIS), fire flow analysis, demand allocation and distribution. It also performs water quality analysis, for example water age, blending water from different sources, contamination propagation and growth of disinfection products.

1.2.3 Porteau

Porteau [por11] is freeware software for simulating *WDN* and it is developed by the CEMAGREF [cem11], which is a public research institute in France focusing on land management issues such as water resources and agricultural technology.

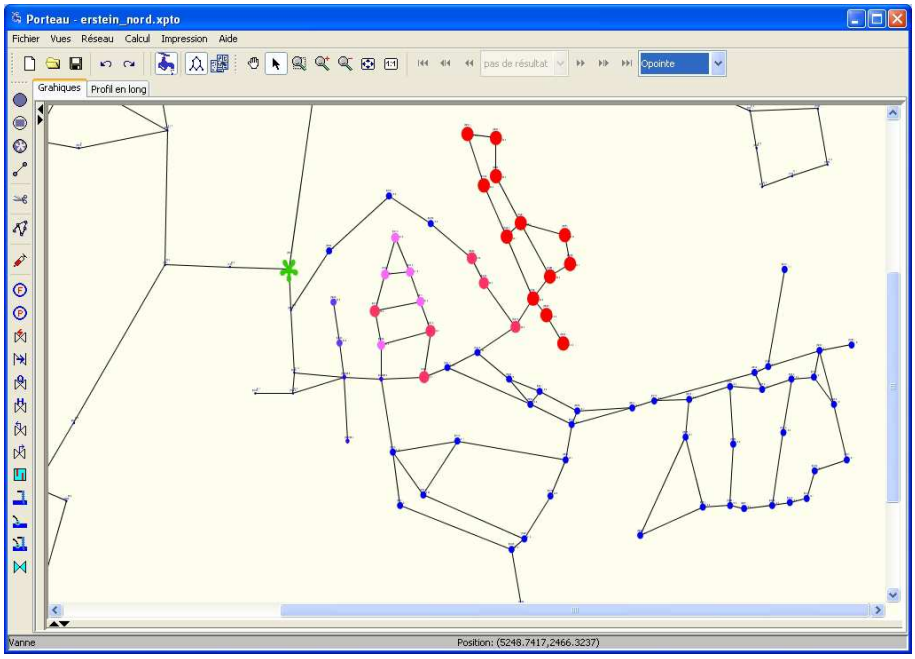

Figure 1.4: Screenshot of Porteau's Opointe module.

The first version of Porteau was written in FORTRAN released in 1977. In 2008 CEMAGREF released the version 3.0, it is written in Java with graphical interfaces for editing and visualizing the networks. Porteau consists in three main modules: Zomayet for performing simulations of extended periods of time, Opointe for steady state simulation, including a probabilistic method

for estimating the demand, and Qualité that allows to simulate the water quality in the network. Figure 1.4 shows a screenshot of a network being edited using the the Zomayet module.

1.2.4 Flowmaster

Flowmaster [flo11] is a commercial thermo fluid simulation software suite for the analysis of fluid mechanics within pipe networks. The module for water systems may be used to help engineers in many areas, for example it allows to perform several to predict water distribution to meet cyclic demands, optimize pump station sizing and energy consumption to minimize pumping costs, predict pressure surges to ensure safety and pipe integrity, size piping systems.

1.3 Société du Canal de Provence

Société du Canal de Provence (*SCP*) is a French company with more than 50 years of experience in designing, building and maintaining *WDN*. The *SCP* main activities are the water distribution in the french region "*Provence-Alpes-Côte d'Azur*" and providing assistance for designing and deploying infrastructures for water distribution. The infrastructure maintained by the *SCP* in France is mostly used for agricultural purposes, but it also serves other clients such as industries, potable water providers and local communities.

1.3.1 Infrastructure overview

SCP's most important activities are located in France, as it can be seen in Figure 1.5, where the most relevant channels are represented.

Figure 1.5: Overview of SCP's main distribution infrastructures in south of France.

Here are some key figures to summarize the managed infrastructure:

- 271 km of underground galleries and open channels
- 280 km of secondary/distribution channels
- 587 km of pipe with diameter $\geq$ 500 mm
- 5000 km of pipe with diameter < 500 mm

- 87 pumping and booster stations
- 16 drinking water plants and 3 treatment plants
- 4 reservoirs (> 1 million m^3)
- 12 reservoirs (between 10.000 and 1 million m^3)
- 70 tanks (between 500 and 10.000 m^3)

Figure 1.6: Bimont Dam, next to Aix-en-Provence.

All this infrastructure is used to supply raw water to more than 2 million inhabitants. The total amount of stored water is around 50 million of m^3 coming from the Verdon river and 10 million of m^3 coming from the Saint-Cassien lake. Figure 1.6 gives an example of one of the reservoirs, the Bimont dam, that is located in Saint-Marc-Jaumegarde next to Aix-en-Provence.

Since the concerned structure in this project are the pressured piped distribution networks, next section presents the software used by the *SCP* for designing and managing these networks.

1.3.2 IRMA

IRMA is a commercial software developed by the *SCP* for simulating meshed *WDN*. The initial program was based in a FORTRAN developed by the CEMAGREF and in 1977 the *SCP* created a branch in order to adapt the software for its own needs. CEMAGREF continued the development independently on his side to create the software that is now called Porteau.

The main reason for creating a separate branch was the need adapt IRMA in order to fit the *SCP*'s needs, while implementing the probabilistic resolution created and used by the *SCP* in order to estimate agricultural clients demand. This probabilistic method is called *"Débit de Clément"* and consists basically in performing separate resolutions for each class of offtake. More details can be found in Section 3.2 and [Clé66].

IRMA networks consist of pipes, nodes, tanks/reservoirs and equipments, such as pumps, valves (standard or shutoff/security) and pressure/flow regulators. The main simulation features are: steady state and extended period of time simulations, peak demand estimation by using

```
{  1er       2nd       No Nb pri.  COTE    COTE      CHARGE           No DIAMETRE             SERVICE    SERVICE
{ NOEUD      NOEUD     LOI cumul.    TN     PIEZO     SUPPL.   LONGUEUR DANS BORDEREAU          EN ROUTE   EN ROUTE
{   |          |       VAR. |(2)   au 1er   DESIREE   P.G.B.      (m)    (si besoin)             CONSTANT   VARIABLE
{   |          |         |  |     noeud(1)  au 1er    (mCE)       |             RUGOSITE   (l/s)       (l/s)
{   |          |    INDIC|  | No   (NGF)    noeud(1)   (1)        |       |  DIAMETRE     (mm)       |          |
{   |          |       |  |  |BOR.    |      (NGF)       |         |       |    (mm)       (2)        |          |    MATERIAU
{   |          |       |  |  |CAN.    |        |         |         |       |      |         |          |          |      (3)
{   |          |       |  |  | |      |        |         |         |       |      |         |          |          |       |
{+++++++++ ++++++++++---+++---++------.---++++++.+++------.---++++++.+++-----++++++.+++------.---++++++.+++------.--- --------
{ antenne No 41 :
{DESSOUS PRENDRE LE GT 267 (DETAIL DU GT 478)
9804413002 9804413003    1    2    146.60                      70.0    1     300.0      1.80                    ACIER
9804413055 9804413830    1    2    140.00                       0.6    2      75.0                              PVC
9804410830 9804413830    1    2    140.00                       2.0    3      64.0                              PVC
```

Figure 1.7: Excerpt from IRMA's text network file: pipe description section.

the *Clément*'s Formula, maximum pressure, pipe diameter optimization (Labye [REF]), water quality analysis, tank and pump design.

Although IRMA provides most of the simulation features required by the *SCP* users, it lacks graphical interfaces for editing network information and also does not allow any integration with data available in other data bases used by the *SCP* (such as GIS). Figure 1.7 show an excerpt from IRMA's network description file, it uses a standard text file where lines beginning with { are comments. These comments are used to help the user to write the values in the correct position, each field is precisely delimited and therefore values misplaced will be attributed to a neighbor field. Each pipe is described by one line containing the first and second node, length, diameter. Some fields are optional and other values, such as roughness, are taken from the line above if they are not explicitly informed.

Table 1.1: Time required for performing topology analysis of three IRMA networks.

Network ID	Pipes	Extension (km)	Offtakes	Time (min)
R095.05	4350	160	1573	10,12
R155.05	2806	189	1276	12,25
R093.05	7369	287	2693	61,3

With the growth of the *SCP* networks, IRMA is also facing performance limitations. More precisely, the algorithms for analyzing the topology and preparing the equations system became inefficient when dealing with complex networks. Table 1.1 presents the time required to analyze the topology of 3 networks along with some information regarding the size of the networks. They have from 2806 up to 7369 pipes and a total length of $287km$ in the most complex case. In these examples the topology analysis takes from 10 minutes to 1 hour, however there are cases that can take up to several days. As a workaround for this issue, IRMA caches the topology analysis results in temporary files so that they can be reused in future executions. However, this mechanism is efficient only if there aren't any changes in the network topology and thus changes in the topology are usually avoided or postponed in complex networks.

1.4 Project goals

IRMA's code base is becoming 30 years old, it consists in more than 40.000 lines of FORTRAN code that were originally written for being used with punch cards. The code has been re-engineered and updated to FORTRAN 90, but some deprecated software engineering techniques still remain. The hydraulic simulations performed by IRMA are complex by nature and the interactions among the equipments generate a big number of numeric exceptions. These exceptions that were discovered as new networks were designed during the 3 decades of IRMA's usage, the patches for fixing these cases are spread over different parts of the code and therefore it is very difficult to understand and to specify the behavior of a given equipment.

For all these reasons this version of IRMA is clearly becoming unsustainable in terms of maintainability. With the need of overcoming the performance issues and in order to improve user productivity providing modern graphical interfaces, the adoption of a new tool is unavoidable.

This new tool can be obtained by adapting one of the existing tools for designing *WDN*, what would mean to implement the *Clément*'s Formula, migrate and validate all the existing networks and projects to the new network format, and finally training all the users for using the new environment. Alternatively, the new tool could be rewritten from scratch based on IRMA's source, keeping the original model and, thus, minimizing the impact of replacing the tool. Assuming that costs would be higher if a commercial software is purchased and all the networks had to be migrated to a new format, the latter option was chosen as a solution.

1.5 Solution proposal overview

The language chosen for this new tool is Java, mainly for its portability, its ability to integrate easily with other systems deployed at SCP (such as Netview, the GIS developed by the *SCP*). Concerning performance, the use of Java can enable distributed execution on computing Grid infrastructures, using especially the *ProActive Parallel Suite* [pro11].

Although Java provides very interesting features for easing the development and integration with other systems, it's also a recent language compared to FORTRAN. The language itself is not yet consolidated for high performance scientific applications and therefore it lacks some numerical libraries that are available in FORTRAN and C/C++, like direct linear system solvers with sparse matrix storage. More details about this issue can be found in Sec. 5.2.

The following sections introduce the main projects used for enabling distributed and service oriented execution on the Grid: the *ProActive Parallel Suite* and the context of the AGOS Project.

1.5.1 ProActive Parallel suite

ProActive Parallel Suite [BBC+06] is an innovative Open Source solution (OW2 consortium) for parallel, distributed, multi-core computing. Developed by the OASIS team through a joint project between INRIA, CNRS, I3S and UNSA, *ProActive* has produced innovation (INPI patents) and new standards currently evaluated by the European standards organization (ETSI).

ProActive provides 3 main packages:

Programming: features a concurrent and parallel programming model, offers distributed and asynchronous communications, mobility and a deployment framework. It simplifies the programming of multithreaded, parallel, and distributed applications for Clouds, Grids, multi-cores, clusters, and data-centers.

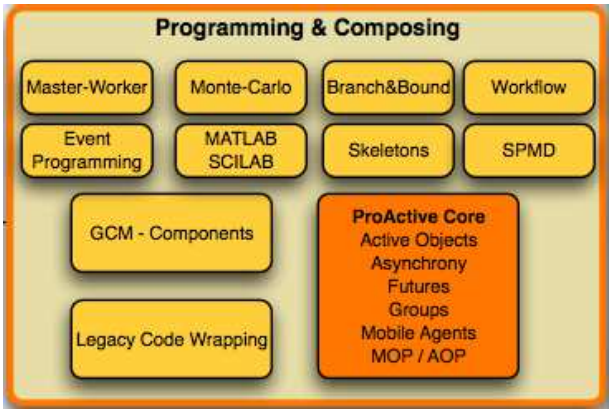

Figure 1.8: Main components of the *ProActive* Programming framework.

Scheduling: the *ProActive Scheduler* allows a multi-user sharing of resources to run workflows, *ProActive* applications, native or Java code on the Grid.

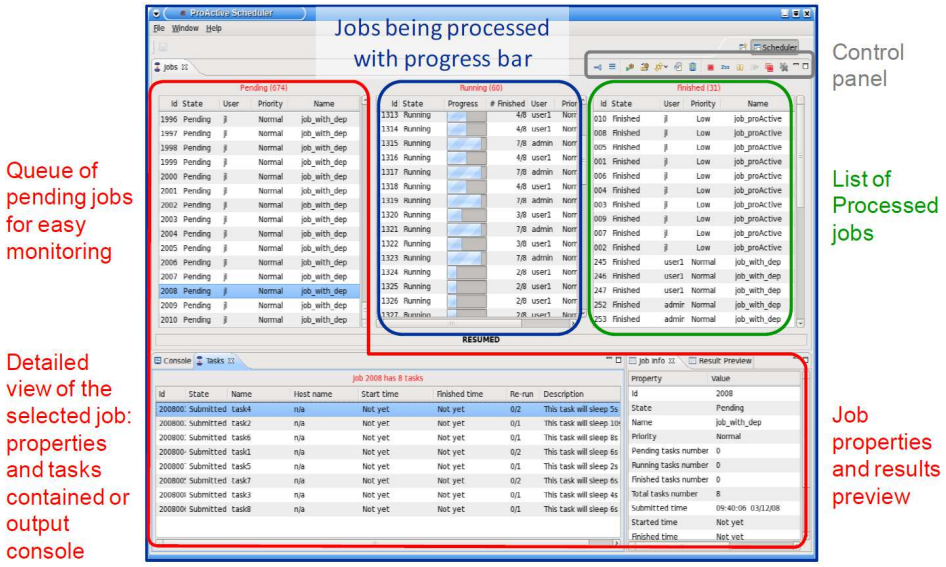

Figure 1.9: Screenshot of the *ProActive* Scheduling tool.

Resourcing: the *ProActive Resource Manager* allows to manage in real-time resources on the grid, users can monitor their activity, add or remove resources, etc...

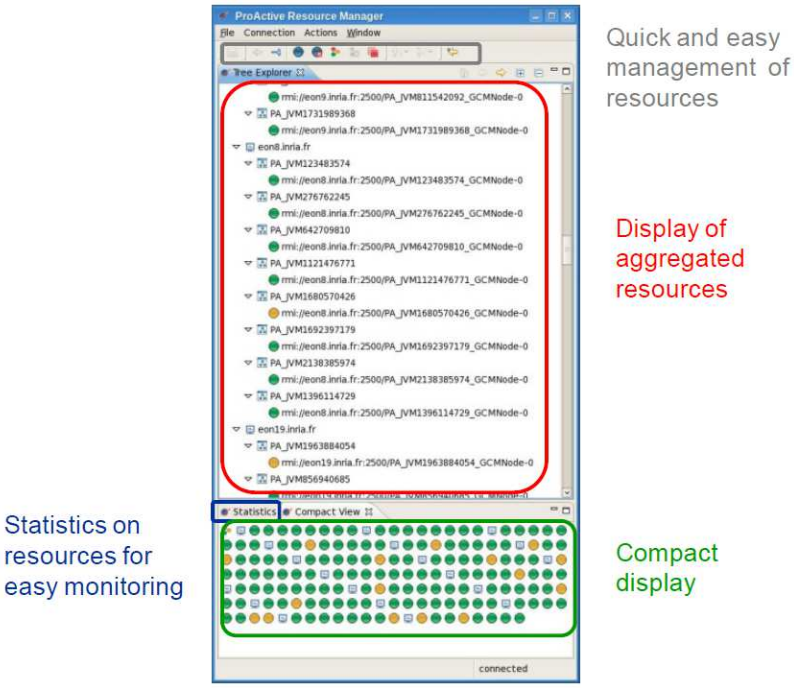

Figure 1.10: Screenshot of the *ProActive* Resourcing tool.

1.5.2 AGOS Project

AGOS [1], is a development project that integrates and standardizes a scientific approach (INRIA) and a industrial approach (HP, Oracle, ActiveEon and Amadeus) of 2 innovative technologies: Grid (Grid Computing) and Service Oriented Architecture (SOA).

AGOS is a secure platform that provides a:

- Reference architecture of the technology integration,
- Standard based services library,
- Set of tools and processes to build AGOS applications and Business/IT monitoring solutions,
- Methodology to migrate existing applications to AGOS architecture.

Companies can benefit from an automated and integrated IT management environment based on secured and distributed communicating agents. This project experiments the Open Source *ProActive*, a solution that implements the European component model for Grid computing, aimed at providing a concrete solution that a company with an extended or local network would implement. AGOS leverages recent progress in secured communicating agents and services technologies based on Open Source *ProActive* (European standards, portability and multi-platform/operating systems availability) that can support SOA architectures.

It provides tools and dashboards to monitor real-time company activity at the 3 levels:

- **Company management** level: At this level Business Intelligence (BI) covers two domains, company business and market trends analysis based on historical data. This requires access to various geographically distributed data and Grid computing is a key enabler of this domain. Real-time monitoring of company results in relation to the company objectives. Automation of business processes based on Information technology is growing and these business processes can be monitored using AGOS BI tools. AGOS approach provides Business real-time based decision to business and geo managers, what is made possible by the innovative standards and component base of AGOS layers.
- **Business process management** level: The manager of the business process is able to monitor his business processes without knowing which platforms and applications execute it, nor the IT problems that could affect it. Business processes is based on services that include a service level agreement that matches business requirements (response time, availability...)
- **System management** level: AGOS provides an integrated management of grid computing resources based on the service level agreed with the business group.

AGOS integrates these 3 levels, the definition of a new business process is able to leverage existing services (SOA approach). The new process will be implemented as a flow of services (such as product availability in stock service). One or several Service Level Agreements (SLAs) is associated to the process and the execution of the services is automatically deployed and managed on the Grid/SOA and reconfigured according to SLAs and resources. The Business process manager is able to real-time monitor the process activity and is integrated into the Company business dashboard for top management monitoring.

The role of IRMA in the AGOS Project is to provide a practical use case: a computing intensive application that can be deployed as a service in order to validate the AGOS Infrastructure.

[1] http://h30423.www3.hp.com/?fr_story=c4ec832e00c43f2791a9e4f15189d1cd5a91f819

1.6 Thesis structure

- **Chapter 2:** presents the hydraulic model behind IRMA for describing the water networks, including the different methods for calculating the head loss and the equipments that can be represented.
- **Chapter 3:** presents the types of simulation that can be performed by IRMA, including the *Clément*'s Formula that is used to estimate the peak demand in irrigation networks.
- **Chapter 4:** describes the algorithms used by IRMA for performing the different type of simulations presented in Chapter 3.
- **Chapter 5:** presents the different methods and libraries for solving linear systems that were analyzed for using with IRMA, the optimizations that were performed and the final proposed solution.
- **Chapter 6:** first presents the validation of the new IRMA Java and then it presents the most important feature improvements for the users: the performance gain regarding the FORTRAN solution, the elimination of the waiting time for the topology analysis phase and the new graphical user interface integrated with Geographical Information Systems.
- **Chapter 7:** presents two examples of distributed execution for validating IRMA's capability of harnessing resources on the Grid and Service Oriented Architectures. First, a demonstration of how the AGOS Infrastructure allows to deploy IRMA as a service that takes a set of networks as input and automatically launches IRMA instances on the available resources. The second example illustrates how to reduce the execution time of a very computing intensive simulation mode (pump profiling), by implementing a distributed simulation using the *ProActive Parallel Suite*.
- **Chapter 8:** concludes this work and summarizes the contributions obtained in this project.

2
IRMA Network Modeling

Contents

This chapter presents the hydraulic model behind IRMA for describing the water networks, including the different methods for calculating the head loss and the equipments that can be represented.

2.1 Overview

In order to describe a meshed network in IRMA, it is necessary to identify the **nodes**, **pipes**, **points with known charge**, its **equipments** and to specify the **discharges** at each point. The network in IRMA is represented as directed graph, which is a set of vertices (representing nodes) and edges (representing pipes.

A **node** is a junction of two (or more) connected pipes in the network. A **point with known charge** is a node where the head (piezometry) is known at the beginning of the simulation (such as a reservoir). An **equipment** is a device that creates a specific head loss that is added to the pipes' head losses, which can be due to friction, due to turbulent flows or it can be head gains in case of pumping stations.

From this description, IRMA needs to find some specific paths before starting the computation: all the closed loops and the path between points with known charge needs to be identified. These paths are used for building the equation system and also for calculating the probabilistic demand model.

The mathematical model used by IRMA is based on the Kirchhoff's circuit laws that states: *the algebraic sum of currents in a network of conductors meeting at a point is zero*. These laws can be also applied to water networks, they are used in IRMA to write equations that represent the flow conservation at the nodes and also the expected head differences between two nodes.

IRMA's system of equations is divided in 3 groups:

$$\begin{cases} \sum_i \sigma_i Q_i = 0, j = 1, ..., KN & (2.1) \\ \sum_i \epsilon_i J(Q_i) = 0, j = 1, ..., KL & (2.2) \\ \sum_i \epsilon_i J(Q_i) = \Delta H j = 1, ..., KE - 1 & (2.3) \end{cases}$$

where
$\sigma_i = -1$, if the pipe i is directed entering node j, else +1,
Q_i are the incoming and outgoing flowrates from each node,
$\epsilon_i = +1$, if the pipe i is directed in the same sense than the path j, else -1,
$J(Q_i)$ are the head losses associated to each pipe,
KN is the number of junction nodes in the network,
KL is the number of independent (non redundant) loops,
KE is the number of nodes with known charge, fixed head (tanks/reservoirs)
KX is the number of nodes in the network ($KN + KE$) and
ΔH is the head difference between the tanks.

The matrix can be noted as introduced in [Pil95]:
$AQ = 0\ M(J(Q) - A_f^T H_f) = 0_{kk-kn}$
where
A is the node-arc incidence matrix,
M is the loop-arc incidence matrix and A_f is the fixed head node-arc incidence matrix.

- First group of equations (Eq. 2.1) represents the *flow conservation at nodes*: sum of the flowrates in each node must be *zero*. The use of 0 in the second member allows to calculate the maximum pressure and it can be replaced by the discharged value in order to simulate the demands in each node. Each node is represented in the linear system by one equation that defines its connections with all other nodes: $'1'$ is used to represent a connection between two nodes and $'0'$ otherwise. These equations represent largest portion of the linear system and since each node is usually connected to a maximum of 3 other nodes the matrix representing the system will be sparse.
- Second group (Eq. 2.2) represents the *head loss equations in a loop*: sum of head losses from the first to the last node in each loop must be *zero*.
- Third group (Eq. 2.3) represents the *head losses in the paths between tanks*: head loss in each path between two tanks must be equal to head difference between first and last tank.

The unknown variables in the equation system are the flowrates in each node (Q_i). The heads (pressures) in each node are obtained from the flowrates, by calculating the pipes head losses. The number of equations in a connected network must respect the following equation:

$$KK = KN + KL + KE - 1$$

where KK is number of pipes (unknown flowrates).

The number of independent loops is given by:

$$KL = KK - (KN + KE) + NCC$$

where NCC is number of connected components (or sub-networks).

The first group is filled with linear coefficients but second and third groups contain non linear elements and must be linearized (Sec. 2.4) in order to be solved using traditional equations solving algorithms (Sec. 5.2).

Iterative mechanism: IRMA determines initial flow values for each pipe and then calculates the head in each node. The system is then iteratively solved by applying the Kirschoff's

laws to calculate the resulting flowrates and then updating the heads based on these new flowrates. This process is repeated until the sum of all differences between input and result flowrates are smaller than a determined convergence value or until a maximum number of iterations is performed (without reaching a stable state).

2.2 Head loss equations

The head loss (J) in each pipe is calculated in function of the flow (Q) and is given by the formula:

$$J(Q) = -(a + b.Q + c.Q^2) + \alpha.Q^\beta + D.Q^2 \text{ (positive flowrate values)}$$

where a, b, c represent pump coefficients (if it exists),
α represents friction head loss coefficients,
β is given by the head loss calculation method and
D represents singular head loss coefficient.

