WatERP

Water Enhanced Resource Planning

"Where water supply meets demand"

GA number: 318603

WP 4: Decision Support System

D4.4: Inference and Simulation Engine Final Prototype

V1.0 10/06/2015

www.waterp-fp7.eu

Document Information

Project Number	318603	**Acronym**	WatERP
Full title	Water Enhanced Resource Planning “Where water supply meets demand”		
Project URL	**http://www.waterp-fp7.eu**		
Project officer	Grazyna Wojcieszko		

Deliverable	**Number**	4.4	**Title**	Inference and Simulation Engine Final Prototype
Work Package	**Number**	4	**Title**	Decision Support System

Date of delivery	**Contractual**	M32	**Actual**	M33
Nature	[✓ Prototype] Report ☐ Dissemination ☐ Other ☐			
Dissemination Level	[✓ Public] Consortium ☐			

Responsible Author	Gabriel Anzaldi Varas	**Email**	ganzaldi@bdigital.org
Partner	BDIGITAL	**Phone**	+34 93 553 45 40

Abstract (for dissemination)	This deliverable is focused on describing the latest development according with the WatERP-DSS. Indeed, this deliverable describes the latest prototypes of the RBR and the CBR inference engines towards managing the upper (ACA) and lower part (SWKA) of the water supply and distribution chain. The presented version of the RBR includes a fully implementation of the rules extracted from the water manager knowledge and new mechanisms to acquire the facts from sensor data and the knowledge base. Furthermore, the RBR also presents a fully implementation of the evaluation functionality defined for the DSS. The latest version of the CBR includes the improvement of the “*retrieval*” and “*Adaptation and Evaluation*” phase derived from the pilot’s implementation and the tests performed under the WP7. Furthermore, this version also includes an enhancement in the simulation procedure based on the application of rule-based inference towards generating more accurate alerts.
Key words	DSS, Decision Support System, inference engine, rule based reasoning, RBR, case based reasoning, CBR

Glossary of Terms

ACA - Agencia Catalana de l'Aigua

BRMS - Business Rules Management System

CBR - Case Based Reasoning

CPU - Central Processing Unit

CSV - Comma Separated Values

DDBB - Data Bases

DMS - Demand Management System

DRL - Drools Rule Language

DSS - Decision Support System

ISO - International Organization for Standardization

KB - Knowledge Base

KDD - Knowledge Discovery in Database

KNN - K-Nearest Neighbors

LHS - Left Hand Side

MAS - Multi Agent System

OGC - Open Geospatial Consortium

OMP - Open Management Platform

RAM - Random Access Memory

RBR - Rules Based Reasoning

RDF - Resource Description Framework

RHS - Right Hand Side

SOA - Service-Oriented Architecture

SQL - Structured Query Language

SWKA - Stadtwerke Karlsruhe - Wasserwerke

WDW - Water Data Warehouse

WPS - Web Processing Service

WPS - Web Processing Service

XML - Extensible Markup Language

Executive Summary

Deliverable 4.4 (D4.4) describes the current development status of the WatERP-DSS as an evolution of the initial prototype of the WatERP-DSS presented in D4.2 "*Inference and Simulation Engine 1st prototype*" and D4.3 "*Inference and Simulation Engine 2nd prototype*". The present deliverable is the last one of a series of deliverables that takes part of the WatERP vertical integration. In detail, this deliverable is focused on tuning the WatERP-DSS module towards pilot's implementation. Hence, the work described throughout this document is focused on depicting the full implementation of the Rules Based Reasoning (RBR) inference engine and the introduction of latest enhancements over the Case Based Reasoning (CBR).