The equation needs to be adapted when the flowrates are negatives:

$$J(Q) = -(a + b.Q + c.Q.|Q|) + \alpha.Q.|Q|^{\beta-1} + D.Q.|Q|$$

with positive pump coefficients a, b and c.

This head loss expression is continuous and derivable. Therefore we can write, with $Q = Q_0 + \Delta Q$ and $J'(Q_0) = (\frac{dJ}{dQ})_{Q_0}$:

$$J(Q) = J(Q_0) + J'(Q_0).\Delta Q = (J(Q_0) - Q_0.J'(Q_0)) + J'(Q_0).Q$$

Head loss in a loop: the sum of all head losses in a loop is $null$. In a loop we have $J(Q) = 0$, therefore:

$$\sum J'(Q_0).Q = -\sum(J(Q_0) - Q_0.J'(Q_0))$$
First member . X = Second member

This formula will be used to write one equation for each loop in the network.

Head loss in a path between tanks: in these path equations the sum of all head losses in each path is equal to the head difference between the two tanks. In each path we have $J(Q) = \Delta H$, therefore:

$$\sum J'(Q_0).Q = -\sum(J(Q_0) - Q_0.J'(Q_0)) + \Delta H$$

This formula will be used to write one equation for each 2 tanks in the network. If a network contains KE tanks, there will be $KE - 1$ tank equations in the system.

2.3 Methods for calculating head loss

The methods for expressing head losses usually contain non linear elements in its formulas and thus they must be linearized in order to build the system of linear equations. IRMA can use different methods for calculating the head loss: Williams/Hazen, Lechapt/Calmon and Colebrook.

2.3.1 Colebrook-White

The unitary head loss in a pipe can be obtained using this formula

$$\Delta h = \frac{\lambda.V.|V|}{2.g.D}$$

where
λ is the Darcy-Weisbach friction factor ($\frac{64}{Re}$ for laminar flows),
V is the water velocity,
g is the acceleration of gravity ($9.81 m/s^2$) and
D is the pipe diameter.

This section presents two iterative methods for obtaining the *Darcy-Weisbach friction factor* λ: first a traditional method by using the original Colebrook-White formula and a new method based on the Lambert W-function ([CGH+96, CJK97]).

2.3.1.1 Traditional resolution

The Colebrook head loss coefficient is calculated by first obtaining the Darcy-Weisbach friction factor λ:

$$f(\lambda) = \frac{1}{\sqrt{\lambda}} + 2\log_{10}\left(\frac{\varepsilon}{3,7D} + \frac{2.51}{\mathrm{Re}\sqrt{\lambda}}\right)$$

where
λ is the Darcy-Weisbach friction factor,
ϵ is roughness height,
D is the pipe diameter and
Re is the Reynolds number.

This value is calculated traditionally using successive approximations (Newton's method):

$$\lambda_{n+1} = \lambda_n - \frac{f(\lambda_n)}{f'(\lambda_n)}$$

where
λ_{n+1} is the next approximation of λ_n,
λ_n is the last calculated approximation of λ,
$f'(\lambda_n)$ is derivative function of $f(\lambda)$, which is calculated as follows:

$$f'(\lambda) = -0,5\left(\frac{\frac{1}{\sqrt{\lambda}}}{\lambda}\right)\left(1 + \frac{\frac{2}{\log_{10}} + \frac{2.51}{\mathrm{Re}}}{\left(\frac{\varepsilon}{3,7D} + \frac{2.51}{\mathrm{Re}\sqrt{\lambda}}\right)}\right)$$

2.3.1.2 Using the $\omega-$function

Since the traditional method might need several iterations for achieving an accurate result, newer methods have been developed for optimizing this computation. For example, the iterative method defined in [Cla09] can achieve double precision accuracy by performing only two iterations.

This method considers a generic Colebrook-like equation as:

$$\frac{1}{\sqrt{\lambda}} = c_0 - c_1 ln\left(c_2 + \frac{c_3}{\sqrt{\lambda}}\right)$$

where the c_i are given constants. Then, the $\omega-$function is defined as:

$$y + ln(y) = x \Longrightarrow y = \omega(x)$$

In the terms of the the $\omega-$function, the solution for Colebrook equation is:

$$\frac{1}{\sqrt{\lambda}} = c_1\left[\omega\left(\frac{c_0}{c_1} + \frac{c_2}{c_1 c_3} - ln(c_1 c_3)\right) - \frac{c_2}{c_1 c_3}\right]$$

The iterative quartic scheme using the derivatives of the $\omega-$function is:

$$y_{n+1} = y_n - \frac{\left(1 + n_i + \frac{1}{2}\varepsilon_n\right)\varepsilon_n y_n}{\left(1 + y_n + \varepsilon_n + \frac{1}{3}\varepsilon_n^2\right)} \text{ for } n \geq 0$$

with

$$\varepsilon_i \equiv \frac{y_n + ln(y_n) - x}{1 + y_n} \text{ and initial guess } y_0 = x - \frac{1}{5}$$

2.3.2 Lechapt/Calmon

The Lechapt/Calmon formula used in IRMA for calculating the *unitary* head loss is:

$$\Delta h = \frac{L.Q.|Q|^{M-1}}{D^N}$$

where
Q is the flowrate,
D is the pipe diameter,
L, M and N are coefficients determined by pipe roughness.

For example, assuming a roughness value of $K = 0,25$ mm, we have:

$$\Delta h = \frac{1,16 \times 10^{-3}.Q.|Q|^{0,93}}{D^{5,11}}$$

2.3.3 Williams/Hazen

The general form of the Williams/Hazen equation relates the mean velocity of water in a pipe with the geometric properties of the pipe and slope of the energy line. The general form is expressed as follows:

$$V = k.C^{1,852}.R^{0,63}.S^{0,54}$$

where
V is velocity,
k is a conversion factor for the unit system (k = 0.849 for SI units),
C is a roughness coefficient,
R is the hydraulic radius and
S is the slope of the energy line (head loss per length of pipe or h_f/L).

This formula is specialized to pipe flows, the unitary head loss is then calculated by:

$$\Delta h = \frac{10,69.Q.|Q|^{0,852}}{CD^{4,871}}$$

where
Q is the flowrate,
C is a roughness coefficient and
D is the pipe diameter.

2.4 Linearizing head losses

Head loss values (J_1 and J_2) are calculated with one of these formulas using two reference flowrate values ($1m^3/s$ and $0.5m^3/s$). Then, a linear head loss coefficient α is determined by the equation:

$$\alpha = (2.\sqrt{J_1} - \sqrt{J_2})^2.(1 - \frac{\sqrt{J_1}-\sqrt{J_2}}{2.\sqrt{J_1}})^2$$

After obtaining the linear coefficient α, the following equation is used for calculating the head loss:

$$J(Q) = \alpha.Q.|Q|$$

Finally, matrix's head loss equations (loops and path between tanks) will be filled using this formula:

$$J'(Q_0) = \frac{J(Q_0+h)-J(Q_0)}{Q_0.h}$$

where h is a constant small value.

2.5 Modeling a sample network

This section presents a sample network containing the basic elements modeled by IRMA and its equation system when preparing the initial state for the iterative resolution.

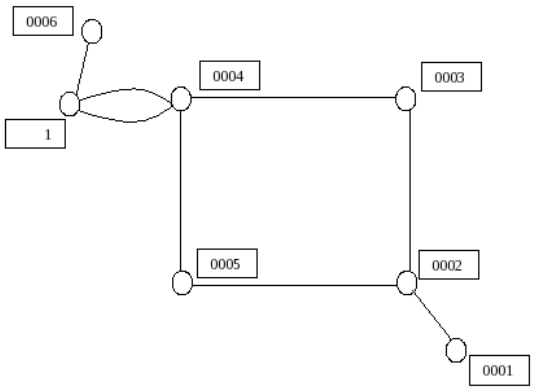

Figure 2.1: Sample network with 7 nodes, 8 pipes, 2 tanks and 2 loops.

Figure 2.1 shows a diagram of this network, nodes 0001 and 0006 are tanks and there aren't any equipments. In this gravity-flow network there are no offtakes or equipments. The loops are formed by the nodes $\{1, 0004\}$ and $\{0002, 0003, 0004, 0005\}$.

Figure 2.2 shows the equation system that represents the first iteration when solving the network from Figure 2.1. Lines 1 to 5 represent connections among ordinary nodes and since there's no consumption the second member is always equals to 0. Lines 6 and 8 represent loop equation and line 8 represents the path equation between the 2 tanks.

2.6 Equipments

This section presents the equipments that can be described using IRMA and how they are modeled.

$$\begin{bmatrix} -1 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & -1 \\ 0 & -5.82813 & -5.82813 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -612.153 & -612.153 & -58.2814 & -58.2814 & 0 \\ 5.82813 & 5.82813 & 0 & 612.1526 & 612.1526 & 0 & 0 & -58.2814 \end{bmatrix} \begin{bmatrix} Q_1 \\ Q_2 \\ Q_3 \\ Q_4 \\ Q_5 \\ Q_6 \\ Q_7 \\ Q_8 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -1.47132 \\ -53.3477 \\ -50.3808 \end{bmatrix}$$

Figure 2.2: Linear system that solves the sample network in a given iteration.

2.6.1 Pumps

Pumps are mechanical devices used for moving the water, they can be used to feed tanks or as a *booster* to increase the network's pressure. Figure 2.3 display an example of a pumping station.

Figure 2.3: Example of pumping stations.

Figure 2.4 shows a network scheme to illustrate the piezometric line behavior of a pump in a sample network. The piezometric head represents the elevation summed with the pressure head at each point of the network. In this case the pump is used to feed tank 2, located in the higher end of the network.

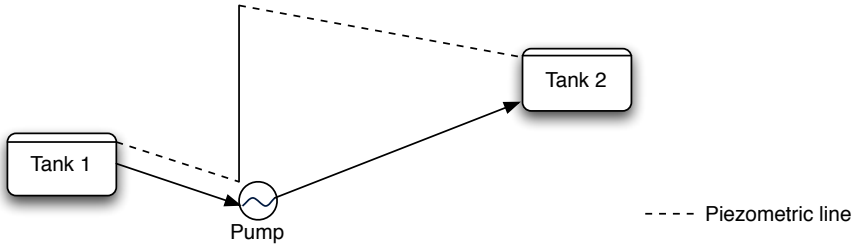

Figure 2.4: Scheme representing the piezometric level of a network with a pumping station.

IRMA supports several types of pumps, they can have variable or fixed behavior and therefore they are divided in two groups: fixed or variable speed pumps.

Fixed speed pumps have a constant behavior during the iterative simulation and are described by the curves that represent their Total Dynamic Head (TDH). The TDH of a pump is given by the following equation:

$$TDH(Q) = a + b.Q + c.Q^2$$

where a, b, c are the curve coefficients (usually given by the pump manufacturer) and Q is the flowrate. The TDH is obtained in function of the observed flowrate and then it will be applied as head gain to the pipe containing a pump.

Variable speed pumps adapt their behavior depending on an observed parameter. They can have 3 regulation modes:

- *Imposed regulation curve*: in this mode there are different curves associated with a flowrate interval. Whenever the observed flowrate is within an interval, the corresponding curve will be adopted.
- *Imposed head in a determined node*: this mode is used to find out the coefficient k so that $TDH(Q) = k^2.a + k.b.Q + c.Q^2$ can be used to obtain the desired head in a specified node.
- *Imposed flowrate value*: pumps in this mode are used to fill a tank using a chosen flowrate value ($Q_i = cst$). In this case, TDH is calculated automatically at each iteration in order to respect the hydraulic equilibrium.

2.6.2 Pressure regulating devices

Pressure regulating devices are used to control the pressure in a specific part of the network. There are 2 types of regulators: pressure reducing and pressure sustaining valves. The head loss for these equipments is automatically adjusted by IRMA at each iteration in order to achieve the desired head.

Pressure Reducing Valves are used to maintain the downstream pressure **below** a chosen head value (Z_{reg}). After each iteration i, the coefficient DH is adjusted for the iteration $i+1$ taking into account the difference between the obtained head and the objective head :

$$DH_{i+1} = DH_i + \frac{Z_i - Z_{reg}}{Q^2}$$

where
DH_{i+1} is the adjusted head loss coefficient for the next iteration,
DH_i is the current head loss coefficient,
Z_i is the obtained head,
Z_{reg} is the desired downstream head and
Q is the flowrate.

Figure 2.5 displays a scheme representing the piezometric line when the pressure reducing valve is active (left) and the case where it is inactive because the downstream head is below the desired head (right).

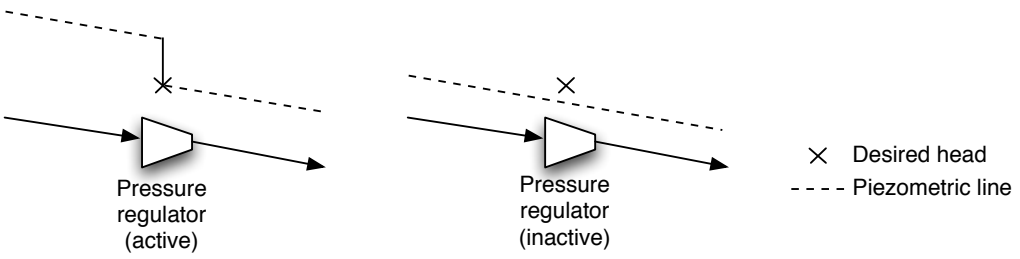

Figure 2.5: Piezometric schemes for a pressure reducing valve.

Pressure Sustaining Valves are used to maintain the upstream pressure **above** a chosen head value (Z_{reg}). After each iteration i, the coefficient DH is adjusted for the next iteration using the following formula:

$$DH_{i+1} = DH_i + \frac{Z_{reg} - Z_i}{Q^2}$$

where
DH_{i+1} is the adjusted head loss coefficient for the next iteration,
DH_i is the current head loss coefficient,
Z_i is the obtained head,
Z_{reg} is the desired upstream head and
Q is the flowrate.

Figure 2.6 displays a scheme representing the piezometric line when the pressure sustaining valve is active (left) and the case where it is inactive because the upstream head is above the desired head (right).

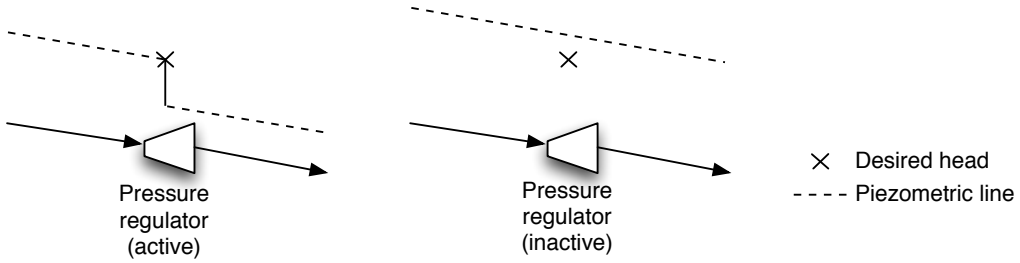

Figure 2.6: Piezometric schemes for a pressure sustaining valve.

If the water flows towards the direction that is not concerned by the valve (open position), a second head loss coefficient k_{open} can be applied using the following formula:

$$DH = k_{open}.Q^2$$

2.6.3 Singular head loss / Valve

A valve is treated as a singular head loss k that is constantly applied regardless of the flowrate direction.

$$DH = -k.|Q|.Q$$

where Q is the flowrate. If k has a high value, this equipment will be treated as a closed valve and this will represent a discontinuity in the network.

2.6.4 Differential head loss / One-way valve

A differential head loss is used to model valves that have different behaviors depending on the flow direction. In this case, two coefficients k_{pos} and k_{neg} are informed for calculating respectively the head loss in positive (Eq 2.4) and negative (Eq 2.5) position.

$$\begin{cases} DH_{pos} = -k_{pos}.|Q|.Q & (2.4) \\ DH_{neg} = -k_{neg}.|Q|.Q & (2.5) \end{cases}$$

where Q is the flowrate. If k_{pos} or k_{neg} has a high value, this equipment will be treated as a one-way (security) valve that only allows the water to flow in one direction. This valve is often used, for example, associated to a tank in order to avoid emptying it.

2.6.5 Flow control valves

Flow control valves are used in order to keep flow under a chosen value. In order to achieve the desired flow value (Q_{reg}), the head loss coeffcient DH are automatically adjusted at each iteration using this formula:

$$DH_{i+1} = DH_i + \frac{K_i}{Q_i^2}$$

where
DH_{i+1} is the adjusted head loss coefficient for the next iteration,
DH_i is the current head loss coefficient,
Q_i is the flow and
k_i is an adjustment coefficient (more details in Sec 4.11.4).

If DH_{i+1} becomes negative (causing a head gain), the value *zero* is imposed to the coefficient.

3

IRMA simulation modes and the Clément demand model

Contents

This chapter presents the types of simulation that can be performed by IRMA, including the *Clément*'s Formula that is used to estimate the peak demand in irrigation networks.

3.1 Continuous flows

Continuous flow is the basic simulation mode where all the demands are known and it is also used for performing the other types of simulation that are detailed later in this chapter. IRMA's system of equations (previously defined in Sec 2.1) is modified in order to take the demand into account and perform this simulation:

$$\begin{cases} \sum_i \sigma_i Q_i = D_n, j = 1, ..., KN & (3.1) \\ \sum_i \epsilon_i J(Q_i) = 0, j = 1, ..., KL & (3.2) \\ \sum_i \epsilon_i J(Q_i) = \Delta H, j = 1, ..., KE - 1 & (3.3) \end{cases}$$

where
Q_i are the incoming and outgoing flows from each node,
D_n is the node's demand,
$J(Q_i)$ are the head losses associated to each pipe,
ΔH is the head difference between the tanks.

Second member of equation 3.1 represents the demand at each node. These equations are then used to iteratively calculate the hydraulic equilibrium for scenarios with known demand values.

3.2 Peak consumption using the Clément demand model

The goal of this simulation mode is to compute the piezometric heads at the moment of peak consumption. Since this consumption is unknown, the flows are estimated at each pipe by using the *Clément's formula* [Clé66], which is a Peak Demand Diversity distribution. Basically, this formula takes as input the number of offtakes (among other parameters) at each point and calculates the flow in a given pipe based on the sum of all the estimatated downstream discharges. The offtakes are seperated in classes according to the nominal flow (estabilished on the contracts) and they are also associated to a probability according to the usage (for example, type of crop).

3.2.1 The general expression

The flow generated for a given class of offtake in a pipe is given by the general expression of *Clément's formula*:

$$Q = Q_{continuous} + \sum n_k.p_k.d_k + \epsilon.U\sqrt{|\sum n_k.p_k.(1-p_k).d_k^2|}$$

where
d_k is the nominal discharge of the offtake class k,
n_k is the number of offtakes from class k served by the considered pipe,
p_k is the opening (usage) probability of the offtakes from class k,
U is a parameter to determine the desired quality of service,
ϵ is the sense of the flow (represented by the sign of the term $\sum n_k.p_k.(1-p_k).d_k^2$)
$Q_{continuous}$ is the pipe's continuous flow (which is known in advance),
Q is the peak flow obtained in the pipe.

A class of offtake is defined by an association of *nominal discharge* and *opening probability*. Therefore, a new class should be created for each offtake with different flow value or probability.

When the distribution of a given class of offtake k is calculated, the distribution of the classes 1 to $k-1$ and the distribution of continuous flows are also known. The total flow in a pipe (in function of the number of offtakes) is given by the following formula:

$$Q = Q_{continuous} + n_k.p_k.d_k + \sum_{j=1}^{k-1} n_j.p_j.d_j + \epsilon.U\sqrt{|n_k.p_k.(1-p_k).d_k^2 + \sum_{j=1}^{k-1} n_j.p_j.(1-p_j).d_j^2|}$$

3.2.2 Adapting conservation equations to Clément formula

When applying *Clement's* formula, the equations need to be adapted in order to treat the conservation of number of offtakes instead of flow. For a given class of offtake, the flow conservation equations defined in Sec 2.1 are modified to:

$$\begin{cases} \sum_i \sigma_i n_i = 0, j = 1, ..., KN & (3.4) \\ \sum_i \epsilon_i J(n_i) = 0, j = 1, ..., KL & (3.5) \\ \sum_i \epsilon_i J(n_i) = \Delta H, j = 1, ..., KE - 1 & (3.6) \end{cases}$$

where n_i is the number of offtakes of this class served by the pipe.

Most networks have several classes of offtake and therefore this system needs to be solved successively for each class of offtake:

$$\begin{cases} \sum_i \sigma_i n_{ik} = 0, j = 1, ..., KN & (3.7) \\ \sum_i \epsilon_i J(n_{ik}) = 0, j = 1, ..., KL & (3.8) \\ \sum_i \epsilon_i J(n_{ik}) = \Delta H, j = 1, ..., KE - 1 & (3.9) \end{cases}$$

where n_{ik} is the number of offtakes of class k served by the pipe.

This means that the hydraulic equilibrium needs to be iteratively calculated for each class of offtake and the obtained results are used for calculating *Clément's* values for the next class.

3.2.3 Calculating usage probability for irrigation networks

One crucial aspect when performing simulations is to check if the model corresponds to what is actually observed in the network. IRMA allows to fit the probabilistic parameters based on the observed consumptions in the network.

Given a subset of network nodes, the usage probability of the offtakes of the class i is calculated by using this formula [Clé66]:

$$p = \frac{Si.v}{r.\sum_i (n_i.d_i)}$$

where
Si is the irrigated surface,
v fictive continuous flow,
r is the crop yield (efficiency)
n_i is the number of offtakes of the class i present in the subset of the network and
d_i is the nominal flow of the class i.

3.2.4 Issue with Clément when dealing with network ends

When applying *Clément's formula* to pipes serving a small number of offtakes (particularly when dealing with network extremities), the obtained flow is too small and therefore not acceptable because the resulting heads are too optimistic.

In a ramified network, this problem is overcome by replacing the *Clément's* flow by the resulting flow by summing the n bigger offtakes served by the pipe, whenever this sum is bigger than *Clément's* flow.

The drawback of this method is that it can only be applied to purely ramified networks. In the generic case of meshed networks, the discontinuity introduced in the flow calculation (when switching between *Clément's* flow and sum of n bigger offtakes) does not allow to correctly solve the equations that state the hydraulic problem (continuity and energy conservation equations).

The solution proposed to this problem with meshed networks is to adapt the probability of the offtake's usage P_k when dealing with small offtake populations. For large populations of offtakes, the probability p_k defined by the network designer is adopted. For small populations, P_k tends to a value p_{k0} higher than the original value p_k. The formula describing the evolution of P_k between p_{k0} and p_k is:

$$P_{k+1} = p_{k0} - (p_{k0} - p_k).\tanh(p_k.\sum_k n_k)$$

The hyperbolic tangent is used for obtaining a continuous evolution, since $\tanh(x)$ is equal to 0 to $x = 0$ and tends to 1 for big x values. The slope value for the evolution is defined by the multiplicative coefficient (in this case the original probability p_k is used).

3.3 Maximum pressure

This simulation mode is used to compute the maximum pressure observed at each node and assure that the obtained pressures are adapted to the infrastructure's specification. Since the discharges lower the network pressure, the basic assumption for this simulation is a scenario without discharges and all the tanks at the maximum level.

The maximum pressure might be obtained in different scenarios and therefore in some configurations it is necessary to perform multiple simulations. These scenarios depend mostly of the equipments that are present in the network and in some cases some theoretical equipments are added in order to model the state where maximum pressure is reached.

3.4 Equipment positioning

This section presents some typical examples for positioning pumps and one-way valves when performing maximum pressure and peak consumption simulations. Sec. 3.4.1 presents scenarios where pump stations (or boosters) are present and Sec. 3.4.2 presents scenarios where distribution is done by gravity-flow.

3.4.1 Pumping station to tank

Pump feeding upper tank: if the demand is present between the pumping station and the upper tank, there is no need of installing one-way valves (Fig. 3.1).

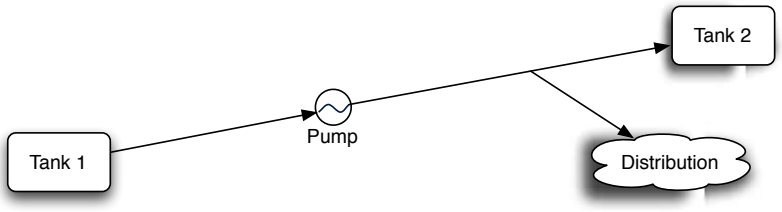

Figure 3.1: Network with demand between pumping station and the upper tank.

Downstream distribution from a tank in the extremity: in this case (Fig. 3.2) the one-way valve should be present in both simulations to avoid emptying the tank towards to the pumping station.

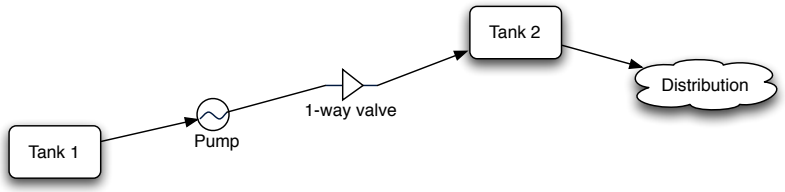

Figure 3.2: Network without demand between the pump and the tank in the extremity.

Booster pumping towards a tank: in this case (Fig. 3.3) the one-way valve is needed only to calculate the maximum pressure in the case the booster is off.

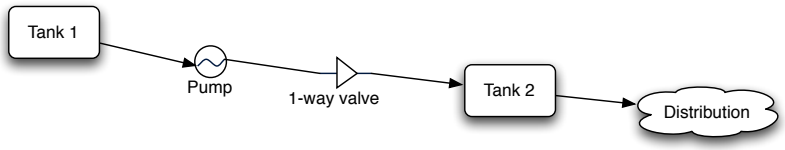

Figure 3.3: Booster pumping into a tank.

3.4.2 From tank to tank

Distribution between two tanks: in this case (Fig. 3.4) the one-way valve avoiding to feed the extremity tank 2 is needed but only for calculating the maximum pressure.

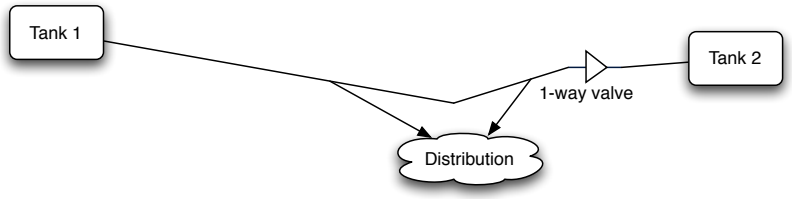

Figure 3.4: Distribution between two tanks.

Downstream distribution from a tank in the extremity: in this case (Fig. 3.5) two different one-way valves are needed. One valve avoiding to feed the extremity tank 2 for calculating the maximum pressure and one valve to avoid emptying this tank towards the upstream network.

3.5 Network profiling in function of a Pump

The network profiling represents the pumping head variation in function of the pumped flow, for given demand scenarios, pipes roughness values and the variation of level in the tanks. In order to calculate the envelope curve for this profiling it is necessary to know how the maximum and minimum heads vary at the pump (upstream and downstream) in function of these different possible scenarios.

In the case of a pumping station associated to a compensation tank, the following method is used:

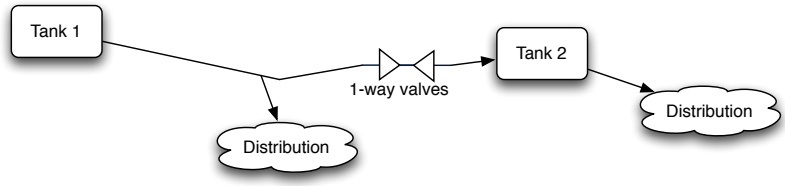

Figure 3.5: Downstream distribution from a tank in the extremity.

- Upstream from the pump: the minimum head is obtained when the peak demand is observed (calculated with *Clément*'s Formula), considering the maximum value for pipes' roughness and minimum level on the upstream tank. The maximum head is obtained when there aren't any discharges, considering the minimum value for pipes' roughness and maximum level on the upstream tank.
- Downstream from the pump: the minimum head is obtained when the peak demand (between the pumping station and the tank) is observed, considering the minimum value for pipes' roughness and minimum level on the downstream tank. The maximum head is obtained when there aren't any discharges (all the pumped flow reaches the tank), considering the maximum value for pipes' roughness and maximum level on the downstream tank.

In order to perform this computation, IRMA simulates the four different scenarios. The minimum envelope curve is obtained from the difference between the maximum downstream head and the minimum upstream head. The minimum envelope curve is obtained by the difference between the minimum upstream head and the maximum upstream head.

3.6 Pump Profiling

IRMA can also be used to help the design of new pumps in a network without tanks. Pump profiling is performed first by defining a fixed amount of flow (demand) objective values, varying from the minimum demand that can be observed (only continuous flows) up to a demand that is equivalent to 25% more than the estimated peak consumption.

Second step is to generate the possible discharge scenarios that will reach each objective flow value. These scenarios could be generated exhaustively in networks containing only a few nodes, however in most networks this is not possible and for that reason they will be randomly generated. The number of scenarios will be calculated in function of the number of offtakes, with a minimum of 250 scenarios in order to guarantee a minimum sample size in case of networks with a small number of offtakes.

Finally, all these scenarios will be simulated and the resulting value for each scenario will be the required head in the node where the pump will be placed in order to provide the guaranteed pressure to all clients. All this data will then gathered to plot one graph where each line represents the required head to satisfy a percentage of scenarios.

Figure 3.6 displays a sample result of this simulation in an existing network, objective flows are represented in the x axis, required head in the y axis and each line represent a % of scenarios that are satisfied.

3.7 Pressure driven analysis

The demand is usually fully satisfied under normal pressure conditions, but in scenarios with low pressures modeling the demand independently from the pressure is not realistic and may

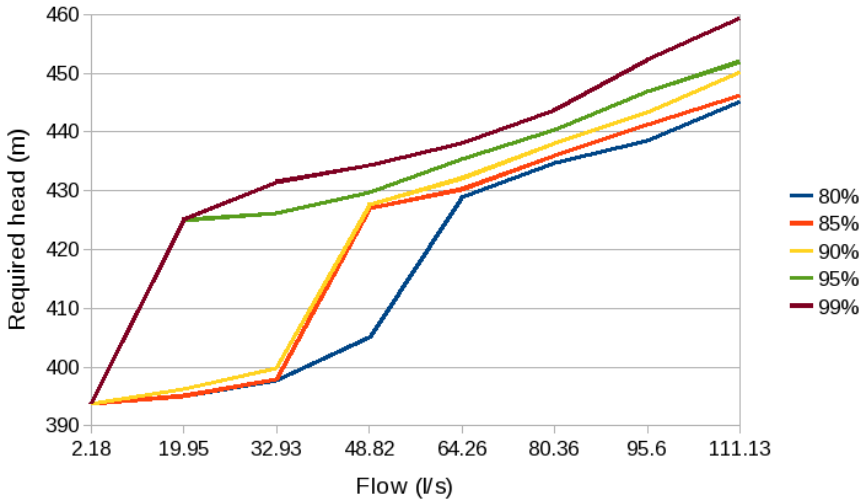

Figure 3.6: Pump profiling in the "Boutre" network.

often lead to negative pressures in the results.

The alternative for demand driven analysis is to take into account the pressure obtained in each point in order to estimate the actual demand. There are several approaches for performing this analysis, for example [GSK08] and [CVZR05] where it is presented an extension to EPANET to perform pressure driven analysis.

In IRMA the user can associate to each node a nominal flow value Q_n and a couple of bounding pressure values P_0 and P_1. The estimated flow Q is obtained as follows:

- If pressure P is between the bounding pressure values $[P_0, P_1]$, flow Q is taken as a fraction of Q_n equals to $[\frac{P-P_0}{P_1-P_0}]^{\frac{1}{2}}$
- If P is lower than P_0, Q is taken as 0
- If P is higher than P_1, flow Q is taken as:
  Q_n if there's a flow limiter
  $[\frac{P-P_0}{P_1-P_0}]^{\frac{1}{2}}.Q_n$ if there's no flow limiter

These coefficients will be iteratively calculated in function of the obtained pressures, until convergence is reached.

3.8 Discharge margin

This functionality was designed to help the network manager to know more precisely the network areas where it is possible to consider adding new discharges without deteriorating the existing clients.

The goal is to determine what consumption can be added (or should be removed), in terms of flow or number of offtakes, in a given served node in order to respect a condition given by the user. This result is obtained by adjusting the additional consumption on the target node and performing successive peak consumption simulations, until finding the consumption value that corresponds to the criteria given by the user.

At the end of the simulation, the detailed results of obtained flows and pressures in the network are given taking into account this additional consumption.

Two types of criteria can be defined, individually or combined:

- Exceeding pressure at the most unfavorable node served by the network.

- Exceeding velocity.

If there aren't any problems initially (pressure deficit or exceeded velocity), additional discharges will be added and the condition will be to reach null exceeding pressure or null exceeding velocity. If problems are present, discharges will be removed in order to reach an acceptable (or null) deterioration.

3.9 Extended period simulation

This mode allows to simulate the network behavior in a period of time, by performing a chain of static simulations that follow a variation law for the discharges. This law is expressed by a series of coefficients that are applied to the demanded flows and are used to distinguish the discharges in a certain number of time steps. For each time step, the hydraulic equilibrium is calculated in order to obtain flows and heads.

In order to perform this type of simulation, the discharges are transformed from number of offtakes to continuous flows in the following way:

- Calculation of the demanded flow at moment peak consumption (total flow that should be fed to the network by a unique source) by using the *Clément's* formula in function of offtakes of each class i and each serving point j.
- Calculation of the equivalent continuous flows d_j, demanded by each serving point j, by distribution of the demanded flow Q_p proportionally to the sums ($\sum$(number of offtakes x probability x nominal flow)

$$d_j = Q_p . \frac{\sum_i N_{ij}.p_i.q_i}{\sum_j \sum_i (N_{ij}.p_i.q_i)}$$

where
Q_p is the total demanded flow by the network at moment peak consumption,
N_{ij} is the number of offtakes of each class i and each serving point j,
p_i is the opening (usage) probability of the offtakes from class i and
q_i is the nominal flow of the offtakes from class i.

These results can be used to calculate the water flow in the tanks, the stored volume periodic variations by summing of the incoming and outgoing flows. Also, the tank level variations can be calculated from this results (if the data concerning the tank shape is known).

3.10 Pipe size optimization

This section presents the optimization model adopted by IRMA for calculating optimal pipe sizes using the Labye's iterative discontinuous method [Lab88]. This method is designed for ramified networks and is also used in other software, such as CEMAGREF's [xer91] and COPAM [LS00].

3.10.1 Overview

The optimization procedure involves 3 distinct phases:

- **Phase 1.** Construction of the "lower envelope curve" for a section. This indicates the minimum price of the section as function of the head loss in the pipe. (Sec. 3.10.2)
- **Phase 2.** Ascent of the network: moving upstream section by section, the lower envelope curves of the individual sections are summed either in groups or in series (Sec. 3.10.3 to Sec. 3.10.6) to obtain the network characteristic curve. This curve represents the total cost of the network as a function of the hydraulic head the upstream extremity.

- **Phase 3.** Descent of the network: moving downstream section by section, for a given hydraulic head at the upstream extremity selected as a function of pumping or reservoir costs, the diameter of each section is defined together with the hydraulic head at each node (Sec. 3.10.7).

3.10.2 Construction of the lower envelope curve

First step for calculating pipe sizes is to perform a peak consumption simulation (using *Clément*'s Formula) using a pre-defined diameter value for the pipes that should be optimized. Then, a list of suitable diameters for each section is constructed by calculating the velocity for each possible diameter D_i:

$$V_i = Q_i / S_i$$

where
Q_i is the peak flow observed at the pipe i and
S_i is the pipe's i sectional area.

Table 3.1 presents the diameters list used as reference by IRMA, including the unitary price and bounding values for minimal velocity (V_{Min}) and maximum velocity (V_{Max}).

Table 3.1: List of commercial pipe diameters with respective unitary prices and velocity limits.

Diameter (mm)	Price (eur)	$V_{Min}(m/s)$	$V_{Max}(m/s)$
1000.	677.00	0.65	3.10
900.	568.00	0.65	3.10
800.	458.00	0.60	3.10
700.	375.00	0.60	3.10
600.	302.00	0.50	3.10
500.	232.00	0.50	2.85
400.	174.00	0.50	2.50
350.	137.00	0.40	2.30
300.	117.00	0.40	2.30
250.	102.00	0.40	2.15
200.	84.00	0.35	2.05
150.	74.00	0.35	1.95
100.	62.00	0.30	1.85
85.	51.00	0.30	1.60
70.	49.00	0.30	1.60
58.	46.00	0.30	1.60
49.	43.00	0.30	1.60
39.	42.00	0.30	1.60

Next step is to create a list of pipes suitable for a given section $l(n)$, by calculating the total head losses and prices:

$$H_i = j_i.L_{(n)} \text{ and } P_{t(n)} = p_i.L_{(n)}$$

where
H_i is the total head loss,
j_i is the unitary friction loss for the diameter i,
$L_{(n)}$ is the length of the section $l(n)$,
$P_{t(n)}$ is the total price and
p_i is the unitary price for the diameter i.

The lower curve envelope is a curve representing the minimum price of a section as function of the permissible head loss in the section. This curve is created by plotting the slopes of the suitable pipes for the section. The slope (β) of a segment representing pipes i and $i+1$ is given by:

$$\beta_{i,i+1} = \frac{P_{i+1}-P_i}{H_{i+1}-H_i} < 0$$

Figure 3.7 presents an example of lower envelope curve, the extremities of the curve are filled with straight segments: a vertical segment rising from the point of minimum head, which indicates that the discharge cannot be conveyed with a smaller head loss, and a horizontal segment extending from the point of minimum price and which indicates that the cost of the section cannot be reduced even if excess head is available. This curve can be used for example to calculate the optimal diameters for a given head loss H_n. The benefit obtained by accepting an increase of head loss can be also calculated.

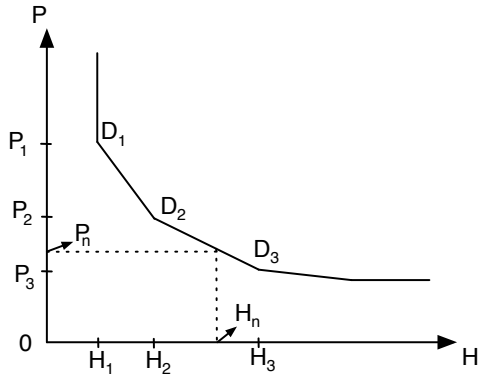

Figure 3.7: $P(H)$: Lower envelope curve or minimum price of section.

3.10.3 Ascent of the network

The ascent of the network is performed starting from the lowest point in the network for calculating the price curve P_i for each section. Here are presented the three possible section scenarios when calculating the price curve: end sections, sections in derivations and in sections in series.

3.10.4 Adding end sections

End sections are defined as sections downstream of which it is essential to satisfy a minimum piezometric head either because of the presence of a hydrant or because a branch originates at the end of the section.

The least-cost section, as function of the upstream piezometric head is obtained by shifting the lower envelope curve through a distance Z_m equals to the imposed minimum head (as shown in Figure 3.8).

3.10.5 Adding sections in derivations

Addition in derivation can happen when connecting either terminal nodes with a single section or sub-networks. In both cases the the lower envelope curve has been determined on the basis of a minimum piezometric head, since the head is added to the head loss curve for all end sections. Consider two terminal sections 1 and 2, the resulting envelope curve $P_i(Z)$ for the upstream section i is given by:

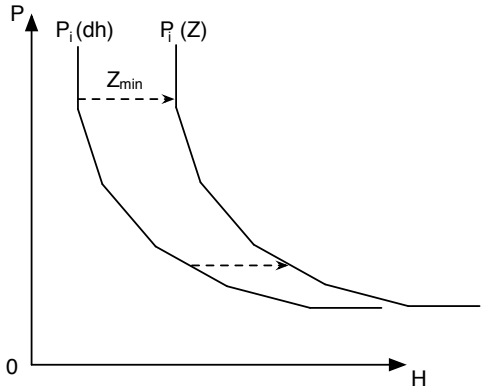

Figure 3.8: Adding the equivalent distance Z_m to the section's head loss curve envelope.

$$P_i(Z) = P_1(Z) + P_2(Z)$$

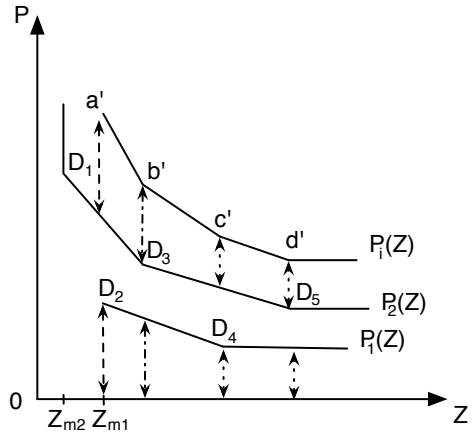

Figure 3.9: Addition of price curves $P_1(Z)$ and $P_2(Z)$ in order to obtain the upstream lower envelope curve $P_i(Z)$ in a derived section i.

Since the curves $P_1(Z)$ and $P_2(Z)$ consist of a series of segments, each intersection of these segments on either of the two initial curves will give rise to an intersection on the lower envelope curve of the common upstream node. Similarly, each segment forming the curve $P_i(Z)$ will have a slope equals to the sum of the slopes of the two initial segments:

$$\beta_i = \beta_1 + \beta_2$$

Figure 3.9 illustrates this derivation example in the ascent process: the price curves $P_1(Z)$ and $P_2(Z)$ represent the downstream curve envelopes and $P_i(Z)$ is the resulting price curve for the section i. The minimum head of the lower envelope curve cannot be less than Z_{m1}, which leads to the elimination of $Z_{m2} < Z_{m1}$.

3.10.6 Adding sections in series

Addition in series happen when a section i is added to a network, as illustrated in the left of Figure 3.10. The problem in this case is to define the lower envelope curve at the upstream node

n_u as function of the head in the downstream node n_d and of the head loss in the section i.

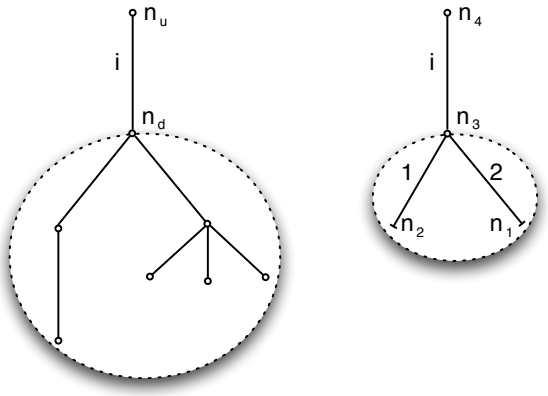

Figure 3.10: Examples of section addition in series: in the left addition of section i to a sub-network and in the right the addition in terminal sections.

As an example, in the right of Figure 3.10 section i is added to the sub-network formed by sections 1 and 2 added in derivation. The procedure is illustrated in Figure 3.11.

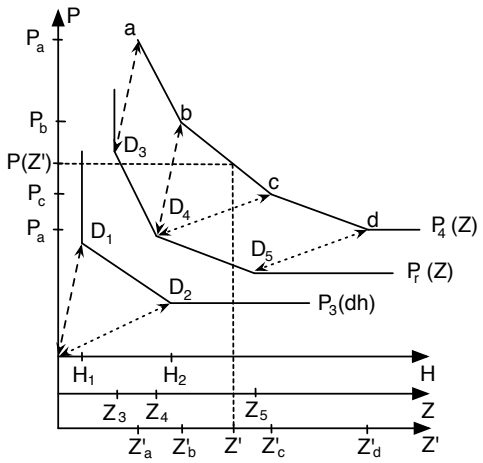

Figure 3.11: Addition of section i to the sub-network.

If $P_r(Z)$ is the lower envelope curve with respect to the head of the sub-network and $P_3(H)$ is the lower envelope curve with respect to the section 3, it follows that the minimum head at node 4 is the sum of the minimum head loss in section 3 and the head of the downstream sub-network formed by sections 1 and 2. The price curve is expressed by:

$$P_4(Z) = P_3(H_1) + P_r(Z_3)$$

The lower envelope curve $P(Z)$ represents the range of diameters of pipes of least cost for each head. Once these curves are calculated for all sections, all the parameters for determining the minimum cost of a network are available and the ascending process can be performed. When ascending the only assumption made is that the minimum (desired) head is known and the upstream head will be calculated for each suitable diameter for each section.

3.10.7 Descent of the network

The optimal head at the upstream end having been determined, the next step is to decide upon the head loss in each section. This is done by reading off the head from the lower envelope curve $P(Z)$ drawn for each node and selecting the diameter in function of $P(Z)$.

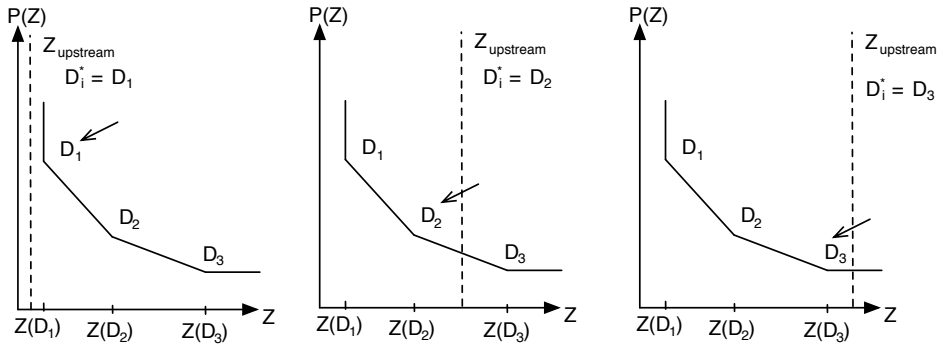

Figure 3.12: Examples of diameter selection in the descent process in 3 cases, considering $D_1 > D_2 > D_3$ and the selected diameter noted by D_i^* .

Figure 3.12 illustrates the selection of diameter in the descent process in 3 cases. In the first case (left) the obtained head in the upstream node is lower than the minimum head and therefore the largest diameter D_1 is selected. In the second and third cases (center and right), the smaller diameter that allows to obtain the minimum head is selected.

4
Specification of IRMA

Contents

This Chapter describes the algorithms used by IRMA for performing the different type of simulations presented in Chapter 3.

4.1 Overview

IRMA has 4 main classes to drive the simulation (Fig. 4.1): *IRMContext* where all the data needed for the simulation is gathered, *IRMAGraph* that stores the graph representing the network topology, *IRMASimu* that performs the iterative simulation and *LinearSystem* that stores the linear system representing $A.x = b$.

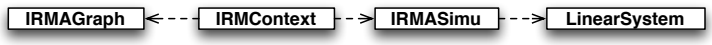

Figure 4.1: IRMA's main data classes.

The classes *IRMASimu* and *LinearSystem* can be extended in order to implement different simulation algorithms and different methods for solving the linear system.

Figure 4.2 displays a sequence diagram to illustrate a basic execution of IRMA, the pre processing phase consists in the following steps: IRMA first create an object of type IRMContext, that will then be filled by parsing the input file in IRMAInit, then a consistency check is performed in order to check for malformed networks and finally the topology analysis is performed in order to find all the meshes and the paths required for the simulation (as explained in Sec. 2.1). At this point IRMA is ready to launch the simulation, represented here by *IRMASimu*, but for each simulation mode the actual computation is performed by a class that extends *IRMASimu*.

4.2 Continuous flow simulation

4.2.1 Overview

Continuous flow simulation is used to calculate the hydraulic equilibrium in scenarios with constant discharges and is the building block for all the other simulation modes.