In regards to the implementation of the RBR inference engine, the depicted architecture is an evolution of the first prototype presented in D4.2 "*Inference and Simulation Engine 1st prototype*". Over this initial architecture, the remaining rules gathered from the water resource managers have been implemented in the Drools Rules Language (DRL). Additionally, changes to the already implemented rules have been introduced in order to adapt to the recent changes of the ontology. Furthermore, it is also remarkable the implementation of a series of unitary tests to validate the correctness of the most complex rules. Regarding the general RBR architecture, several improvements have been performed. For example, the engine Alibaba (Sesame Community, 2015b) has been replaced by an ad-hoc ontology-to-Java adapter. This change has been motivated by continuous problems found using Alibaba. In order to permit the interoperability with the rest of the SOA-MAS architecture, the RBR has been published like a OGC WPS process following the guidelines presented in D2.3 "Open Interface Specification" (OSGeo-Live, 2011). As a result of all the described work, the RBR engine is fully implemented and integrated with the rest of the architecture.

In reference to the CBR, this last version includes changes in the "*retrieval*" phase towards solving the memory leakage found during the tests (see D7.5.2- "*Implementation of the Water DSS*"). Furthermore, minor changes have been performed concerning the definition of a configuration file that enables the customization of the CBR according to the demo-sites. Additionally, a rule-based inference engine has been included in order to enhance the simulation results with more robust alerts regarding water quality, infrastructure, safety and environmental constraints. As a result, the water manager will receive more and more specific information which will result on better decisions over the water supply network.

In a general perspective, this deliverable describes the development of a complete version of the RBR that takes advantage of the water manager expertise in order to generate efficient water allocation strategies for the upper part of the water supply and distribution chain (ACA demo-site). Furthermore, this deliverable also depicts the development of a CBR inference engine to achieve energy efficient water-work strategies to satisfy the expected demand in the lower part of the water supply and distribution chain (SWKA demo-site) based on the past-lived experience.

To understand this document the following deliverables have to be read.

Number	Title	Description
D1.4.4	Extension of taxonomy and ontology to the pilot's	This deliverable depicts the improvements performed onto the WatERP general ontology and taxonomy based on pilot's data information. In this version, the knowledge base has been expanded using the population methods that converts syntactic information into semantic models. Furthermore, the knowledge base has also been enriched with information regarding WP5 and OGC recommendation regarding data quality measurement in time series.
D2.3	Open Interface Specification	This document describes the analysis of the building blocks and the pilots' interfaces in order to understand the general open interface requirements for system integration and interoperability within the WatERP project. The guidelines to integrate a system in the WatERP framework are also described. It also includes the definition of the system integration road-map.
D4.3	Inference and Simulation Engine 1st Prototype	This deliverable depicts the current development state of the WatERP-Decision Support System (WatERP-DSS). It corresponds with the second prototype that has been focused on finishing the Case Based Reasoning (CBR) inference engine for the simulation and evaluation procedures. Furthermore, the CBR functionalities (evaluation and simulation) have been incorporated into an Open Geospatial Consortium –Web Processing Service (OGC-WPS) server towards being interoperable with the WatERP architecture.
D6.3	Open Management Platform 2nd Prototype	The Open Management Platform (OMP) represents the user interface to enable the entire water supply distribution chain to be managed from a holistic point of view. This deliverable describes the current development state and the new requirements and mock-ups derived from the design process.
D7.5.1	Implementation of the Water DSS	This deliverable describes the procedure that has been designed to validate the implementation of the Decision Support System in the pilots. A set of test cases based on the analysis of the WatERP pilot sites is presented in order to validate both inference engines used, namely the rule-based engine and the case-based engine. The preliminary results of the rule-based engine tests are also explained.
D7.5.2	Implementation of the Water DSS	The present deliverable is focused on the SWKA specific case test and the SWKA test based on a controlled scenario. The main aim of the controlled scenario test is to evaluate the CBR inference engine of the WatERP-DSS analyzing the generated pumping

		recommendations and extracting the first conclusions regarding the usage of the system resources (e.g. execution time, CPU and memory), the sense of the recommendations in order to provide energetic efficient pumping schedules and the DSS configuration in order to retrieve coherent recommendation with a low execution time. Additionally, the usage of WaterML 2.0 format to encapsulate the CBR-related data is explained and the modifications made to the logic model corresponding to the SWKA pilot site are documented.