4.2.2 Initialization

Before starting the iterative simulation, IRMA fills the second member of the linear system $A.x = b$ for the first group of equations 3.1 with the constant discharges at each node. Then, IRMA gives an initial flow values Q_i for each pipe i, calculated in function of the diameter d_i:

$$Q_i = 2.\pi.d_i^2$$

4.2.3 Simulation steps

The iterative execution of the continuous flow simulation is illustrated in Fig. 4.3:

The simulation loop consists basically in iteratively executing the following steps until convergence is reached:

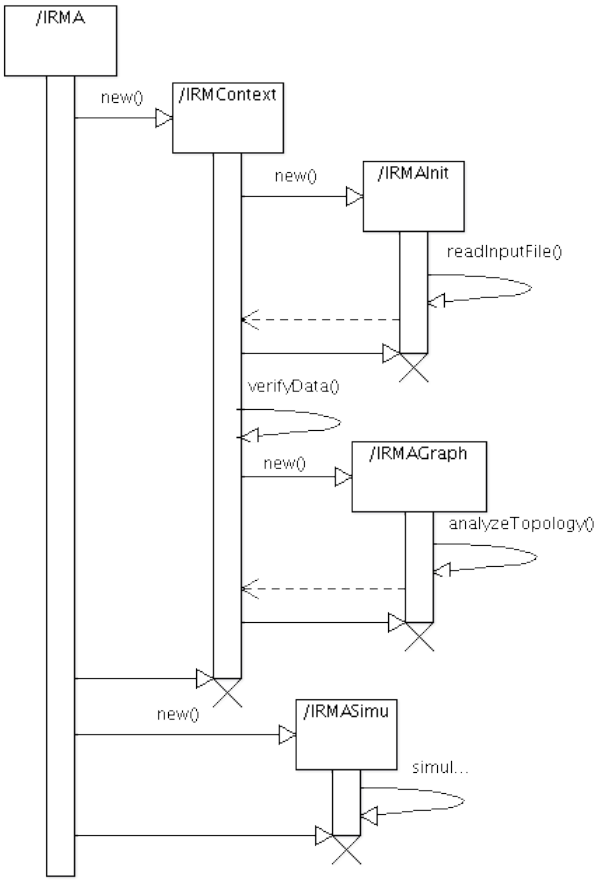

Figure 4.2: IRMA's simplified sequence diagram.

1. `initFlowParams()` : initialization of flow values Q_0 and ΔQ that will be used for writing the head loss equations for meshes and path between tanks (Sec. 2.3). In the first iteration Q_0 is calculated with the formula $Q_0 = 2.\pi.d_i^2$ and for the following iterations Q_0 is taken from the solution flow of the previous iteration.
   ΔQ is flow value obtained in function of Q_0 using the following formula $\Delta Q = Q_0 + |Q_0.h|$, where h is a small constant value.
2. `buildSystemAxb()` : initialization of the first member of flow conservation equations at the nodes;
   initialization of head loss equations in paths using $\frac{J'(Q_0)-J'(\Delta Q)}{|Q_0.h|}$ for the first member.
   The second member represents the sums of all head losses in the path. If the considered path is a loop, first and last node are the same and therefore the path head loss is equal to *zero*.
3. `solve()` : this method solves the linear system and stores the flow results for open pipes. For closed pipes (discontinuity), the flow is a known value (*null*) and the obtained result is the head loss observed at the pipe.
4. `calcHeadLosses()` : updates the head losses in all open pipes, using one of the proposed methods for calculating the linear head loss (Sec. 2.3).
5. `calcHeads()` : updates head values at each node using the new flow and head loss values.

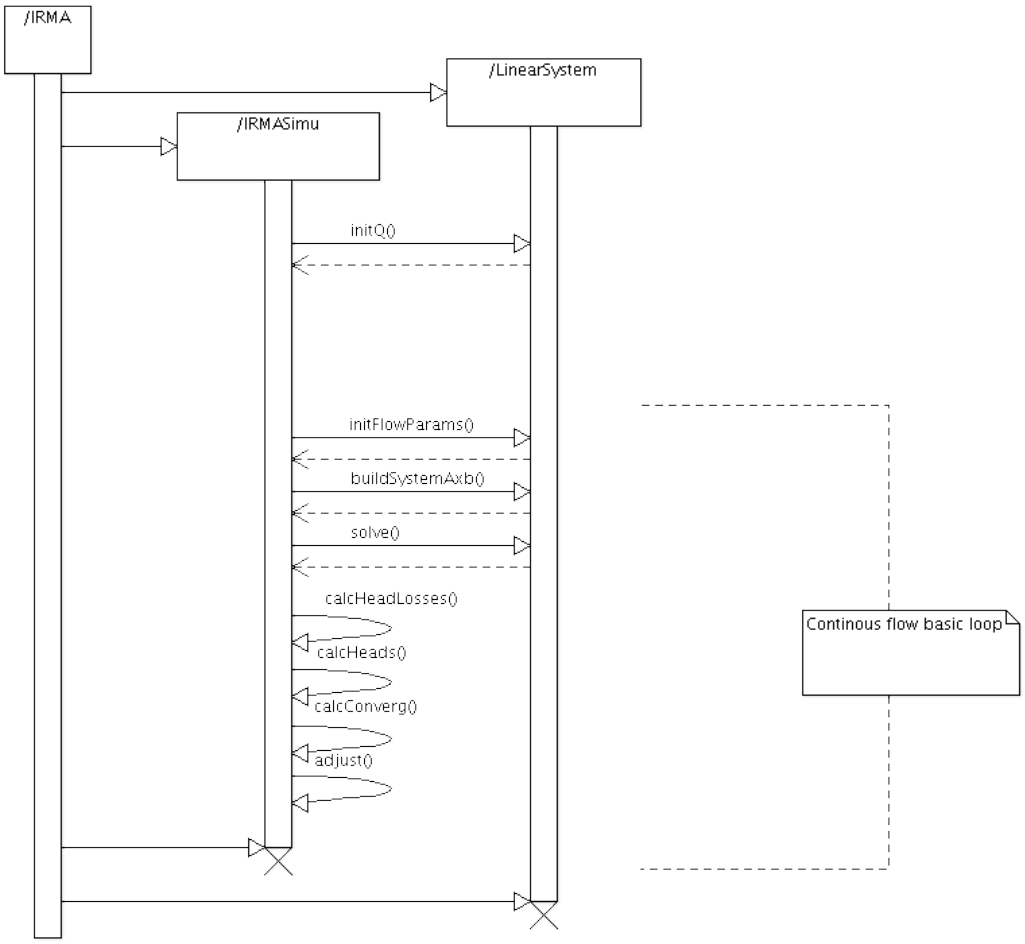

Figure 4.3: Continuous flow simulation sequence diagram.

6. `calcConverg()`: checks if the hydraulic equilibrium has been reached by comparing with results from previous iteration (more details at Sec. 4.5). If the resolution converges, the observed pipe flows, head losses and node heads will be taken as the result of this simulation.

7. `adjust()`: in case the resolution does not converge, the coefficients of some equipments might have to be adjusted in order to reach the hydraulic equilibrium. If there are no equipments, no adjustments have to be done among the iterations.

4.3 Maximum pressure

4.3.1 Overview

The computation of the maximum pressure can be performed in 1 or 2 steps, depending on the equipments that are present in the network and also depending on the results of the first simulation. In some cases, it is also needed to add hypothetical equipments in order to simulate states that each network might reach.

4.3.2 Simulation steps

Figure 4.4 presents the flow diagram for this simulation mode: the first simulation is performed and then if there is at least one pump present with significant flow (higher than $1l/s$), all the pumps are replaced by closed pipes and a second simulation is triggered. A particular case is treated in this second simulation: if there are disabled security valves associated to pumps they will be enabled in order to prevent flows in the wrong direction (against the pump direction).

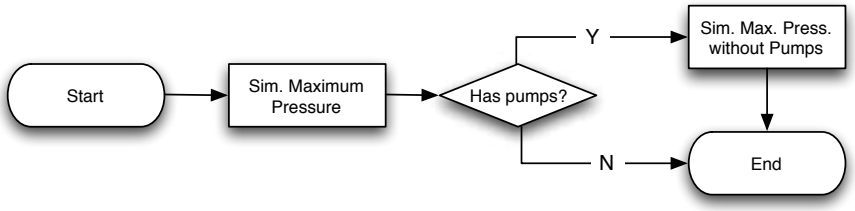

Figure 4.4: Flow diagram for maximum pressure simulation.

4.4 Peak consumption

4.4.1 Overview

The computation of the hydraulic equilibrium at the moment of peak consumption consists in a series of chained simulations, where the result of each simulation is used as input for the following simulation.

4.4.2 Simulation steps

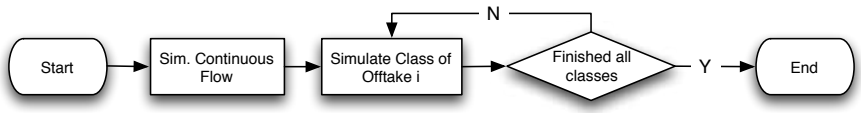

Figure 4.5: Flow diagram for peak consumption simulation.

The number of simulations performed is equal to $i + 1$, where i is the number of classes of offtake with different nominal flow or usage probability (also called classes of probability). Figure 4.5 display the flow diagram for the peak consumption simulation: the first simulation performed is the simulation with continuous flows, where all the known demands are taken into account. Then, the hydraulic equilibrium simulation is performed for each class of probability.

Figure 4.6 presents the sequence diagram that illustrates the simulation of one class of probability. The hydraulic equilibrium is calculated similarly to the continuous flow (as explained in Sec. 4.2) except for the following two methods:

- `initOfftakes()`: estimates the number of offtakes of the current class of probability that are being used at the moment of peak consumption (using the *Clément*'s Formula).
- `initClassParams()`: calculates the flow that will be adopted in the pipe, it takes the number of offtakes in use and convert to a flow value using either the original *Clément*'s Formula for the main parts of the network or the modified version if the pipe is located in the network extremity (as described in Sec. 3.2.4).

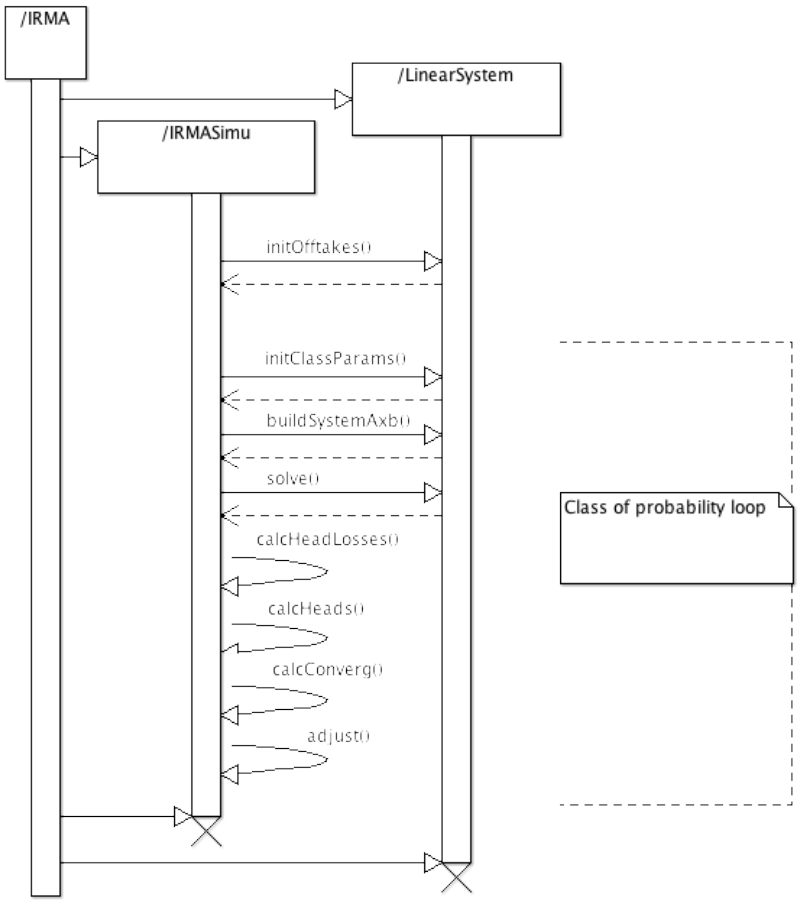

Figure 4.6: Sequence diagram for simulating a class of probability.

4.4.3 One way valve and Clément resolution issue

When performing a peak consumption simulation using the *Clément*'s Formula, the presence of one way valves may cause problems for estimating the demanded flow. These valves might close during a simulation and therefore they introduce a discontinuity in the network flows.

As workaround for this issue, all the one way valves are removed from the network model for performing the peak consumption simulation. At the end of the simulation, the flow direction is verified in the pipes where the valves have been removed: if the flow is going to the forbidden direction then the one way valve is replaced by a closed valve in the concerned pipe.

This process is repeated in the end of the second simulation, the fact of closing one (or more) pipes might affect the flow direction in other pipes and the closed valves may also be blocking flows in the allowed direction (and thus have to be removed). A simulation is performed with the new parameters in order to make sure all the flows are in the allowed direction.

4.5 Convergence parameters

This section presents the parameters used for determining the convergence of continuous flow and peak consumption simulations.

4.5.1 Basic parameters

When IRMA performs a **continuous flows** simulation, the first convergence test is regarding the variation of flow values in each pipe. For each pipe with a significant flow value (> 0.005 l/s), the following sum is calculated:

$$\Delta Q = \frac{\sum_{j=0}^{Npipes} \frac{|Q_i - Q_{i-1}|}{|Q_{i-1}|}}{Npipes}$$

where:
ΔQ is the average flow variation between current and previous iteration,
$Npipes$ is the total number of pipes in the network,
Q_i is the obtained flow value in the pipe for the current iteration and
Q_{i-1} is the obtained flow value in the pipe for the previous iteration.

If a **peak consumption** simulation is performed, a similar test is performed but instead of summing the flow values, the sum of the number of offtakes is taken:

$$\Delta n = \frac{\sum_{j=0}^{Npipes} \frac{|n_i - n_{i-1}|}{|n_{i-1}|}}{Npipes}$$

where:
Δn is the average variation of number of offtakes between current and previous iteration,
$Npipes$ is the total number of pipes in the network,
Q_i is the obtained number of offtakes in the pipe for the current iteration and
Q_{i-1} is the obtained number of offtakes in the pipe for the previous iteration.

Then the following test is executed:

$$\Delta Q \text{ or } \Delta n < TOL$$

where TOL is a constant tolerance value.

4.5.2 Flow control valve

If this first test fails, IRMA considers that the simulation has not yet reached the hydraulic equilibrium and triggers a new iteration. If it passes, IRMA continues to verify the remaining convergence parameters.

The second parameter tested is regarding the equipments with **regulated flow** value (imposed flow pumps or flow control valve). For each of these equipments the following value is calculated:

$$\Delta Q_{reg} = |Q_i - Q_{reg}|$$

where ΔQ_{reg} is the difference between the observed flow Q_i and the desired flow value Q_{reg}. The value than taken for the current iteration from the equipment with the higher ΔQ_{reg}.

4.5.3 Pressure regulating devices

For the **pressure regulating devices**, a ΔQ_{reg} is calculated in a similar way:

$$\Delta Z_{reg} = |Z_i - Z_{reg}|$$

where ΔZ_{reg} is the difference between the observed head Z_i and the desired head value Z_{reg}.

4.5.4 Paths equations

Finally, the last convergence parameter checked is regarding the **paths equations**: the average difference between the total head loss in the path and the head difference between the initial and the last node ΔP is calculated for each path (meshes and path between tanks) :

$$\Delta P = \frac{|\sum(\Delta H) - \Delta Z|}{P_{length}}$$

where
$\sum(\Delta H)$ is the sum of head losses in all pipes of the path,
ΔZ is the head difference between the first and the last node of the path and
P_{length} is the total pipe length of the path.

4.5.5 Overview

The convergence parameters can be summarized as follows:

$$\begin{cases} \Delta Q < TOL, \text{ for continuous flows simulation} & (4.1) \\ \Delta n < TOL, \text{ for peak consumption simulation} & (4.2) \\ \Delta Q_{reg} < TOL & (4.3) \\ \Delta Z_{reg} < TOLZ & (4.4) \\ \Delta P < TOL & (4.5) \end{cases}$$

where TOL is a constant generic tolerance threshold and $TOLZ$ is the head tolerance threshold that is usually set to require less precision than TOL in order to avoid convergence problems.

4.6 Network profiling in function of a Pump

4.6.1 Overview

In this mode IRMA simulates 4 scenarios for the different pumped flow values defined by the user. In order to determine the different scenarios and the pumped flow intervals, the following parameters need to be set by the user:

1. Minimum pumped flow.
2. Maximum pumped flow.
3. Minimum pipe roughness.
4. Maximum pipe roughness.
5. Number of steps.

4.6.2 Simulation steps

Figure 4.7 presents the flow diagram for performing this simulation: the starting pumped flow value is set initially to the minimum given by the user and then is increased at each iteration proportionally to the number of desired steps.

The result for this simulation is a table containing for each step (pumped flow value):

- Minimum and maximum heads obtained at pump's upstream node.
- Minimum and maximum heads obtained at pump's downstream node.

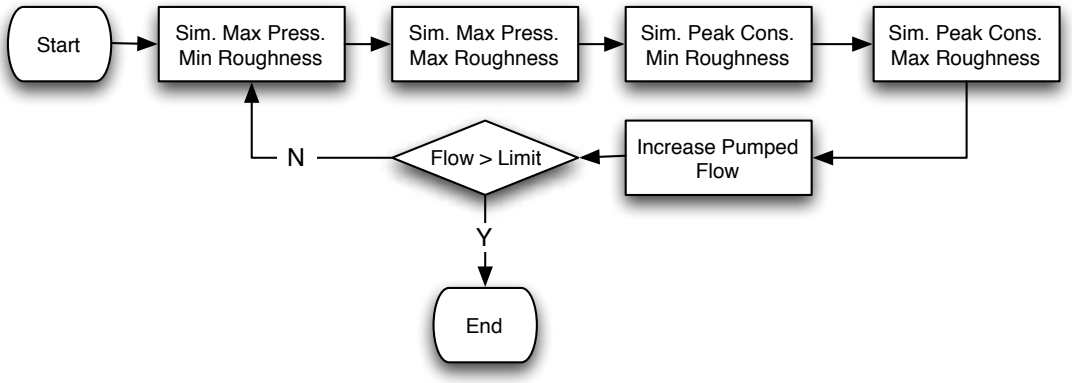

Figure 4.7: Flow diagram for the network profiling in function of a pump.

4.7 Pump Profiling

This simulation mode has two main steps, first the initialization of the network and of the different parameters that will be used for generating all the scenarios that will be simulated.

4.7.1 Initialization

The **initialization** phase consists in the execution of the following steps:

1. Definition of the **number of configurations** (scenarios) to simulate for each objective flow. This value is calculated as follows:

   $$N_{conf} = k.N_{hydrants}$$

   where
   k is a multiplicative coefficient that can be defined by the user and
   $N_{hydrants}$ is the total number of hydrants installed in the network.

   A minimum of 250 scenarios is imposed, in order to have a significant sample size for networks containing a small number of hydrants.

2. Definition of the **objective flow intervals** : a ΔQ is calculated in order to divide the flow interval $[0, Qmax]$ in equal parts respecting the limit of 20 objective flows.
   Q_{max} is defined as 25% more than the estimated peak discharges and it is obtained as follows:

   $$Q_{max} = 1,25.(Q_{peak} + Q_{cont})$$

   where
   Q_{peak} is the peak demanded flow (using the *Clément*'s Formula) and
   Q_{cont} is the sum of all known continuous discharges.

3. **Pumping station neutralization:** this step consists in removing the pumping station from the network description and updating the upstream tank head level:

   $$Z_{tank} = Z_{ref} + a_{pump}$$

   where
   Z_{ref} is reference value for the desired head in the upstream tank and
   a_{pump} is the TDH coefficient taken from the neutralized pump.

4. **Hydrants initialization:** transformation of the discharges represented by number of offtakes of each class into individual objects representing each hydrant (client) and the associated nominal flow.
5. **Computation of the piezometric line:** calculation of the static piezometric line by performing a hydraulic equilibrium computation considering the network without discharges and after the pump neutralization.

4.7.2 Simulation steps

The second step is to iterate through all the objective flows and simulate N_{conf} configurations (scenarios) where the demand is equals (or close to) the objective value. This process is described in the Figure 4.8.

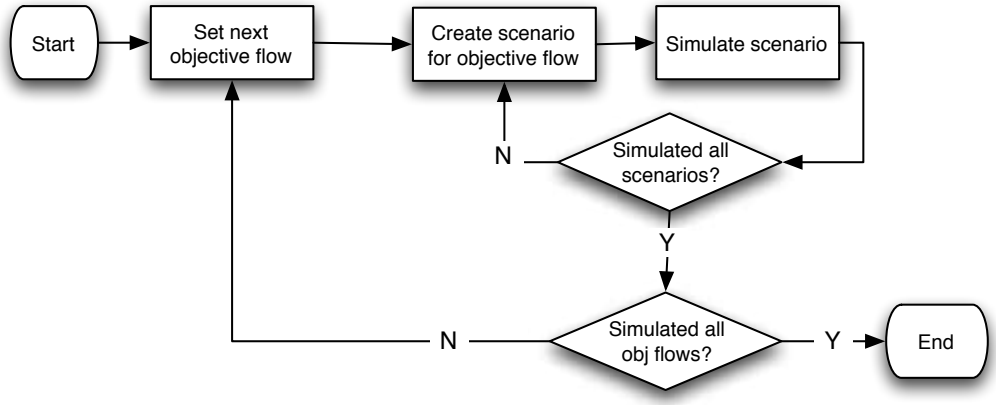

Figure 4.8: Flow diagram for the pump profiling simulation.

The result of each simulation is the required head where the pump will be placed, in order to obtain a satisfying pressure in all points of the network. This required pressure is obtained based on the observed pressure at the point with minimum exceeding pressure.

At the end of all simulations, a curve representing the required head in order to satisfy a certain percentage of scenarios will be plotted for each objective flow (Fig. 3.6, Sec. 3.6).

4.8 Pressure driven analysis

4.8.1 Initialization

The first step is to initialize the bounding pressure values $[P_0, P_1]$ for all nodes. These values can be given by the user or taken from the default values: $P_0 = 0$ and P_1 equals to the desired piezometric head at the node.

4.8.2 Simulation steps

The iterative process for finding the coefficients for nodes with pressure deficit is described in Figure 4.9: an initial simulation is performed with all the demands satisfied in order to find out where the pressure problems are. Then, the iterative process begins: nodes with pressure deficit will have their consumption coefficient adjusted (in order to respect the equations described in Sec. 3.7), until the mean differences in the actual demands (Δ_{mean}) between two iterations reaches a small value.

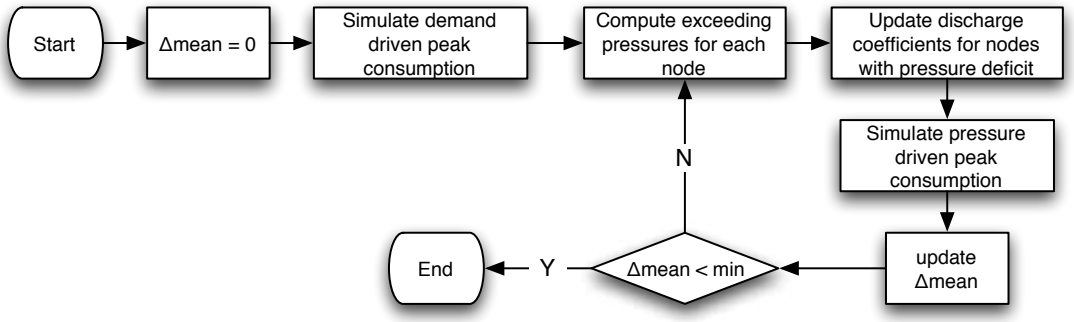

Figure 4.9: Flow diagram for the pressure driven analysis.

4.9 Discharge margin

4.9.1 Overview

In this mode, IRMA calculates the discharge that can be added to a node while respecting the user defined criteria of velocity and/or minimum pressure in the network.

4.9.2 Simulation steps

Figure 4.10 illustrates the main steps of this simulation, first a peak consumption simulation is performed without modifying the discharge and the initial values of exceeding pressure and velocities are stored. If a problem is detected in these initial values, the goal of the simulation is to calculate a reduction of the discharge in order to solve the pressure or velocity problem. The new discharge Q_{margin} is calculated iteratively, until the velocity and pressure criteria are respected.

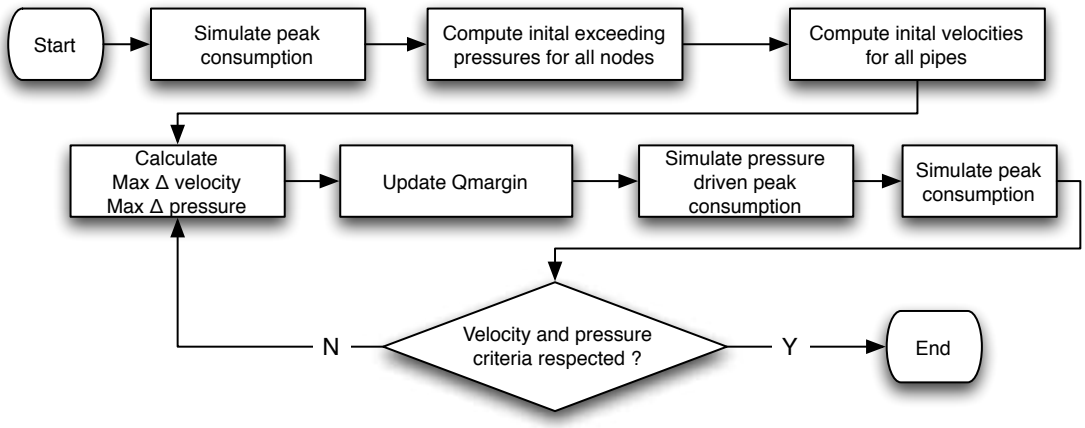

Figure 4.10: Flow diagram for the discharge margin calculation.

4.10 Dynamic network behavior

4.10.1 Overview

In this mode, IRMA simulates the network behavior over the time by using discrete time steps.

4.10.2 Simulation steps

The flow diagram of this simulation mode is described in Figure 4.11, the discharges described in number of offtakes are transformed into continuous flows using the *Clément*'s Formula and then the simulation for each time step is performed. In order to estimate the demand in a period of time, for each time step a multiplicative coefficient is applied to all discharges.

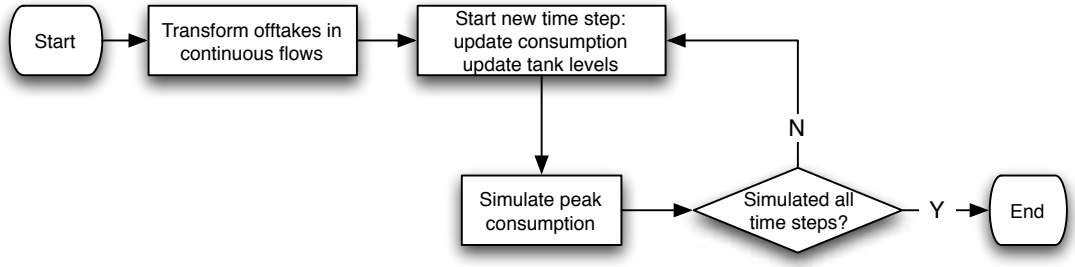

Figure 4.11: Flow diagram for the dynamic network behavior.

4.11 Equipments

All the equipments extend an abstract class that gives the main properties that are common to most equipments. These are the main methods declared in the `Equipment` class:

1. `init()`: initialization before each simulation of flow values and pipe state (open/close).
2. `initAdjustStep()`: initialization before each iterative resolution of the step for adjusting the head loss coefficient. This is needed by the equipments that automatically adjust the head loss, such as pressure/flow regulators and variable speed pumps.
3. `calculHeadLossCoeff()`: update the head loss coefficient for equipments with a variable head loss.
4. `calculHeadLoss()`: calculates the head loss in function of the observed flow Q. The generic formula is:

   $$DH = k.Q^2$$

   where k is the head loss coefficient. This method will be overloaded by the equipments that need to calculate the head loss using different formulas.
5. `calculConverg()`: calculates the Δ in order to check if the equipment has reached the objective head or flow value.
6. `adjust()`: this method is executed in case the resolution has not yet converged, as the last step of each iteration. Depending of the type of equipment, this method is used for triggering state changes (open/close or active/inactive) and the recalculation of the head loss coefficients.
7. `changeCoeffs()`: this method is used for changing the coefficients for each type of simulation, or to change the pump curves during a simulation.

4.11.1 Pumps overview

The relation among the different types of pumps (as detailed in Sec. 2.6) is illustrated in the class diagram (Figure 4.12).

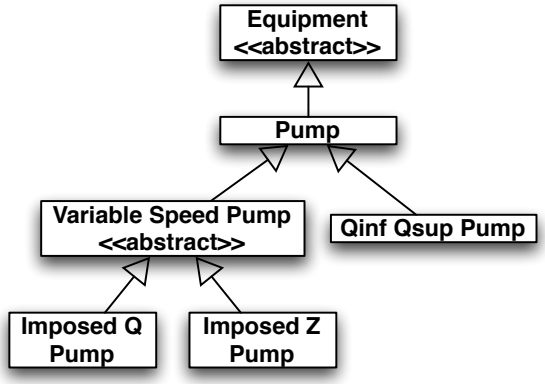

Figure 4.12: Class diagram for the different types of pumps.

The following methods are defined in the class `Pump` common for all types of pumps:

1. `calculNominalFlow()`: the nominal flow is used for initializing the pump flow before each simulation and also in case of divergence. This value can be informed by the user update the head loss coefficient or calculated with the following formula:

$$Q_{nominal} = -b - \frac{\sqrt{b^2 - 4.a.c}}{2.c}$$

   where a, b and c are the curve coefficients of the pump.

2. `calculHeadLoss()`: calculates the total dynamic head loss in function of the observed flow Q using the formula:

$$TDH = -(a + b.Q + c.Q^2)$$

   where a, b and c are the curve coefficients of the pump. The resulting TDH is negative in order to give a head gain.

3. `init()`: initializes pump flow value to the nominal flow of the pump $Q_{nominal}$ before starting each simulation.

4. `adjust()`: checks if the pump is diverging (flow is going into the wrong direction or is outside the curve describing the pump). If the pump is diverging, the nominal flow is taken as the pumped flow for the next iteration.

4.11.2 Variable speed pumps

Pump with imposed regulation curves (Qinf Qsup): this kind of pump adapts its behavior depending on the observed flow. This pump takes as input different curves associated to flow intervals. The following method is overloaded:

- `adjust()`: checks if the pump is working within the accepted flow interval and updates the coefficients if needed.

Pump with imposed head in a determined node: the class implementing this pump overloads the following methods:

1. `converg()` : calculates the difference between the obtained and desired head in the regulated node.
2. `adjust()` : updates the coefficients a, b and c in order to adjust the head gain and reach the desired head in the observed node.

Pump with imposed flow value: the class implementing this pump overloads the following methods:

1. `init()` : initializes pump flow value to the pump's imposed flow value before starting each simulation.
2. `converg()` : calculates the difference between the observed flow and the imposed flow.
3. `adjust()` : updates the coefficients a, b and c in order to adjust the head gain and reach the imposed flow.

4.11.3 Pressure regulators

Both downstream and upstream regulators share a common class as illustrated in Figure 4.13.

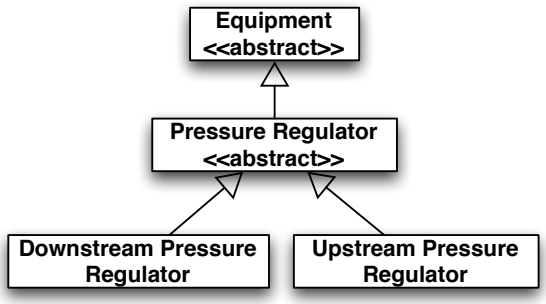

Figure 4.13: Class diagram for the pressure regulators.

In the abstract class `Pressure Regulator`, the following methods are overloaded from the class `Equipment`:

- `init()` initializes the head loss coefficient Δh to *zero*.
- `calculHeadLoss()` computes the head loss, using two different formulas depending on the observed flow Q:

$$\begin{cases} DH = \Delta h.Q^2 & \text{if } Q \geq Q_{min} \quad (4.6) \\ DH = \Delta h & Q < Q_{min} \quad (4.7) \end{cases}$$

This special treatment needs to be done for avoiding numerical problems when dealing with values very close to *zero*: the Δh coefficient tends to increase and yet it is not effective for increasing the head loss.

Downstream pressure regulator:

- `adjust()` updates the head loss coefficient using the formula:

$$\Delta h_{i+1} = \Delta h_i + k.\frac{Z_{reg} - Z_i}{Q^2}$$

where
Δh_{i+1} is the head loss coefficient calculated for the next iteration $i+1$,
Δh_i is the head loss coefficient from the current iteration i,
k is an adjustment coefficient for cases with small flow values,
Z_{reg} is the desired downstream head,
Z_i is the obtained downstream head and Q is the observed flow in the pipe.

If $Z_i < Z_{reg}$ and the obtained coefficient Δh_{i+1} is negative, then the pressure regulator will be deactivated (and therefore Δh_{i+1} will be taken as *zero*).

If the calculated coefficient Δh_{i+1} is smaller than the coefficient in open position k_{open}, then k_{open} is taken as the head loss coefficient for the next iteration $i+1$.

Upstream pressure regulator:

- `adjust()` updates the head loss coefficient using the formula:

$$\Delta h_{i+1} = \Delta h_i + k.\frac{Z_i - Z_{reg}}{Q^2}$$

where
Δh_{i+1} is the head loss coefficient calculated for the next iteration $i+1$,
Δh_i is the head loss coefficient from the current iteration i,
k is an adjustment coefficient for cases with small flow values,
Z_{reg} is the desired downstream head,
Z_i is the obtained downstream head and Q is the observed flow in the pipe.

If $Z_i > Z_{reg}$ and the obtained coefficient Δh_{i+1} is negative, then the pressure regulator will be deactivated (and therefore Δh_{i+1} will be taken as *zero*).

If the calculated coefficient Δh_{i+1} is smaller then the coefficient in open position k_{open}, then k_{open} is taken as the head loss coefficient for the next iteration $i+1$.

4.11.4 Flow regulators

Flow regulators have four input parameters:

1. Maximum desired flow Q_{reg1} for the positive sense (from node 1 to node 2).
2. Maximum desired flow Q_{reg2} for the negative sense (from node 2 to node 1).
3. Head loss coefficient k_{open1} in open position ($Q_i < Q_{reg}$ for the positive sense.
4. Head loss coefficient k_{open2} in open position ($Q_i < Q_{reg}$ for the negative sense.

In the open position k_{open1} or k_{open2} are taken as head loss coefficient. When the flow regulator is active, the head loss coefficient is adjusted at each iteration in order to reach the desired flow.

The class implementing the flow limiter extends the `Equipment` class and overloads the following methods:

1. `init()`: initializes to zero the head loss coefficient before starting each simulation.
2. `converg()`: calculates the difference between the observed flow and the regulated flow.
3. `adjust()`: if active, updates the head loss coefficient in order to reach the desired flow. If inactive, takes k_{open1} or k_{open2} as head loss coefficient.

4.11.5 Valve

The open valve (or singular head loss) uses only methods from the super class (`Equipment`), except for the constructor that initializes the head loss coefficient given by the user.

In the case of the closed valve (high value for the head loss coefficient), a variable indicating the discontinuity in the pipe is set to `true`.

4.11.6 Differential head loss

The differential head loss equipment takes two input parameters:

1. Head loss coefficient k_1 in the positive sense.
2. Head loss coefficient k_2 in the negative sense.

The class implementing the differential head loss extends the `Equipment` class and overloads the following method:

- `calculHeadLossCoeff()`: updates the head loss coefficient depending on the observed sense of the flow, taking k_1 if $Q > 0$ and k_2 if $Q < 0$.

4.11.7 One-way valve

One-way valve is a special case of the differential head loss and it has two input parameters:

1. Allowed sense.
2. Head loss coefficient k_{open} in open position.

The class implementing the One-way valve extends the `Differential Head Loss` class and overloads the following methods:

1. `init()`: initializes the flow before starting each simulation and also some variables for dealing with the peak consumption computation (more details in Sec. 4.4.3)
2. `calculHeadLossCoeff()`: updates the head loss coefficient depending on the observed sense of the flow, taking k_{open} if open and k_{max} if active.

This class also implements a private method:

- `calculIClose()`: checks if the valve must change the current state, by checking the sense of the flow.

5 Optimized linear system solving

Contents

This chapter presents the different methods and libraries for solving linear systems that were analyzed for using with IRMA, the optimizations that were performed and the final proposed solution.

5.1 Overview of methods for linear system solving

This section gives an overview of the methods found in the literature for solving linear systems expressed in the form $Ax = b$, where A is a square matrix containing the coefficients, x represents the variables and b represents the second member of the system.

They can be classified in two main groups: **direct** and **iterative** methods.

5.1.1 Direct methods

These methods attempt to solve the linear system by a finite sequence of operations and aim to obtain an exact solution for the problem. The **LU Decomposition** is an example of direct method, it consists basically in decomposing the A matrix in two triangular matrices (Lower and Upper), as illustrated in Figure 5.1. This decomposition can be obtained by performing the **Gaussian elimination**.

The matrix equation becomes $LUx = b$ and it can be solved in two logical steps:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}.$$

Figure 5.1: LU Decomposition: example of Lower triangular matrix and an Upper triangular matrix.

1. Solve the equation $Ly = b$ for y.
2. Solve the equation $Ux = y$ for x.

In both cases the matrices are triangular and can be solved directly using the forward and backward substitution.

In order to improve numerical stability, a technique called **pivoting** can be used. This technique has also the advantage of treating most types of systems (as long as they have a solution), without requiring special properties of the input matrix. The drawback is the complexity, as this method is very computing intensive.

5.1.2 Iterative methods

An iterative method is a mathematical procedure that generates a sequence of improving approximate solutions by usually giving an initial guess for the solution and then generating successive approximations. These methods in general require some properties and a specific conditions for solving a system, therefore iterative methods have often an associated procedure for preconditioning the problem into a suitable form. Here are some examples of iterative methods for solving linear systems:

- The **Conjugate Gradient** (CG) [She94] is an algorithm for solving systems whose matrices are *symmetric* and *positive-definite*. There is also a variant called **Biconjugate Gradient** (BiCG) that provides a generalization to *unsymmetric* matrices.
- The **Quasi-minimal Residual Method** (QMR) [FN91] solves non-hermitian systems and attempts to overcome irregular convergence behavior of the Biconjugate gradient method.
- The **Generalized Minimum Residual method** (GMRES) [Saa03] solves *unsymmetric* linear systems by approximating the solution vector in a Krylov subspace with minimal residual.
- The **Iterative Refinement** (IR) proposed by [WRB86] to improve the accuracy of numerical solutions to *unsymmetric* systems of linear equations.

5.2 Linear system solving libraries in Java

The model currently used by IRMA generates unstructured matrices containing up to 15.000 elements and the iterative nature of the resolution of the system need to be solved hundreds or up to thousands of times. Therefore, the performance of the linear solving library is a crucial element for executing the simulation in an acceptable time.

This section presents the most relevant libraries that were found and tested in the development process of the Java version of IRMA. A comparative performance chart of these libraries is presented in Sec. 5.4.

5.2.1 JAMA

JAMA (JAva MAtrix Package) [jam11] is a basic linear algebra package for Java and it was developed by the *National Institute of Standards and Technology* (NIST) at the University of

Maryland, USA. It provides user-level classes for constructing and manipulating real, dense matrices.

JAMA proposes five matrix decompositions, which consist of pairs or triples of matrices, permutation vectors, and the like, produce results in five decomposition classes. The five decompositions are: Cholesky Decomposition of symmetric, positive definite matrices, LU Decomposition (Gaussian elimination) of rectangular matrices, QR Decomposition of rectangular matrices, Eigenvalue Decomposition of both symmetric and unsymmetric square matrices and Singular Value Decomposition of rectangular matrices.

This package provides only support for dense systems, no representation for sparse matrices is offered.

5.2.2 Colt

Colt [col11] provides a set of Open Source Libraries for High Performance Scientific and Technical Computing in Java and it was developed by the *European organization for Nuclear Research* (CERN) [1].

Colt uses *JAMA* as underlying library for linear algebra and therefore offers the same five decompositions available in *JAMA*.

5.2.3 MTJ

MTJ [mtj11] is a library for developing numerical applications, both for small and large scale computations. The library is based on BLAS [LHKK79] and LAPACK [ABB+99] for its dense and structured sparse computations, and on the Templates project for unstructured sparse operations.

MTJ is based on the netlib-java project [net11], which can be set up to use machine-optimized BLAS libraries for improved performance of dense matrix operations, falling back to a pure Java implementation. This ensures perfect portability, while allowing for improved performance in a production environment.

MTJ implements 2 direct methods, LU and Cholesky decomposition, and it implements several iterative methods for solving linear systems, such as: Conjugate Gradient, BiConjugate Gradient, Chebyshev method, GMRES, IR and QMR.

5.2.4 BLAS and LINPACK

BLAS [bla11a] stands for *Basic Linear Algebra Subprograms*, it was first published in 1979 and is a standard API for libraries publishing basic linear algebra operations. The FORTRAN version is widely used but it has been also translated into other languages (such as C and Java).

LINPACK [DMBS79] is a collection of FORTRAN subroutines that analyze and solve linear equations and linear least-squares problems. It uses the *BLAS* libraries for performing basic vector and matrix operations. *LINPACK* offers decomposing methods such as LU, QR and Cholesky.

There are some Java translations of these libraries, for example in [bla11b] some methods from the *BLAS* libraries can be found and also decompositions translated from *LINPACK*'s FORTRAN version.

5.3 Proposed solution for the solver

5.3.1 Yale Sparse Matrix Package

This FORTRAN library [EGSS77] was written at the Yale University and it implements the *LU Decomposition* with sparse matrix storage. The resolution is divided in two phases:

[1] http://public.web.cern.ch/public/

1. **Pivoting:** consists in organizing the equation matrix assuring that it contains a *non-zero* element in the diagonal. The row permutations that need to be performed are stored in a *permutation matrix* and is given as input for the second phase.
2. **Resolution:** solves the system by performing the *LU Decomposition* using the **permutation matrix**.

The **pivoting** phase has the highest computational cost and this package also allows the reuse of the *permutation matrix*. Since we need to solve several times our systems and the *pivots* rarely change among the iterations, this feature can dramatically speed up our simulation.

In order to use this FORTRAN library from a Java application it is necessary to perform native calls using JNI (Java Native Interface). Since JNI does not allow direct access to FORTRAN code from Java, a C++ interface was written to wrap the FORTRAN library. The main drawbacks of this approach are reducing the portability of the Java code and complicating the compilation process, but this solution has been already successfully deployed in Windows, HP-UX Unix and Linux.

This library takes as arguments the list of system's *pivots* and the matrix stored using the Yale sparse matrix format. Therefore the last requirements for using this library are to search the *pivots* before first iteration (also after a few particular cases, e.g. when a pipe is closed between two iterations) and write the matrix respecting Yale's compressed format for matrix storage.

5.3.2 Yale's compressed matrix format

The sparse format stores only the *nonzero* entries of the matrix A, row-by-row in the array YA. To identify the individual *nonzeros* entries in each row, it is needed to know which column each entry lies. The column indexes which correspond to the *nonzero* entries of the original matrix are stored in the array JA; i.e, if $YA(K) = A(I,J)$ then $JA(K) = J$.

In addition, it is needed to know where each row starts and how long it is. The index positions in JA and YA where the rows of A begin are stored in the array IA; i.e., if $A(I,J)$ is the first *nonzero* stored entry in the $I-th$ row and $YA(K) = M(I,J)$, then $IA(I) = K$.

Moreover, the index in JA and YA of the first location following the last element in the the last row is stored in $IA(N+1)$. Thus, the number of entries in the $I-th$ row is given by $IA(I+1) - IA(I)$, the *nonzero* entries of the $I-th$ row are stored consecutively in:

$$YA(IA(I)), A(IA(I)+1), ..., YA(IA(I+1)-1)$$

and the corresponding column indexes are stored consecutively in:

$$JA(IA(I)), JA(IA(I)+1), ..., JA(IA(I+1)-1)$$

For example, the $5x5$ matrix:

$$\begin{pmatrix} 1 & 0 & 2 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 4 & 5 & 6 & 0 \\ 0 & 0 & 0 & 7 & 0 \\ 0 & 0 & 0 & 8 & 9 \end{pmatrix}$$

is stored as:

	1	2	3	4	5	6	7	8	9
IA	1	3	4	7	8	10			
JA	1	3	2	2	3	4	4	4	5
YA	1	2	3	4	5	6	7	8	9

In order to simplify the construction of the compressed matrix, a new Java class has been created. This class allows to progressively fill the sparse matrix, row-by-row and avoiding to allocate a dense matrix. This new class offers the following interfaces for dealing with the compressed format:

- `restart()`: reset to *zero* the counters of current row and number of elements stored.
- `addValue(val,j)`: add new value *val* to column *j* of current row.
- `endRow()`: finalize current row by consolidating the values currently stored into the 3 arrays format. This allows to add the elements in any column order for a given row.
- `setBeginPaths()`: set the beginning of the matrix's portion representing the mesh equations and the path equations between tanks.
- `goToPaths()`: after the first iteration, in most cases, only the path equations need to be updated. This method allows to perform an optimization by allowing to keep the first part of the matrix untouched and to modify only the path equations.

5.3.3 Building the JNI Interface to FORTRAN

The *Java Native Interface* (JNI) is a framework that enables Java code running on a *Java Virtual Machine* (JVM) to call native applications, compiled specifically to a hardware and operational system, written in other languages such as *C*, *C++* and *assembly*. However, JNI does not offer any interface to access FORTRAN code. In order perform FORTRAN calls from Java, a new interface needs to be created, for example, using *C++* to make calls to the FORTRAN library (Figure 5.2).

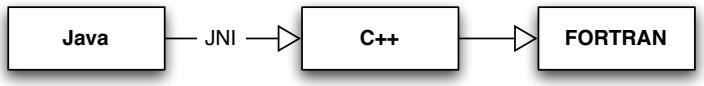

Figure 5.2: Calling FORTRAN code from Java through a JNI / C++ Interface.

FORTRAN routine signature: this is the simplified signature for accessing Yale's *LU Decomposition* FORTRAN routine (parameters not needed by IRMA are omitted):

```
SUBROUTINE CDRV (N, R, IA, JA, A, B, Z, FLAG)
```

where the input parameter:
N is the size of the square matrix A,
R contains the *pivots* of the system,
IA is the $1d$ array containing the row indexes in the Yale's compressed format,
JA is the $1d$ array containing the column indexes in the Yale's compressed format,
A contains the *nonzero* values of the matrix in the Yale's compressed format,
B is the second member of the system $A.x = b$,

and the output parameter:
B is the solution of the system (resulting flows or heads),
Z and $FLAG$ are errors detectors in case of problems in the system resolution.

5.3.4 Java interface

The **Java interface** consists in declaring a native method with the corresponding signature:

```
public static native float[] yale(int n, int[] r, int[] ia, int[] ja,
                                  float[] a, float[] b);
```

Then, the JNI will generate a header file containing the signature:

```
JNIEXPORT jfloatArray JNICALL Java_linear_1algebra_SolverYale
            (JNIEnv *, jclass, jint, jintArray, jintArray,
             jintArray, jfloatArray, jfloatArray);
```

This header file is then included in the C++ code.

5.3.5 C++ interface

The **C++ interface** has two roles:

- export a method that can be called from the Java application and
- call a FORTRAN method with the data retrieved from the JVM and transfer FORTRAN results to the JVM.

In order to access the fortran routine, the C++ code must declare the following interface:

```
extern "C" {
void cdrv_(int* n, int*r, int*ia, int *ja, float * a, float *b, float *z,
           int * flag);
}
```

Then, a standard JNI **C++ method call** is performed:

```
JNIEXPORT jfloatArray JNICALL Java_linear_1algebra_SolverYale (args) {

   jboolean iscopy;
   int flag = 0;

   int * ja = env->GetIntArrayElements(jJA, &iscopy);
   float * A = env->GetFloatArrayElements(jA, &iscopy);
   float * Y = env->GetFloatArrayElements(jB, &iscopy);
   int * iper = env->GetIntArrayElements(perm, &iscopy);
   int * ia = env->GetIntArrayElements(jIA, &iscopy);

   cdrv_(&n, iper, ia, ja, A, Y, Y, &flag);

   env->ReleaseFloatArrayElements(jB, Y, 0);
   env->ReleaseIntArrayElements(jJA, ja, JNI_ABORT);
   env->ReleaseFloatArrayElements(jA, A, JNI_ABORT);
   env->ReleaseIntArrayElements(perm, iper, JNI_ABORT);
   env->ReleaseIntArrayElements(jIA, ia, JNI_ABORT);

   return NULL;
}
```

This piece of code is very sensible regarding the performance and the memory management. Although this call is triggered by the Java application, the JVM does not handle the memory management in this code. The first group of calls are used to access the Java matrix data and pivots from the C++ code, by pinning the variables using the JNI interfaces.

Then, the FORTRAN routine (`cdrv_`) is executed and the variable containing the system's solution `jB` is transfered back to the Java application. The other pinned variables are released using the `JNI_ABORT` flag in order to optimize the call by avoiding copying them back to the Java stack.

The release of pinned variables is necessary even if the JNI code is declared `static` in the Java side, because if not released the JVM will not reuse the allocated memory area between two method calls and this will eventually consume all the available memory for a long simulation.