Table of contents

Table of figures

Table of tables

List of Listing

1. Introduction

In the WatERP project, the decision-making process is achieved by the collaboration between WatERP Demand Management System (DMS), Decision Support System (DSS) and the Knowledge Base (KB) throughout the Service Oriented Architecture-Multi-Agent System (SOA-MAS) architecture. This architecture to intercommunicate the WatERP modules is based on the needs and the goals that the modules have in order to achieve the water manager necessities. This process is called orchestration and facilitates the communication between the modules using a capabilities (or functionalities) matching between the WatERP modules functionalities (services process) and the real location of the WatERP modules (hosted service).

In this architecture, the WatERP-DSS is in charge of generating recommendations and alerts that enhance the water manager's daily operations by avoiding water resource mismanagement and inefficient energy strategies throughout the water supply and distribution chain. Therefore, the WatERP-DSS main aim is to assist the water managers in their decision-making process towards enhancing water resource management and reducing energy inefficiencies inside the water supply and distribution chain. Moreover, the economic impact associated with this efficient management is also improved (e.g. reducing the water distribution costs) by the usage of this kind of intelligent systems.

These mentioned objectives are strongly aligned with the needs found on ACA and SWKA pilots (see Deliverable 4.1 Section 3 on page 32). On the one hand, in ACA case the main awareness is the scarcity so, the main operations performed are focused on fulfilling the expected demand without compromising the water resources (see Deliverable 4.1 in Section 3.1 on page 33). On the other hand, the SWKA pilot-case has a huge amount of water and the main objective is to reduce the waste of energy by selecting suitable pump schedules that permit to distribute the water in an energetically efficient way (see Deliverable 4.1 in Section 3.2 on page 68). Additionally, the SWKA pilot water distribution also needs to accomplish the water distribution policies that permit to assure the integrity of the network (e.g. pressure maintenance, tank feeds, etc.) and also needs to satisfy some water quality constraints (e.g. pump flows, pump concurrency).

WatERP-DSS accomplishes ACA and SWKA decisional needs by using a module called Inference Engine. The Inference Engine is responsible of generating the above mentioned recommendations and alerts. This module consists of a RBR and CBR (see Figure 1). The Inference Engine has to provide the functionalities of evaluation and simulation (see Deliverable 4.1 in Section 4 on page 83). On the one hand, the evaluation functionality is aimed at providing the user with information about suitable water management and distribution strategies to be adopted for the expected scenario (one, two or three days after). Hence, the expected output for the evaluation is a water resource allocation strategy (ACA case) and efficient water distribution management strategy (SWKA case). On the other hand, the simulation functionality permits to assess a state of the water supply and demand chain in order to detect water mismanagement and energetic inefficiencies given an operation policy or water manager experience. As output for the simulation, decisional recommendations are produced based on a tentative water resource allocation strategy or a water distribution strategy (pump scheduling) regarding certain scenario definition. Therefore, the Inference Engine module performs the reasoning process based on the water manager experience combined with policy constraints and current state of the systems. The strategy to be followed for the Inference Engine development during the WatERP-DSS development was explained in previous deliverable (see Deliverable 4.2 on page 13).