5.3.6 Compilation steps

Since the native code is hardware and operational system dependent, the compilation steps may vary from each configuration or compiler (except for the JNI header that is platform independent). This section explains all the steps for compiling the FORTRAN library and the C++ interface using, for example, a GNU/Linux platform.

The first step for compiling this new interface is to **create the JNI header file** from the Java class that declares the native call:

```
javah -jni SolverYale.java
```

This will generate the header file `SolverYale.h` that should be included in the C++ code:

```
#include "SolverYale.h"
```

Compiling the FORTRAN code as a shared library using, for example, the GNU G95 FORTRAN compiler [g9511].

```
g95 -fmultiple-save -ffixed-form -ffixed-line-length-132 -i4 -c -fPIC
    -freal-loops YALE.F90 -o YALE.o
```

where:
`-fmultiple-save` allows multiple declarations of the SAVE statement ,
`-ffixed-form` specifies the format of the source code,
`-ffixed-line-length-132` sets the length of each line to 132,
`-i4` sets the default size of the integer variables to 32 bits,
`-c` tells the compiler to create the compiled object only (without linking to an executable),
`-fPIC` tells the compiler to create position independent (relocatable) code and
`-freal-loops` accepts real variables to control loops.

This step will create the object file `YALE.o` that is used in the next step for **compiling the C++ interface**:

```
g++ -I/jvm/include/ -I/jvm/include/linux -fPIC libf95/*.o -shared
    YALE.o  Yale.cpp  -o libYale.so
```

where:
`-I/jvm/include/ -I/jvm/include/linux` are to include the JNI headers from the JVM,
`-fPIC` tells the compiler to create position independent (relocatable) code,
`libf95/*.o` includes the G95 FORTRAN needed libraries,
`-shared` tells the compiler to create a shared library (.so under linux),
`YALE.o` is the object file containing the FORTRAN routine
`YALE.cpp` is the source code file of the C++ interface.

This will create the shared library `libYale.so` containing the *LU Decomposition* FORTRAN routine, the needed G95 FORTRAN routines and the C++ interface for accessing the FORTRAN code.

In the case of the G95 FORTRAN compiler, the object files `libf95/*.o` need to be extracted from the file `libf95.a` (which is available with the compiler package) in order to be used in a shared library. This extraction can be made using the GNU `ar` command:

```
ar x libf95.a
```

5.4 Performance results

This section presents the performance results of the previously mentioned solvers when simulating the network with 1.763 pipes called *"TAMAGNON LA MARONNE LES PLAINES"*. This network was chosen as test case for benchmarking the solvers because of its size: it can be solved within a reasonable amount of time and memory using the solvers that store dense matrices.

5.4.1 Optimizing the Colt library for IRMA

This section presents the results of the optimization implemented in the Colt library in order to reduce the execution time of the IRMA simulations. The optimization consists basically in applying *two* changes to its *LU Decomposition* class:

1. Modification of the original method that performs the decomposition by creating a *permutation matrix* and storing it after the method invocation.
2. Adding a new solver method that uses the already stored *permutation matrix*.

The usage of these methods is straightforward: the first method is used for the first resolution and whenever IRMA knows that the *pivots* will change (e.g. security valve that opens or close). In all other cases the second method is used for solving the system.

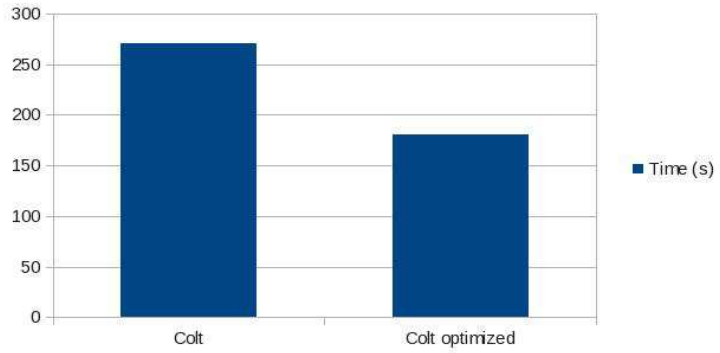

Figure 5.3: Execution time of a network using the original LU Decomposition from the Colt library and the optimized version.

Figure 5.3 presents the execution time of this simulation when using both the original and the optimized version of the LU Decomposition: the observed reduction in the execution time is of 33%.

5.4.2 Performance overview of the solvers

This section presents performance overview of the previously mentioned solvers when simulating the network *"TAMAGNON LA MARONNE LES PLAINES"*. These tests were performed in a Desktop PC running Linux except for the FORTRAN version that was executed in the production HP-UX Unix server.

Figure 5.4 shows the execution times: Jama, Colt and Blas are pure Java solutions, FORTRAN Yale is pure FORTRAN and Java Yale is the proposed mixed FORTRAN/Java solution.

In Colt optimized the results come from a modified library version where we adapted the LU Decomposition in order to save the *pivots* found on the first iteration and reuse them in the following iterations.

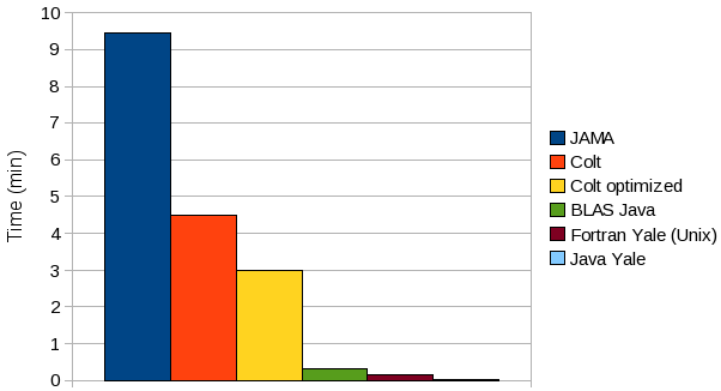

Figure 5.4: Execution time of a network simulation using different solvers in a Linux machine and the former IRMA FORTRAN version executed in a HP-UX machine.

In this example, BLAS Java presents acceptable performance for being a 100% Java solution but lacking sparse matrix storage it does not scale for simulating networks containing more than 3.000 pipes and therefore the only solution that is able to deal with large networks with a reasonable processing time is the proposed mixed FORTRAN/Java solution.

6

Results validation and improvements

Contents

This chapter first presents the validation of the new IRMA Java (Sec. 6.1) and then it presents the most important feature improvements for the users: the performance gain regarding the FORTRAN solution (Sec. 6.2 and 6.3), the elimination of the waiting time for the topology analysis phase (Sec. 6.4) and the new graphical user interface integrated with Geographical Information Systems (Sec. 6.5).

6.1 Validation of IRMA Java

Two main phases were performed in order to validate the results of tne new IRMA Java. The first and more comprehensive phase (Sec. 6.1.1) consisted in comparing the new results with the former version for all the 128 networks that are currently managed by the SCP. The second phase (Sec. 6.1.2) consisted in comparing the simulation results of a given network simulated with IRMA Java with the results of the same network simulated by a standard tool used for simulating WDNs (EPANET).

6.1.1 Comparing results with the former version

The first phase when validating the Java version consisted in comparing new results with the former FORTRAN version. This validation is performed automatically by a module that runs the former IRMA FORTRAN version, read the results file and compare them with the new version. The following values are compared: heads, flows, maximum pressures and head losses. This module reports if there's any difference superior to 1% in any of these values. This module has been used in order to validate all 128 networks from the *Maintenance* department. Table 6.1 presents an overview of all the validated networks, that represent a total fo 5.000 km of piped networks located in the south of France.

Table 6.1: Overview of the networks controlled by SCP's *Maintenance* department.

Networks	128
Pipe Extension	5.000 km
Tanks	240
Pumps	147
Pressure Regulators	322
Security Valves	82
Closed Pipes	2

This validation process also allowed to discover and fix some issues in IRMA FORTRAN version: thanks to the new IRMA version some problems in the equipment model and in the treatment of the input data have been found and corrected. For example, for a given pipe containing two equipments the FORTRAN version failed to treat both equipments. This happens due to the use of the same variable for representing different types of equipments and this has been corrected in the new model by defining a separate class for each equipment.

Another example is the incorrect behavior of a pump due to the usage of one integer variable in order to store two different informations: type of the equipment (identified by the absolute value) and the direction of the equipment (identified by the signal of the variable). This type of practice is error-prone and as a result of testing only the value (instead of absolute value), the pump was incorrectly treated when described in the negative direction. By avoiding this kind of practice, the new model is more clear and less vulnerable to this sort of coding mistake.

6.1.2 Comparing results with EPANET

IRMA's results have also been compared with results obtained with EPANET by using a module that read EPANET's *.inp* files. This module is able to read network topology, but currently it does not treat other sections (like equipments).

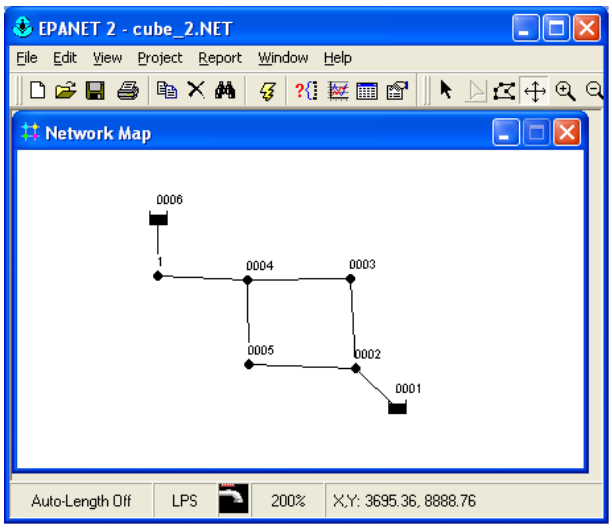

Figure 6.1: Screenshot of the simulated network designed with EPANET.

In order to compare both simulators, a sample network was designed using EPANET, exported through *.inp* file and read by IRMA (Fig. 6.1). It is a simple network containing two tanks and without any equipments.

In this test there are some differences in flow results, but head and head loss values obtained are exactly the same in both simulators. Table 6.2 shows the differences between EPANET and IRMA flow results, they vary in l/s but the proportional difference (in %) is constant.

Table 6.2: Flow differences between EPANET and IRMA in a sample network. Nodes marked with "*" are tanks.

Pipe		Flow (l/s)		Difference	
Node 1	Node 2	EPANET	IRMA	l/s	%
0006*	1	-683.87	683.06	0.81	0.12
1	0004	341.93	341.53	0.40	0.12
0004	1	-341.93	-341.53	0.40	0.12
0005	0002	-95.10	94.99	0.11	0.12
0004	0005	95.10	94.99	0.11	0.12
0003	0004	-588.77	-588.07	0.70	0.12
0002	0003	-588.77	-588.07	0.70	0.12
0001*	0002	-683.87	-683.06	0.81	0.12

6.2 Sequential performance benchmarks

This section presents the performance obtained with the new Java version and compares with the former FORTRAN version. The purpose of these tests is to highlight the performance improvement perceived by the user in most representative simulation scenario. In this case, usually only the number of offtakes are modified in the input file, the topology is not modified and therefore the topology analysis does not necessarily needs to be performed in the FORTRAN version. In these tests, the cached topology results are always given to the former version and the new version performs the topology analysis (with almost no impact in the execution time).

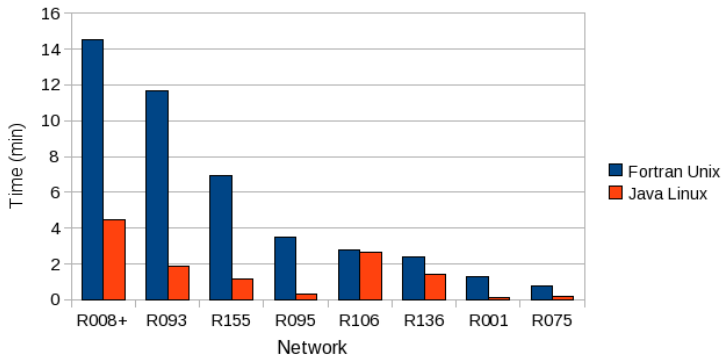

Figure 6.2: Execution time of selected networks using Java on a desktop machine and FORTRAN in the Unix server.

Figure 6.2 presents the execution times using original solution on the Unix server and the new solution on a Linux desktop machine. This set of networks is very representative regarding the network size, they contain the largest networks simulated in the *SCP* (with 1.845 up to 10.395 nodes).

The iterative resolution algorithms are very similar in both versions and they had similar performance results when using the original proprietary compiler. The performance improvement in the Java version is mainly justified by the adoption of a different compiler (G95) [g9511],

as explained in Sec. 5.3.6. However, it was not possible to compare both versions with the new compiler, because the former version currently runs only in the dedicated HP-UX Unix server and the recompilation procedure is too complex due to the high number of dependencies.

6.3 Convergence fine-tuning

Most networks reach stability (hydraulic equilibrium) after a few iterations when simulated with IRMA. However, it may also happen that several iterations are required and in some cases IRMA never reaches a stable state. This can be caused by several reasons, in most cases the equipments are responsible. The way equipments are modeled or the interaction of two or more equipments may cause new adjusts in coefficients in every iteration indefinitely.

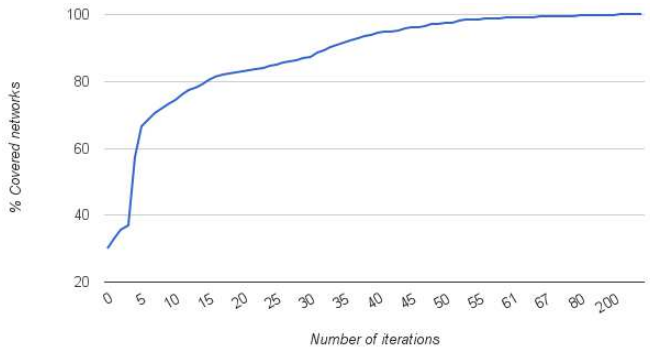

Figure 6.3: Number of iterations required for converging *SCP*'s *Maintenance* department networks. This figures exclude the networks that do not converge at all (about 10%).

Figure 6.3 shows the number of iterations performed before reaching a stable state in all networks from the *Maintenance* department, excluding pathological cases that do not converge at all. The main consequences of not converging are the lack of precision in the results and the increase of execution time because IRMA performs a maximum amount of iterations.

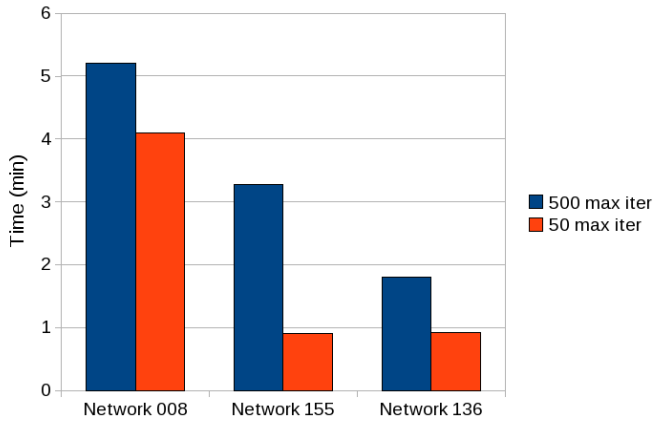

Figure 6.4: Execution time with 2 different iteration limits for 3 networks with convergence problems.

The former IRMA version has set as 500 the maximum number of iterations. In some simulations with convergence problems this limit is reached several times, because the simulation is performed for each class of offtake when calculating the peak consumption. In these cases where

stability is not reached, simulation time is much longer and accurate precision is not guaranteed even after 500 iterations.

In order to decrease execution time in these cases we have decided to change the maximum number of iterations, like in EPANET for example, where this limit can be set by the user. Figure 6.4 compares the execution time of 3 different networks with convergence problems when reducing this limit from 500 to 50 iterations. The given results with this modification are slightly different for flows (lower than 1%) but head values are preserved.

6.4 Topology analysis

Redesigning completely IRMA brought the possibility to improve it in many ways. The improvement that most impacted the users regarding the performance is the **topology analysis** phase. This section presents the new algorithm and data structure proposed for performing the topology analysis and the obtained performance results.

6.4.1 Proposed algorithm and data structure

Given a network, IRMA needs to obtain the following information about its topology in order to perform the simulation:

1. A chained list containing all ordinary nodes, in order to update the head values.
2. The list of paths between all the tanks (*two-by-two*).
3. The list of all non redundant meshes.

The creation of the first *two* lists is straightforward, the first list needs only to have a reference value for the head in each ordinary node: for the node next to the tank the head is calculated by the tank level (that is known) and the obtained head loss in the pipe connecting it to the tank. The process continues in the same way for the remaining nodes, the new heads are obtained using the heads that have already been calculated.

For the second list, a single iteration among all tanks is enough in order to obtain all the paths.

The process for obtaining the third list is more complex, because for every new discovered mesh IRMA needs to make sure that this new mesh is not redundant with the meshes already found. Figure 6.5 displays the flow diagram for building the list of non redundant meshes.

The data structure used to represent the graph is an *undirected multigraph*, meaning that edges do not have an associated sense and multiple edges are allowed. The graph library used for implementing it is called *JGraphT* [jgr11]. This library allows to build the graph by directly using IRMA's classes: the network nodes become the vertices and the pipes become the edges. *JGraphT* allows also to find the shortest path between two nodes using the Dijkstra's algorithm [Dij59].

6.4.2 Performance: FORTRAN vs. Java

Section 1.3.2 presents the time needed to perform the topology analysis using the FORTRAN version of IRMA (Table 1.1) and also explains the need of caching the results of the analysis in order to reduce the execution time. Some cases, very large networks and heavily meshed, can even require days to be analyzed.

With the simplification of the FORTRAN data structures (from cross referenced tables containing pointers and indexes going to a high level multigraph representation), the algorithm for executing the topology analysis could be heavily optimized.

This analysis now takes only a few seconds for any *SCP* network. This means that there's no need to use temporary files for caching topology results and the user can always modify the network structure with no impact in the execution time.

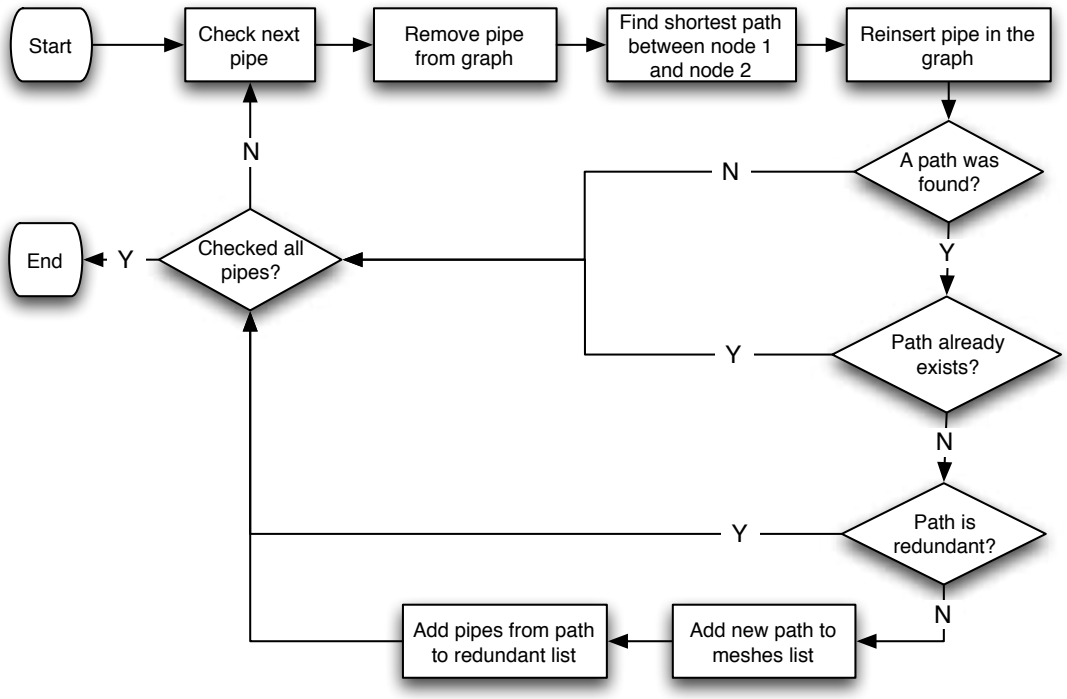

Figure 6.5: Flow diagram representing the process of building the list of non redundant meshes.

6.5 User interface

The new Java version of IRMA also brought the possibility of providing modern graphical interfaces for the users that otherwise would have been too costly to implement using the former FORTRAN version. Besides giving the possibility of inputting network data through user-friendly interfaces, the new interface is also able to deal with GIS (Geographical Information System) for visualizing the network graphically. This interface was developed using an open source toolkit called *uDig* (User-friendly Desktop Internet GIS) [udi11].

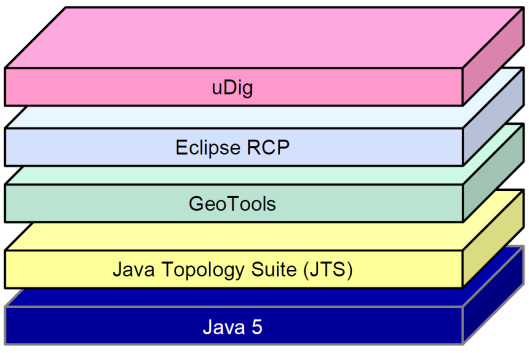

Figure 6.6: Diagram representing the software layers used by the uDig Toolkit.

Figure 6.6 illustrates the software layers used by uDig:

- Eclipse Rich Client Platform [rcp11] is a platform for building and deploying rich client applications.

- GeoTools [geo11] is a Java software library that handles infrastructural GIS tasks, like reading GIS file formats, connecting to spatial databases, rendering GIS objects into colored maps, etc
- The JTS Topology Suite [jts11] is an open source Java software library that provides an object model for Euclidean planar linear geometry together with a set of fundamental geometric functions.
- Java 5 is the Java Platform Standard Edition 5.0.

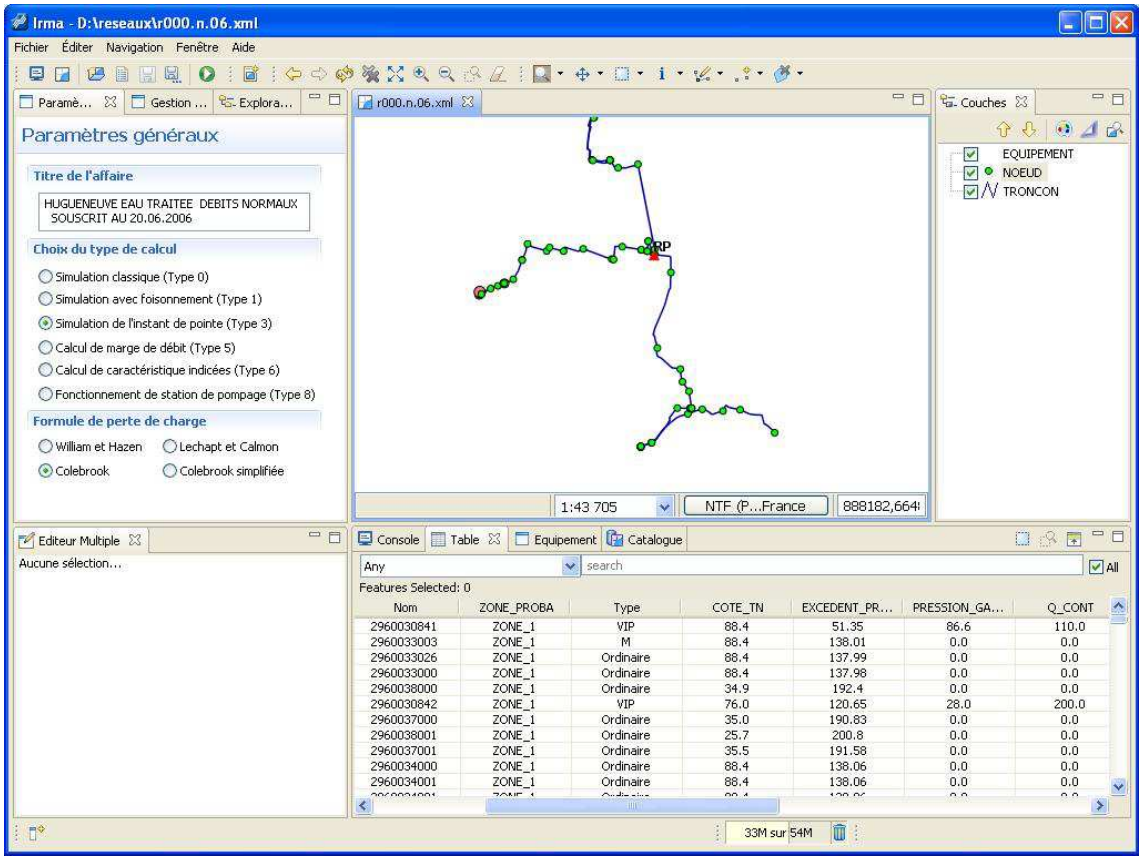

Figure 6.7: Overview of the new IRMA Java user interface.