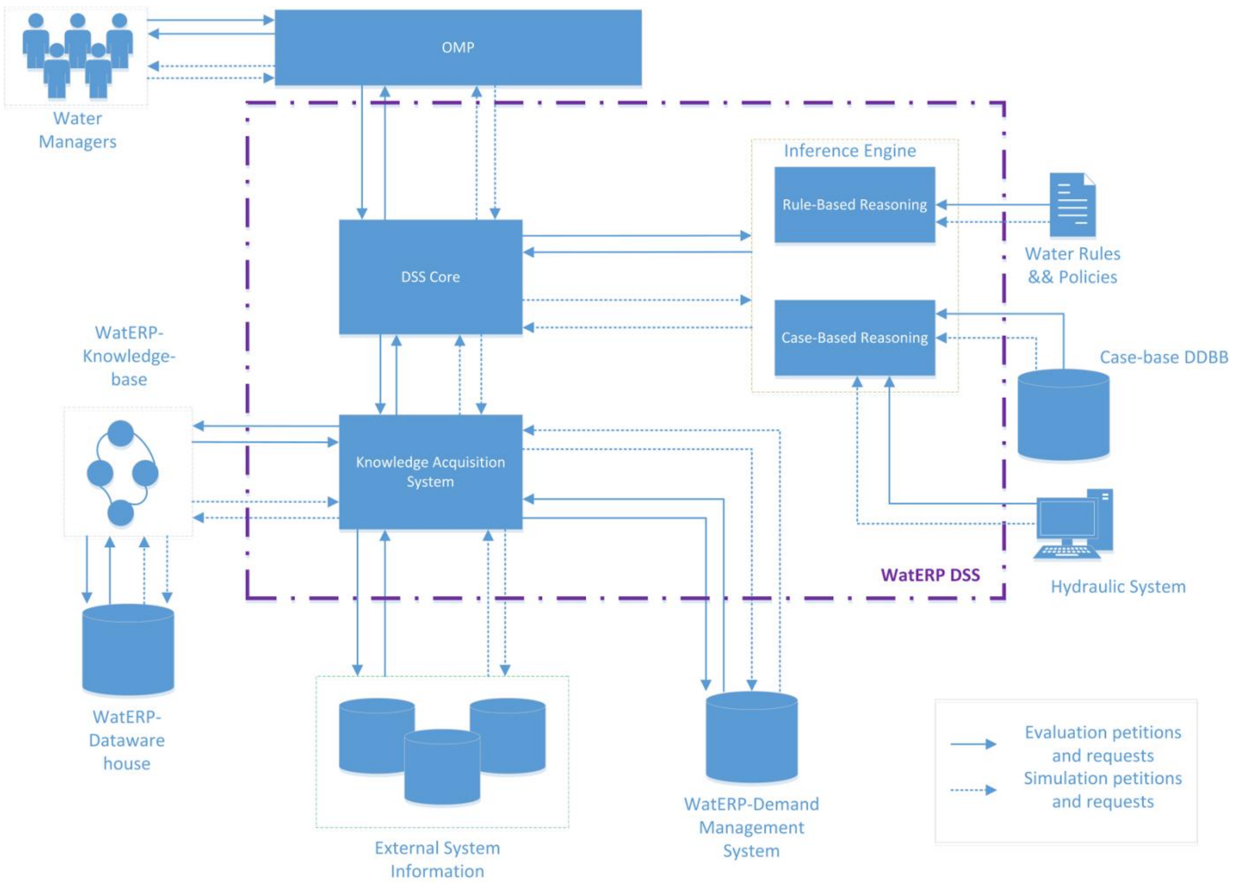

Figure 1"Waterp-DSS schema"

In Figure 1 is shown a schema of the WatERP-DSS module. The DSS core uses the Inference Engine to generate the decisional information, which later is delivered to the water manager by means of the OMP module. As previously explained, the Inference Engine consists of two modules: RBR and CBR. The Inference Engine makes use of some external tools: (i) Water Rules and Policies where the expert knowledge has been introduced, (ii) Case-base DDBB where historical cases are stored, and (iii) an Hydraulic System simulator called EPANET (Rossman, 2000).

Focused on the inference engines, the main goal of the RBR is to perform evaluation and simulation processes by means of expert knowledge (logic part) over a set of observations which represent the water supply and distribution chain (model part). By separating model and logic, the RBR achieves a new abstraction layer, where the expert knowledge is stored in a Rules Repository (where the knowledge is centralized) and the observations (specific information from the scenario) are stored in a database of facts. Thus, both logic and model are independent entities, which can be modified separately at any time. In spite of this independency, when rules are combined with facts (observations) they achieve the same conclusions that an expert would reach. Regarding the RBR implementation, Drools Inference Engine has been used. Drools (Bali, 2013) offers a Java-based framework to configure an inference engine by modelling the expert knowledge through the Rules Repository. Furthermore, rules are applied over on an instantiated model representation that is built using Java classes (see Deliverable 4.2 in Section 3.1 on page 20). Drools is one of the most used Business Rules Management System (BRMS) and it can be easily integrated in a Java running environment system. Moreover, it provides a modified version of the RETE algorithm (Forgy, 1982) called RETE-OO (Sottara, Mello, & Proctor, 2010) which enhances the performance of pattern matching in Object Oriented (OO) programming languages. This enhancement provides greater scalability while speeds up the overall performance.

The main goal of the CBR is to provide a feasible solution for a given problem based on the study of previous experience. One of the advantages of CBR systems is that they improve their performance, becoming more efficient, by recalling old solutions given to similar problems and adapting them to fit the new problems. In this way, the CBR does not have to solve new problems from scratch. The memorization of past problems/episodes is integrated with the problem-solving process, which thus requires the access to past experience to improve the system's performance. Additionally, case-based reasoners become more competent during their functioning over time, so that they can derive better solutions when faced with equally or less familiar situations because they do not repeat the same mistakes (learning process). The CBR implementation follows the strategy described in Deliverable 4.2 in Section 4.2 on page 40. To tackle the problem of finding a similar previous case the CBR consists of two parts: (i) usage of a KDD procedure to create a similarity vector that permit to compare the cases without consuming so much resources; (ii) a KNN algorithm which search for a set of most similar cases according with the new defined problematic. The adaptation and further evaluation of the solution are performed by the water distribution manager through the Open Management Platform (OMP) (see Figure 1). Thus, the water distribution manager interacts with the OMP by modifying the CBR pumping scheduling towards a more suitable strategy (case adaptation). Once the water distribution manager finishes with the modification, he/she simulates the modifications until a correct strategy is achieved. At this moment, the CBR stores the solution in the database of cases, closing the learning cycle.

The present deliverable is focused on depicting the final versions of both the RBR and the CBR. Special emphasis is given to the RBR due to the previous deliverable focus on the CBR development.

Therefore, the present deliverable will describe the RBR development strategy to materialize, in form of rules, the water manager expertise (see Section 2). For that, the RBR architecture is explained. This explanation is accompanied with the main improvements made to acquire facts from the scenario through the SOA-MAS architecture. Referring the rules, the usage of templates and Java-programming has permitted to translate water manager expertise into DRL. In order to demonstrate the correct execution of the RBR, the Section 2 will depict some initial testing performed over this inference engine. Regarding the CBR (see Section3), main improvements in performance referring the memory leakage are explained. Moreover, the CBR simulation functionality has been improved by a rule-based checking of the main water distribution constraints according to the existing infrastructural and policy/environmental constraints. This latter aspect has been included due to the fact that hydraulic engine normally forces the accomplishment of these rules during simulation, aspect that can derive in internal changes regarding the introduced pumping scheduling by the water manager. In the last part of the document, main conclusions about the WatERP-DSS and specifically the inference engines have been depicted (see Section 4).

2. Rule-Based Reasoning (RBR)

One of the main advantages of using a RBR as an Inference Engine is that it represents the expert knowledge using a declarative language instead of an imperative language. Declaratives languages rely on the separation between the intrinsic domain knowledge from the domain information, facilitating the knowledge discovery when specific domain information accomplishes the domain knowledge conditions (rule conditions). Improvements against imperative languages (driven execution of rules) are motivated by a reduced complexity in decisional process modelling. Thus, main advantages of using a rule based system are: (i) separation between decisional process and domain information; (ii) decisional process is guided by the rules (what to do) instead of lead by them (how to do it); (iii) good performance parameters in terms of execution speed and scalability achieved by the RETE algorithm; (iv) the knowledge is centralised using a Rules Repository. Derived from the main RBR characteristics, the RBR provides a new layer of abstraction which splits data model and expert knowledge into sets of rules and facts. Rules represent water manager experience and facilitate the knowledge inference by the activation of other rules and/or the achievement of conclusions. As contrary, facts instantiate the domain model with water resource information (e.g. reservoir levels, environmental parameters, desalination plants performance, etc.) or water distribution information (e.g. pump flows, pump pressure, tanks levels, etc.). This separation provides a great functionality as, while the model can change over the time, rules can be preserved, and vice versa, the expert knowledge can grow by increasing the Rules Repository without the necessity of any model modification.