Figure 6.7 illustrates a screenshot of the new interface. In the left pane some simulation options are displayed: name of the network, choice of the simulation type and head loss calculation method. In the top right pane the network is represented graphically: blue lines represent the pipes, nodes are represented by circles and equipments are represented by triangles (in this case RP stands for pressure regulator). In the bottom right tab the nodes are listed using a table format, including information such as node name, terrain level, obtained pressure, probability zone, discharges, etc.

Figure 6.8 illustrates the edition of equipments using IRMA. In this example the equation of a pump is being edited and the the curve representing its Total Dynamic Head (TDH) is also displayed.

6.5.1 Database interconnection

In the former version of IRMA, the network files used for simulation were stored in text files without geographical information for representing the network graphically. This data is stored in a spatial database and it can be easily integrated by using a modern GIS, like uDig.

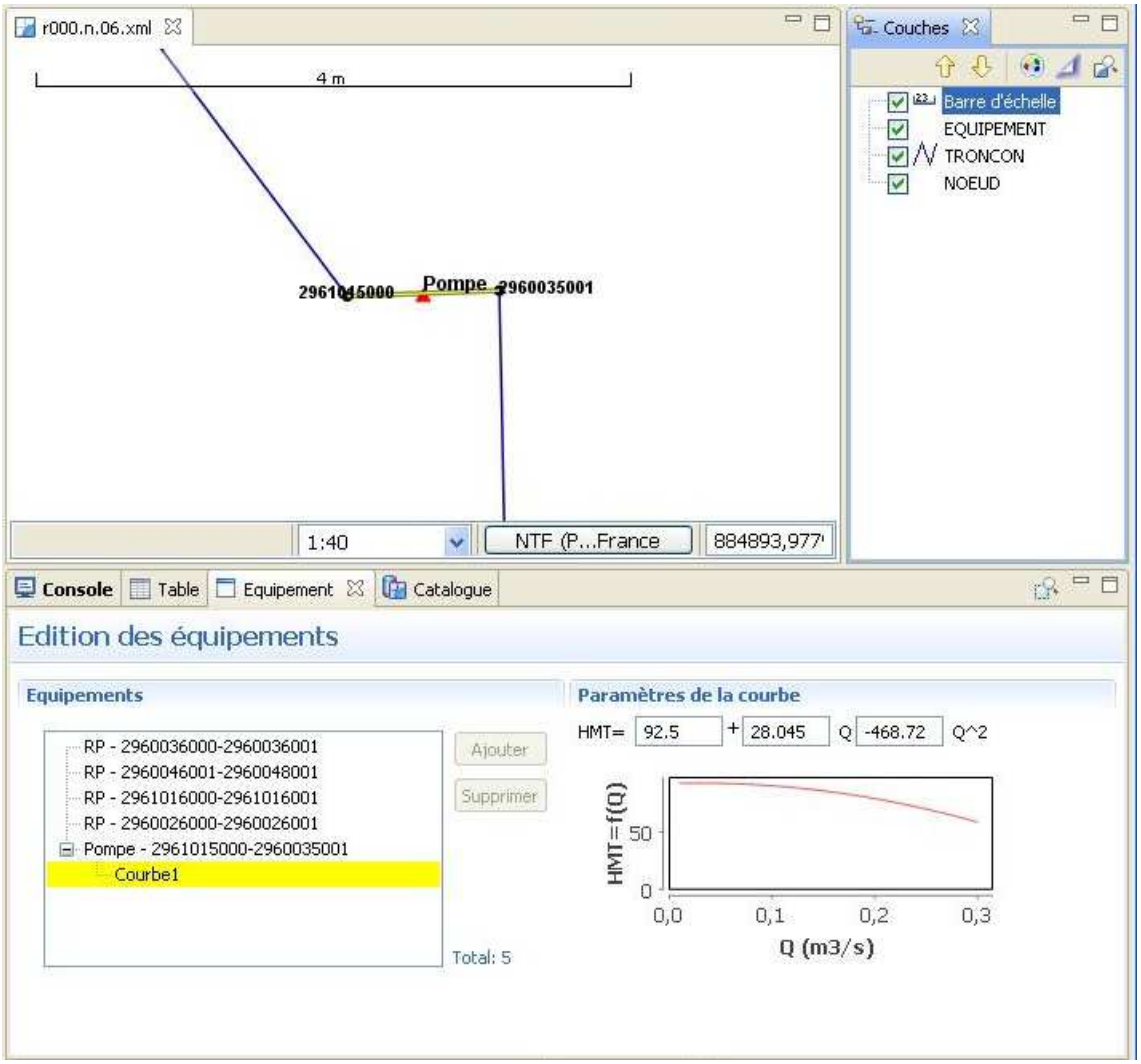

Figure 6.8: Equipment editing with IRMA.

Figure 6.9 illustrates an example of a network that is stored using the legacy network files (without geographical information) and is graphically displayed using the uDig toolkit. uDig allows to integrate the network information from the legacy file with the spatial database and also to add different layer in order to enrich the visualization experience. In this example, a layer contaning a map is added in order to help locate the network.

6.5.2 Result visualization

The obtained results in IRMA are typically given by a report that begins with an overview of the network: pipe lengths and diameters, list of equipments, probability zones with the respective number of offtakes of each class, global flow information (peak consumption, total number of offtakes, total continuous flows). The second part of the report is filled by a table containing key input and result values for all network pipes, such as:

- Diameter (input)
- Length (input)
- Flow (result)

Figure 6.9: Example of integration of legacy IRMA network file with existing SCP spatial database. This diagram represents the *"Hugueneuve"* network, in the south of France next to Toulon.

- Velocity (result)
- Unitary head loss (result)
- Terrain elevation (input)
- Obtained head at node (result)
- Desired head at node (input)
- Maximum pressure at node (result)

The new interface allows to visualize these results graphically, making it much easier to understand the results and to identify the pipes and nodes in the map.

Figure 6.10 illustrates the pressure results visualization with IRMA, the network is plotted over a geographical layer containing buildings, contour lines, roads, etc. The circles in the legend represent the obtained pressure in the nodes: from green circles that indicate acceptable pressure to black circles that indicate a very large pressure deficit.

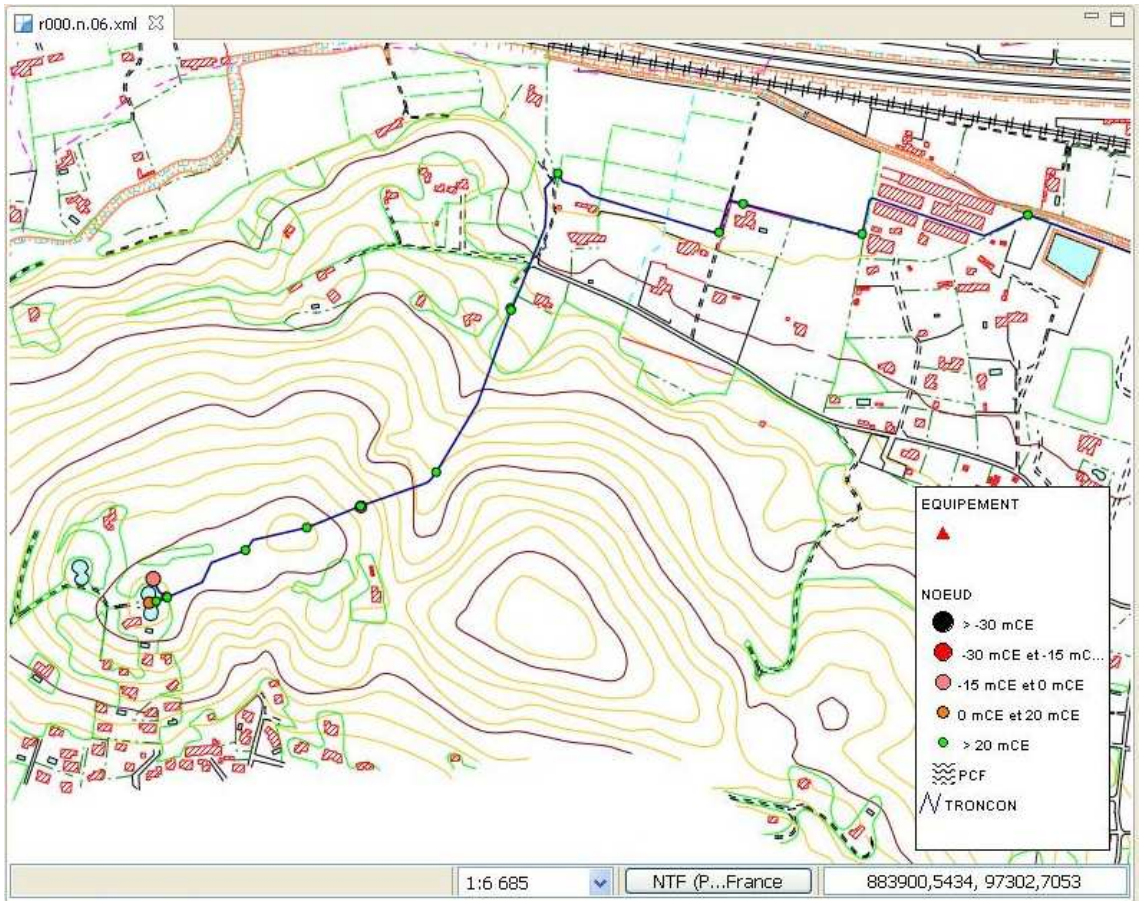

Figure 6.10: Pressure results visualization with IRMA.

7

Parallel execution using the Service Oriented Architectures and the Grid

Contents

This chapter presents two examples of distributed execution for validating IRMA's capability of harnessing resources on the Grid and Service Oriented Architectures.

The first example (Sec. 7.1) demonstrates how the AGOS Infrastructure (introduced in Sec. 1.5.2) allows to deploy IRMA as a service that takes a set of networks as input and automatically launches IRMA instances on the available resources.

The second example (Sec. 7.2) aims to reduce the execution time of a very computing intensive simulation mode (pump profiling), by implementing a distributed simulation using the *ProActive Parallel Suite* (introduced in Sec. 1.5.1) and deploying it on PACA Grid, a Cloud Infrastructure maintained by the french region "*Provence-Alpes-Côte d'Azur*".

7.1 IRMA distributed execution as AGOS Use case

Rewriting IRMA was done as part of the AGOS Project (Sec. 1.5.2), a project that uses a Service Oriented Architecture (SOA) Framework for deploying and running applications on the Grid. The AGOS Project consists basically in providing a Framework for implementing, managing, deploying and monitoring parallel services and the related Grid infrastructure. IRMA was chosen as use case of computing intensive application in order to validate the performance of the Parallel Services implemented by the AGOS Framework.

One of the main reasons for choosing Java as new programming language for IRMA is to have a portable solution and also to be able to easily integrate with the AGOS infrastructure. This section presents the work done with IRMA as use case for validating the AGOS framework and the performance results obtained in the Grid using the AGOS IRMA prototype.

The goal of this use case is to use the AGOS Infrastructure to deploy IRMA as a service that is able to perform several simulations simultaneously.

7.1.1 Overview of the AGOS Architecture

Figure 7.1 illustrates the different layers of the proposed architecture for the AGOS Service Level Management (SLM) Platform.

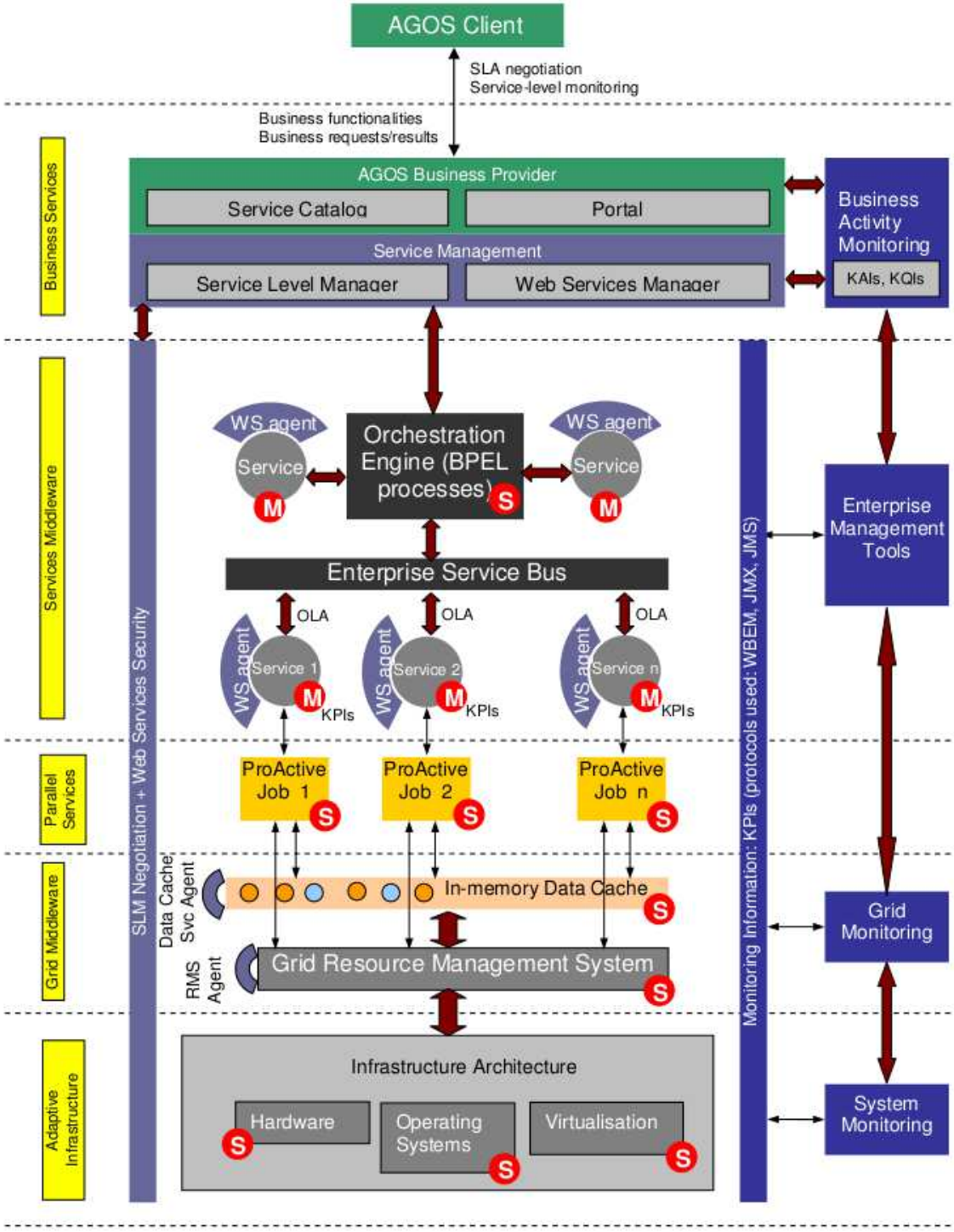

Figure 7.1: Architectural diagram of the proposed framework for the AGOS platform. Legend: (WS Agent) = Web Services Agent / (S) = Sensors installed at the module level, to relay performance information to the monitoring systems / (M) = Monitoring agent installed at the service level.

Business Services: the interaction of an AGOS Client with the AGOS platform is made through the **AGOS business provider** which is an interface to the client: it processes the

requests of the client, forwards the responses, manages interaction with the SLM module, etc.

Services middleware: the Service Management module is concerned with assuring the system-wide management of the web services exposed internally by AGOS, in terms of Web Services security (through a Web Services Manager) and negotiation of Service Level Agreements (SLAs) through the Service Level Manager. This layer employs "Web Services agents", which are modules attached to web services at every level of the platform, that act as proxies in:

- Formulating Service Level Requirements (SLR's) and negotiating SLAs.
- Assuring security policies are in place.
- Passing authentication between the boundaries of AGOS.

The **Business Process Execution Language (BPEL) Orchestration Engine** is responsible for executing and managing the business processes of AGOS. This layer will use "BPEL templates" plug-ins, which are pieces of generic BPEL code that are able to manage the calling and the pre- and post-interaction with a *ProActive* (Sec. 1.5.1) parallel service.

From an ITIL (*Information Technology Infrastructure Library*) point of view, the level of *ProActive* services could be interpreted as an "operational level", thus all SLAs with these services will be considered "Operational Level Agreements" (OLAs). *ProActive* services OLAs should expose "pre-negotiated" service-level terms. That means OLA templates had been already constructed by the service before runtime. These templates contain the available tasks, the set of service-level terms and Key Performance Indicators (KPIs) that the client then accept or deny ("take or leave"). This way, SOA clients can choose the *ProActive* service that best suits their runtime needs, and eventually "mix and match" between various levels of parallel services.

The **Parallel Services** and the **Grid Middleware** are both implemented by the *ProActive Parallel Suite*, that also handles the resource management of the **Adaptive Infrastructure**.

The **Monitoring** of services in the AGOS platform (in terms of security, SLAs, performance, etc), will be carried out platform-wise with the help of various industry-standard methods and protocols such as:

- Monitoring agents installed at the service level, that relay (and some of them even aggregate) the different KPIs associated with the respective service.
- Sensors installed at the module level that relay performance information to the monitoring systems.
- Monitoring protocols such as Web-Based Enterprise Management (WBEM).
- Technologies such as Java Management Extensions (JMX) and Java Message Service (JMS).

Monitoring data is collected, centralized and made available also through industry-standard technologies and tools, such as system monitoring tools, grid monitoring tools, enterprise managers, business activity monitoring platforms, etc.

For IRMA's use case, the main modules used are in the Service Middleware, Parallel Services and Grid Middleware layers. The implementation and usage of these modules are detailed in the next section.

7.1.2 AGOS use case implementation using Parallel Services

The goal of the AGOS use case is to transform the standalone Java version of IRMA into a service that is capable of automatically deploying parallel simulations on the AGOS Infrastructure. The standalone version processes an input file, simulates the *WDN* and generates an output file to write the results. In the use case, the user wants to simulate several networks and therefore his input is a folder containing all the files that describes the networks and simulation parameters.

The role of the AGOS Infrastructure is to offer a service that takes the input folder, splits the different networks in order to deploy IRMA instances in the existing resources and then

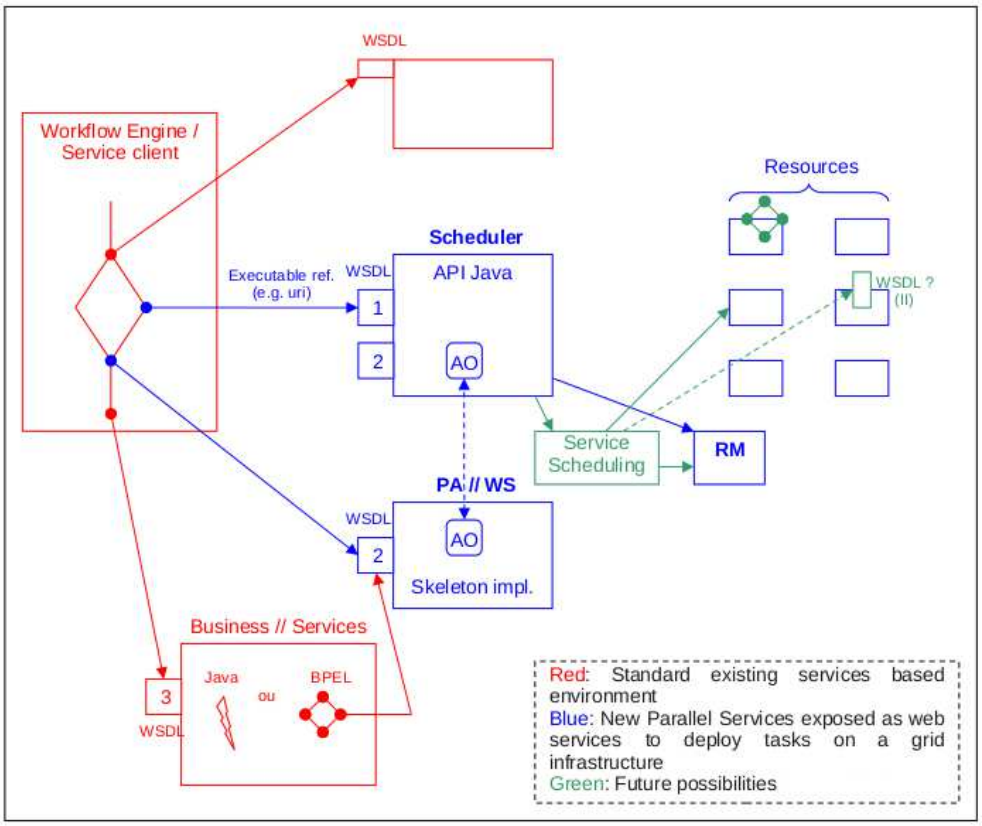

Figure 7.2: Diagram illustrating the AGOS infrastructure for deploying parallel services. Standard services based environment (not implemented by the AGOS Project) are represented in red. The blue components represent the elements that are used to deploy the IRMA use case: PA // WS (*ProActive* Parallel Web Services), *ProActive* Scheduler and Resource Manager (RM). The green part represent the future work that has been identified as a continuation of the AGOS project. (WSDL) = Web Services Description Language / (AO) = Active Object.

retrieve the results to finally give them back to the user. Figure 7.2 illustrates (in blue) the AGOS modules that has been used.

AGOS provides a set of Parallel Services and, in this use case, the module responsible for splitting the input folder is called **Parameter Sweeping**. Figure 7.3 illustrates how this module works:

1. User launches the application by giving the input data to Parallel Services.
2. Parameter Sweeping Service splits the input data into (*I1, I2, . . . , In*).
3. Parallel instances of IRMA are executed in the Grid using the different parameters.
4. Outputs (*O1, O2, . . . , On*) are gathered and given back to the user.

One last key element in this approach is the synchronization of files that is performed by using **ProActive Data Spaces API** [1]. On top of the *ProActive Parallel Suite*, Tomcat [tom11] is used for deploying the services and submitting the jobs.

7.1.3 Benchmark using all Maintenance department networks

This section presents IRMA's performance for the use case using the AGOS Infrastructure that has been deployed by *HP Sophia Antipolis* for developing and testing the AGOS prototype.

[1] http://proactive.inria.fr/release-doc/Programming/ReferenceManual/multiple_html/DataSpaces.html

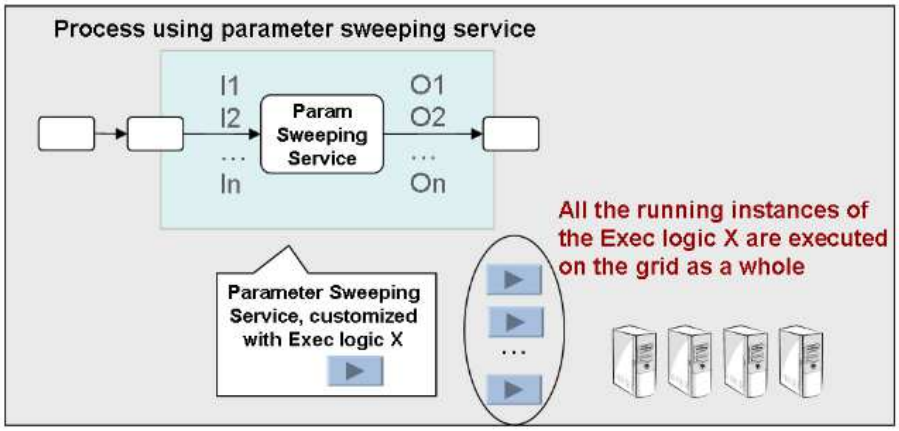

Figure 7.3: Diagram illustrating the parameter Sweeping service. Legend: (*I1, I2, . . . , In*) represent the input parameters, (*O1, O2, . . . , On*) represent the output of each parallel instance, (*Exec logic X*) represents the generic execution logic (user code), which in this case is implemented by the sequential version of IRMA.

Figure 7.4 shows the execution time when simulating a batch of 119 networks using 1 machine up to 4 machines from the Grid. Total sequential execution time is around 22 minutes and with 4 machines we reach 11 minutes, which is already close to the longest sequential execution time since the largest network runs in 7 minutes.