Furthermore, the use of a rule-based reasoner can be seen as the application of the expert knowledge over an instantiation of the data model. According to this, the rule engine follows the same inference process that an expert would follow, and achieves the same conclusions over the facts which represent a given model state. In the current case, these conclusions are new knowledge that the water managers receive from the expertise coded in the rules repository.

Regarding the ACA pilot-case, where water resource management is mainly performed using the Inference Engine, water manager decisional expertise has been modelled by a set of rules that describes the process used by the water manager in the daily operations. Therefore, the main output of this inference engine are suitable recommendations to manage water resources in an efficient way by considering the water manager experience with the objective of accomplishing the expected demand without compromising the water resources.

2.1 RBR enhancements performed from pilot's deployment and integration process

In this section, all the changes made to the RBR procedure described in the Deliverable 4.2 Section 3 on page 20 are presented. As described in the mentioned deliverable, the RBR procedure (Figure 2) had two major inputs: an ontology describing the ACA logical model ("FeaturesOfInterest", "WaterResources", "Observations", etc.) and a set of rules to apply. The ontology was published using

Sesame, a Java framework for processing and handling RDF data (Sesame Community, 2015a). Using an ontology query language called SPARQL (Prud'hommeaux & Seaborne, 2008), a series of queries to a local Sesame repository containing the ontology were performed in order to extract the needed information that also was obtained and translated to Java classes. This translation was automatically done by Alibaba. The rules that are applied to the case were inserted into the Rules Repository. These rules were divided in literal rules (rules that literally described the expert knowledge) and template rules (generalization of multiple rules that describes the expert knowledge and needs to be particularized before inserting them into the Rules Repository). Once the Working Memory was populated with the facts coming from the ontology and the Rules Repository was filled with rules, the rules evaluation procedure started. During this procedure, Drools was in charge of, in an iterative manner, selecting the rules to be executed based on the status of the Working Memory, resolving the conflicts that might occur and firing the rules. The execution of rules could lead to new knowledge inserted in form of facts into the Working Memory, which could result in execution of other rules. Once this iterative process ended, a water allocation strategy was achieved (or not).

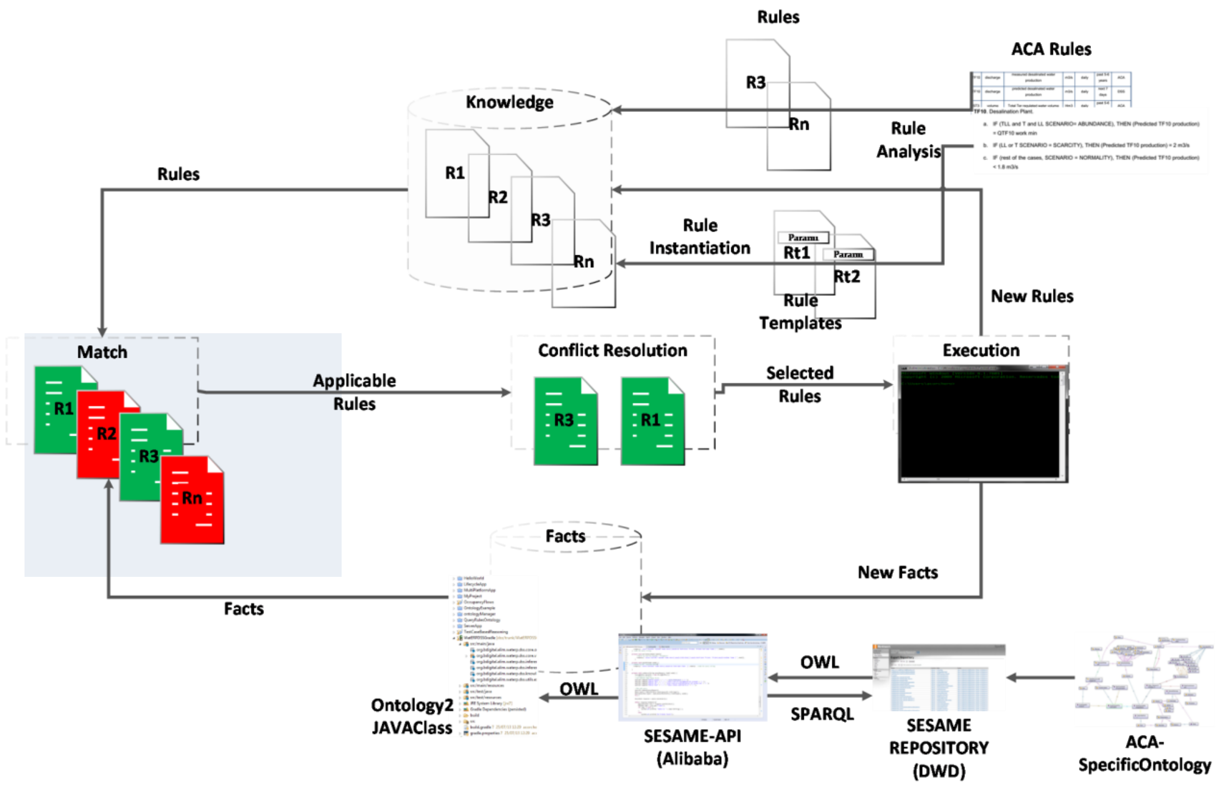

Figure 2 "RBR procedure"

Continuing to the work regarding the RBR application to the ACA case (presented in Deliverable 4.2) the current deliverable describes the progress made during this last year over this inference engine. From a general perspective, the work carried out has been focused on: (i) finishing the implementation of the rules; (ii) changing the working memory population procedure; (iii) defining the interfaces for a service-oriented architecture and (iv) implementing tests to validate the results of the RBR.

Finishing the Implementation of the Rules

The work regarding the implementation of the domain expert rules into DRL rules has been principally divided into two tasks: adaptation of existing rules and implementation of new rules.

- **Adaptation of existing rules:** at the time when Deliverable 4.2 was presented, about of 90% of the rules that modelled the expert knowledge from the ACA case where already implemented. Nevertheless, due to the changes made to the ontology (Deliverable 1.4.4) some changes in the rules implementations have been performed. These changes in the ontology have been motivated by the informational requirements generated during the integration of the WatERP modules in the platform, the introduction of a semantic model to measure quality in time series, the continuous monitoring of semantic water-standards models, and changes generated by the ontology testing in WP7 (see Deliverable 1.4.4). The vast majority of changes in the ontology that affect the rules already defined are focused on the change performed about the names of class properties and object properties. For instance, the change in the notation used to express that an *"Observation"* belongs to a *"FeatureOfInterest"* or *"WaterResource"* or *"WaterResourceManagement"* formerly noted as *"isObservedByFoI"* to the new notation *"isObservationOf"* forced the rewriting of part of the rules and the associated tests.
- **Implementation of new rules:** during this period, the remaining 10% of rules have been implemented along with their corresponding unitary test cases (when considered necessary). The main efforts put on this task have been focused on implementing the rules from paragraph "*F.- Defining resources priorities"* found in the Deliverable 4.1 section 3.1.4.3 page 61. This paragraph contains a series of tables that describe the process that has to be followed when allocating the water resources to fulfill the water demand and reduce the excess of water. This process is defined by a series of actions (Extract the minimum from *"TF8"*, extract the maximum possible from *"ST1"*, etc.) that should be performed on a specific point in time (today, tomorrow or two days after) according to the defined priorities. For each action, there is a series of related rules that should be taken into consideration when executing the rule in order to assure that the provided solution is valid.