7.2 Distributed execution of pump profiling on the Grid

In order to perform the pump profiling (Sec. 3.6 and 4.6), several different scenarios need to be simulated. The total number of simulated scenarios is proportional to the number of hydrants present in the network and therefore the execution time can be prohibitive in case of large networks.

This section presents a solution for reducing the execution time by performing a distributed simulation of these scenarios on a Grid, using the *ProActive Parallel Suite* (without using the AGOS Platform).

7.2.1 Parallelization algorithm

The initialization phase is executed sequentially (as described in Sec. 4.7.1), since it is not computing intensive. The simulation phase is executed as described in Fig 7.5: instead of simulating sequentially each created scenario (Fig. 4.8), parallel tasks containing X scenarios to be simulated are created. This parameter X defines the granularity of each task and should be adjusted according to the network size and number of available resources where the tasks can be deployed.

The implementation with the *ProActive* Scheduler is straightforward: all the parallel tasks are aggregated into a job and then submitted to the *ProActive* Scheduler. The Scheduler is responsible for deploying the tasks on the Grid machines, for retrieving the results from each task and returning them back to the user machine where the simulation was launched.

7.2.2 Benchmarks on PACA Grid

This section presents the benchmarks performed on PACA Grid for validating the parallel pump profiling. PACA Grid is a set of machines (1 368 cores, 30 TB, 480 CUDA cores) accessible via

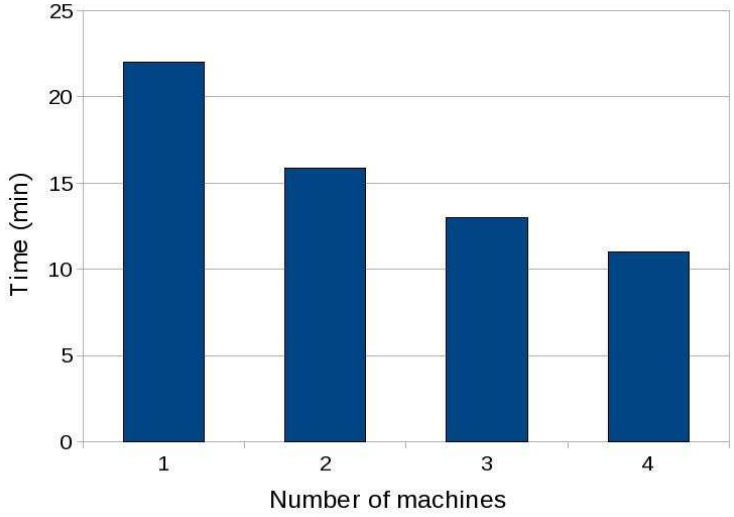

Figure 7.4: Execution time of 119 networks on the Grid using up to 4 machines.

ProActive Parallel Suite tools. The Cloud aggregates dedicated machines, running both Linux and Windows.

The network file used for this profiling models the downstream networks from *Bouteillère* (Fig. 7.6) and it contains:

- a total pipe length of 190 km.
- 2832 nodes.
- 5 tanks.
- 4 distinct probability zones (total of 28 classes of probability)
- 6 downstream pressure regulators.
- 4 pumping stations.
- 2 security valves.
- 2 differential head losses.

The total sequential execution time of this pump profiling using one machine from the Grid is: **25 hours and 23 minutes**. Figure 7.7 displays the obtained times by splitting the scenarios into different number of parallel tasks: from 18 up to 360 tasks (the initial number 18 comes from the 18 different objective flows values that are taken for this network and the remaining values are taken as multiples of 18).

The improvement in the performance drastically reduces the execution time: from **25 hours** sequentially to **13 minutes** using the Grid (360 parallel tasks). These test were repeated at least 5 times, the values represent typical execution times obtained when there are enough resources available to simultaneously run all the parallel tasks.

Figure 7.8 represents the evolution of the *Speedup* in order to measure how much a parallel execution is faster than a corresponding sequential version. The *Speedup* (S_p) is defined by the following formula:

$$S_p = \frac{T_1}{T_p}$$

where:
p is the number of processors,

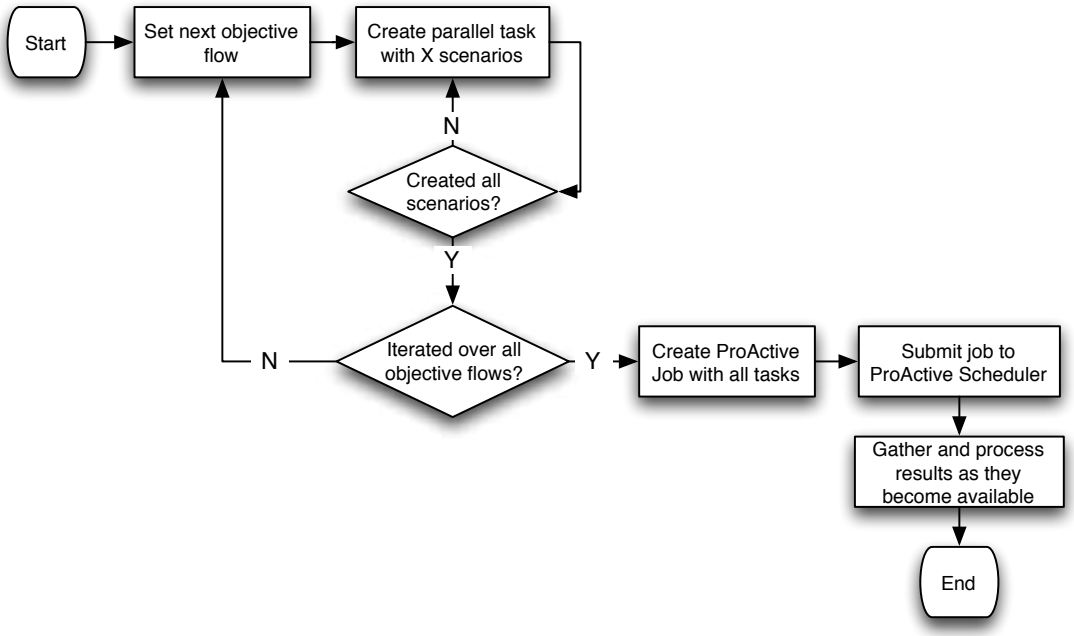

Figure 7.5: Flow diagram representing the parallelization algorithm for performing the pump profiling.

T_1 is the execution time of the sequential algorithm and
T_p is the execution time of the parallel algorithm with p processors.

The ideal environment for measuring the *Speedup* is dedicated environment made of homogeneous resources. Since the resources available in PACA Grid are heterogeneous and cannot be reserved, these figures do not give a precise representation of the *Speedup*. These numbers give approximative values by using as reference the performance of the machine where the sequential version was executed.

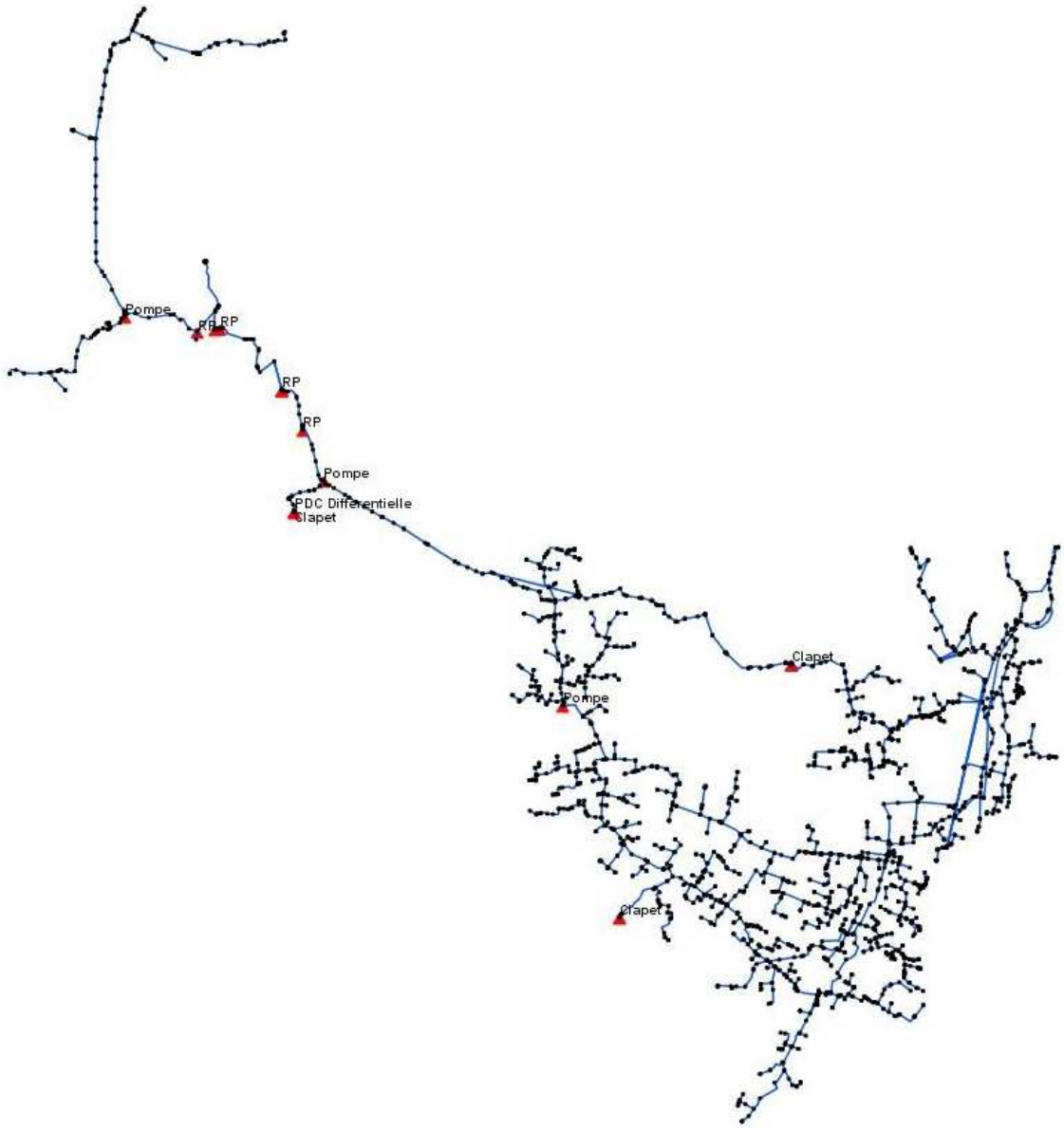

Figure 7.6: Overview of the downstream networks from *Bouteillère*.

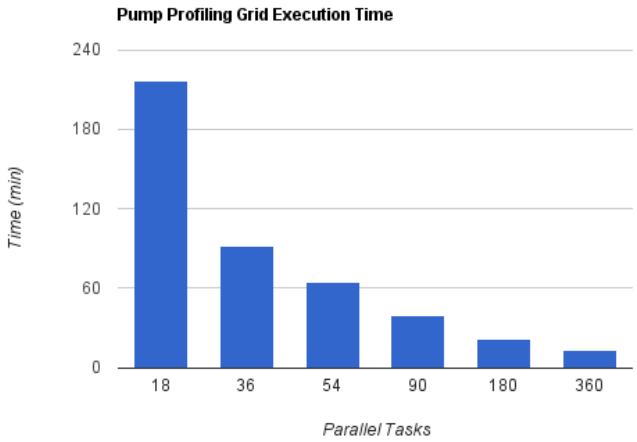

Figure 7.7: Pump profiling execution time of the downstream networks from *Bouteillère* in function of the number of parallel tasks.

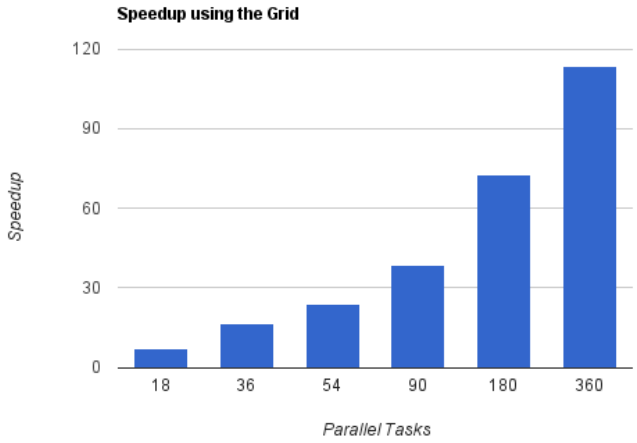

Figure 7.8: Obtained Speedup by number of parallel pump profiling tasks executed in parallel on PACA Grid.

8

Conclusion and Perspectives

IRMA has been developed over the last 30 years, with several persons been involved. As developers left the project, the SCP lost most of the knowledge behind IRMA. This happened due to different reasons: lack of documentation, the programming techniques used, and the limitations of the computational environments that were available at the time. This project has started with two main goals: the first was to redesign IRMA in order to fully understand it, and the second is to be able to solve the performance issues.

The first and more complex step was about understanding IRMA and this has been achieved by performing a detailed analysis of the ancient source code, along with a permanent consultancy of a hydraulic engineer. This work allowed to clean up the simulation model and to provide a more complete documentation about all the simulation processes.

The following step was to confront the results of all networks maintained by the SCP when simulated with the new and the former versions. This allowed first to have a second validation of the results being used by the SCP and then also allowed to find out and correct some flaws in the former implementation (that were mostly caused by programming faults).

This thesis explores the usage of the Java language for scientific high performance simulations, facing the challenges of using the modern languages for solving numerical problems and how to overvome this problem (by using a C++ interface in order to access an existing library). The final solution exploits the advantages of standard numerical tools, by using a scalable high performance FORTRAN library with minimal memory footprint, and the advantages of developing the new model using a modern object-oriented language.

The new high-level simulation model is modular and it can be understood more easily. Equipment specification has become much simpler and independent from the linear system construction. Code reading has been enormously simplified by eliminating global variables, `goto` calls and other deprecated programming practices that were used back then when IRMA was coded using punch cards.

The new Java implementation has been already successfully deployed in production and has replaced the FORTRAN version since July 2011. Many possibilities for improving user experience are now being exploited, text input file editing has been replaced by geographically-aware graphical editors and simulation data can be integrated with other GIS tools available at the company.

The performance issues have been solved mostly with sequential optimizations and the upper bound execution times, that before could take up to one week in some cases, have now been reduced to only a few minutes. The capability of performing distributed simulations has been demonstrated by deploying parallel simulations on the Grid using *ProActive Parallel Suite* and the AGOS Infrastructure. The parallel approach allowed to reduce the simulation time of the existing networks and gives new perspectives for dealing with the growth of the networks.

There are still improvements and tests that can be done in the simulation core, such as implementing another method for solving flow continuity and head loss equations. For example, the use of the gradient method proposed by Todini [TP88, Tod10] could allow to replace the FORTRAN solver with one of the Java iterative linear solver implementation and therefore it

could be possible to eliminate the C++/FORTRAN JNI bridge and have a pure Java solution.

Bibliography

[ABB+99] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.

[BBC+06] Laurent Baduel, Françoise Baude, Denis Caromel, Arnaud Contes, Fabrice Huet, Matthieu Morel, and Romain Quilici. *Grid Computing: Software Environments and Tools*, chapter Programming, Deploying, Composing, for the Grid. Springer-Verlag, January 2006.

[bla11a] Basic linear algebra subprograms (blas) website at netlib, 2011. http://www.netlib.org/blas/.

[bla11b] Linpack java translation, 2011. <http://www1.fpl.fs.fed.us/linear_algebra.html>.

[cem11] Cemagref website, 2011. http://www.cemagref.fr/.

[CGH+96] R. M. Corless, G. H. Gonnet, D. E. G. Hare, D. J. Jeffrey, and D. E. Knuth. On the lambert w function. In *ADVANCES IN COMPUTATIONAL MATHEMATICS*, pages 329–359, 1996.

[CJK97] Robert M. Corless, David J. Jeffrey, and Donald E. Knuth. A sequence of series for the lambert w function. In *ISSAC*, pages 197–204, 1997.

[Cla09] Didier Clamond. Efficient resolution of the colebrook equation. *Ind. Eng. Chem. Res.*, 47:3665–3671, 2009.

[Clé66] R. Clément. Calcul des débits dans les réseaux d'irrigation fonctionnant à la demande. In *La Houille Blanche*, volume 5, pages 553–576. Société Hydrotechnique de France, 1966.

[col11] Colt project website, 2011. <http://acs.lbl.gov/software/colt/>.

[CVZR05] P. B. Cheung, J. E. Van Zyl, and L. F. R. Reis. Extension epanet for pressure driven demand modeling in water distribution system. In *Computing and Control for the Water Industry*, volume 1, pages 311–316. Center for Water Systems, University of Exeter, 2005.

[dhi11] Dhi website, 2011. http://mikebydhi.com/.

[Dij59] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[DMBS79] J. J. Dongarra, C. B. Moler, J. R. Bunch, and G.W. Stewart. *LINPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1979.

[EGSS77] S.C Eisenstat, M. C. Gursky, M. H. Schultz, and A. H. Sherman. Yale sparse matrix package, ii. the nonsymmetric codes. Technical report, Department of Computer Science, Yale University, 1977.

[flo11] Flowmaster water applications website, 2011. http://www.flowmaster.com/water.html/.

[FN91] Roland W. Freund and Noël M. Nachtigal. Qmr: a quasi-minimal residual method for non-hermitian linear systems, 1991.

[g9511] Gnu g95 compiler website, 2011. <http://www.g95.org/>.

[geo11] Geotools website, 2011. http://geotools.org/.

[GSK08] O. Giustolisi, D.A. Savic, and Z. Kapelan. Pressure-driven demand and leakage simulation for water distribution networks. *Journal of Hydraulic Engineering*, 134(5):626–635, 2008.

[jam11] Java matrix (jama) website, 2011. <http://math.nist.gov/javanumerics/jama/>.

[jgr11] Jgrapht library website, 2011. <http://www.jgrapht.org/>.

[jts11] Jts topology suite website, 2011. http://tsusiatsoftware.net/jts/main.html.

[Lab88] Y. Labye. *Design and optimization of irrigation distribution networks*. Number 44-45 in FAO irrigation and drainage paper. Food and Agriculture Organization of the United Nations, 1988.

[LHKK79] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. Basic linear algebra subprograms for fortran usage. *ACM Trans. Math. Softw.*, 5:308–323, September 1979.

[LS00] N. Lamaddalena and J.A. Sagardoy. *Performance Analysis of On-demand Pressurized Irrigation Systems*. Food and agriculture organization of the United Nations, Rome, 2000.

[mik11] Mike urban website, 2011. http://mikebydhi.com/Products/Cities/MIKEURBAN.aspx.

[mtj11] Matrix toolkits java (mtj) website, 2011. <http://code.google.com/p/matrix-toolkits-java/>.

[net11] Netlib java website, 2011. <http://code.google.com/p/netlib-java/>.

[Pil95] O. Piller. *Modeling the behavior of a network - Hydraulic analysis and sampling procedures for parameter estimation*. PhD thesis, University of Bordeaux (PRES), Talence, France., 1995.

[por11] Porteau website, 2011. http://porteau.cemagref.fr/.

[pro11] Proactive website, 2011. <http://proactive.inria.fr/>.

[rcp11] Eclipse rich client platform, 2011. http://www.eclipse.org/home/categories/rcp.php.

[Saa03] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd edition, 2003.

[She94] Jonathan R Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical report, Pittsburgh, PA, USA, 1994.

[Tod10] Ezio Todini. Pressure-driven demand and leakage simulation for water distribution networks. *Journal of Hydroinformatics*, 13(2):167–180, 2010.

[tom11] Apache tomcat website, 2011. <http://tomcat.apache.org/>.

[TP88] E. Todini and S. Pilati. *A gradient algorithm for the analysis of pipe networks*, pages 1–20. Research Studies Press Ltd., Taunton, UK, 1988.

[udi11] Udig website, 2011. http://udig.refractions.net/.

[WRB86] J.H. Wilkinson, C. Reinsch, and F.L. Bauer. *Handbook for Automatic Computation: Linear Algebra*. Grundlehren Der Mathematischen Wissenschaften, Vol 186. Springer-Verlag, 1986.

[xer91] Logiciel xerxes-renfors, calcul économique optimisé d'un réseau d'irrigation sous pression, 1991. http://www-old.cemagref.fr/informations/Produits/Logiciels/ReseauxEau/XERXES.html.

High Performance Hydraulic Simulations on the Grid using Java and ProActive

Abstract

Optimization of water distribution is a crucial issue which has been targeted by many modeling tools. Useful models, implemented several decades ago, need to be updated and implemented in more powerful computing environments. This thesis presents the redesign of a legacy hydraulic simulation software (IRMA) written in FORTRAN that has been used for over 30 years by the Société du Canal de Provence in order to design and to maintain water distribution networks. IRMA was developed aiming mainly the treatment of irrigation networks – by using the Clément demand model and is now used to manage more than 6.000 km of piped networks. The growing complexity and size of networks requested to update IRMA and to rewrite the code by using modern tools and language (Java). This thesis presents IRMA's simulation model, including its head loss equations, linearization methods, topology analysis algorithms, equipments modeling and the linear system construction. Some new specific simulation features are presented: scenarios with probabilistic demands (Débit de Clément), pump profiling, pipe sizing, and pressure driven analysis. The new adopted solution for solving the linear system is described and a comparison with the available Java solvers is presented. The validation of results is achieved with a comparison between the previous FORTRAN results of all networks maintained by the Société du Canal de Provence and the values obtained using the new solution. A second validation is performed by comparing the results obtained from a standard and well-known simulation tool (EPANET). Regarding the performance of the new solution, a sequential benchmark comparing with the former FORTRAN version is presented. Finally, two use cases are presented in order to demonstrate the capability of executing distributed simulations in a Grid infrastructure, using the ProActive solution. The new solution has been already deployed in a production environment and demonstrates clearly its efficiency with a significant reduction of the computation time, an improved quality of results and a transparent integration with the company's modern software infrastructure (spatial databases).

Simulations hydrauliques d'haute performance dans la Grille avec Java et ProActive

Résumé

L'optimisation de la distribution de l'eau est un enjeu crucial qui a déjà été ciblé par de nombreux outils de modélisation. Des modèles utiles, implémentés il y a des décennies, ont besoin d'évoluer vers des formalismes et des environnements informatiques plus récents. Cette thèse présente la refonte d'un ancien logiciel de simulation hydraulique (IRMA) écrit en FORTRAN, qui a été utilisé depuis plus de 30 ans par la Société du Canal de Provence, afin de concevoir et maintenir les réseaux de distribution d'eau. IRMA a été développé visant principalement pour le traitement des réseaux d'irrigation - en utilisant le modèle probabiliste d'estimation de la demande de Clément - et il permet aujourd'hui de gérer plus de 6.000 km de réseaux d'eau sous pression. L'augmentation de la complexité et de la taille des réseaux met en évidence le besoin de moderniser IRMA et de le réécrire dans un langage plus actuel (Java). Cette thèse présente le modèle de simulation implémenté dans IRMA, y compris les équations de perte de charge, les méthodes de linéarisation, les algorithmes d'analyse de la topologie, la modélisation des équipements et la construction du système linéaire. Quelques nouveaux types de simulation sont présentés: la demande en pointe avec une estimation probabiliste de la consommation (débit de Clément), le dimensionnement de pompe (caractéristiques indicées), l'optimisation des diamètres des tuyaux, et la variation de consommation en fonction de la pression. La nouvelle solution adoptée pour résoudre le système linéaire est décrite et une comparaison avec les solveurs existants en Java est présentée. La validation des résultats est réalisée d'abord avec une comparaison entre les résultats obtenus avec l'ancienne version FORTRAN et la nouvelle solution, pour tous les réseaux maintenus par la Société du Canal de Provence. Une deuxième validation est effectuée en comparant des résultats obtenus à partir d'un outil de simulation standard et bien connu (EPANET). Concernant les performances de la nouvelle solution, des mesures séquentielles de temps sont présentées afin de les comparer avec l'ancienne version FORTRAN. Enfin, deux cas d'utilisation sont présentés afin de démontrer la capacité d'exécuter des simulations distribuées dans une infrastructure de grille, utilisant la solution ProActive. La nouvelle solution a déjà été déployée dans un environnement de production et démontre clairement son efficacité avec une réduction significative du temps de calcul, une amélioration de la qualité des résultats et une intégration facilitée dans le système d'information de la Société du Canal de Provence, notamment la base de données spatiales.