Priority Order	***For all SCENARIOS (A, N or S)***	***Temporal Scale****	***Related rules***
	IF PLANTS ARE WORKING	(days)	
1	Ter min TF8min	1	C7.c
QTF8 work min = 2 m3/s (minimum production when the transfer is working)			

Table 1 "Sample rule"

In that sense, Table 1 shows a sample rule. This rule, with a priority of 1, states that the output from *"TF8"* should be set at its minimum. Because of the temporal scale (1 day), this action is proposed to be taken the next day (because it takes one day for the water coming from *"TF8"* to arrive to *"Si2"*). Finally, when executing this rule, the rule C7.c should also be taken into consideration. Particularly, the rule ("*C7.c*") sets the value for the minimum production of *"TF8"* when it is working (*2m^3/s*).

At certain points in water allocation process, the demand satisfied at that time is checked in order to decide whether the evaluation of the rules should keep on or stop. Those checkpoints compare the amount of demand satisfied with the total demand and stop the execution of the allocation process if all the demand has been satisfied.

Priority Order	***For all SCENARIOS (A, N or S)***	***Temporal Scale****	***Related rules***
IF PLANTS ARE WORKING		(days)	
D<R(1+2+3+4)?			
YES	Aquifer ST4 = ST4 work min -> END	2	C8, E20, G.1.27, G1.28.d
NO			
5	Aquifer	2	G.1.27

Table 2 "Checkpoint example"

For instance, Table 2 shows an example of checkpoint. This particular example evaluates the demand satisfied from the execution of rules 1, 2, 3 and 4. If the demand is satisfied, *"ST4"* will be set to its minimum in two days time and the water allocation process is considered to be finished. If the demand has not been satisfied yet, the production of *"ST4"* in two days time will be set to its maximum to fulfill the demand.

There are other singular points in the water allocation process where, after evaluating a certain resource allocation strategy and observing that the water demand has not been met, a decision to start over, following a new strategy, is taken.

Priority Order	***IF LL SCENARIO = A***	***Temporal Scale****	***Related rules***

IF PLANTS ARE WORKING		(days)	
6	New Priority Order = 1 - 2 - 4 - 5 , taking into account the same rules described for those priorities		
	Discharge from reservoirs	0	G.1.28.a, G.1.28.d, E18

Table 3 "Example of change in the resource allocation strategy"

For instance, Table 3 shows an example of change in the resource allocation strategy. After having tried to meet the demand with the resources allocated by rules 1, 2, 3, 4 and 5, and having come to the conclusion that the water demand is not met; a new resource allocation strategy is proposed. The new strategy consists on fulfilling as much demand as possible with the rules 1, 2, 4 and 5 and trying to fulfil the remaining unsatisfied demand with the water extracted from the reservoirs.

The iterative process modeled by the rules found under *"Paragraph F"* is based on evaluating multiple water resource allocation strategies in order to meet the demand. Each of these approaches requires for the execution of multiple rules in a sequential manner in order to decide whether the chosen allocation strategy is able to fulfill the water demand or not.

To accomplish the sequential execution of the rules, the concept of *"ActivationPattern"* has been introduced. An *"ActivationPattern"* is a mechanism to instruct Drools engine to evaluate the rules following a certain order. It is defined as a sequence of collections of disjunctions of rules. That is, for each step of the activation pattern, a list of rules that can be executed during that step is defined. The decision of which of those rules will be executed depends on the accomplishment of the conditions defined in their corresponding LHS and salience.

```
1   RulesUtils.setActivationPattern(WM,
2          "F.0",
3          "F.0.1",
4          "F.0.2",
5          "F.0.3",
6          "F.0.4",
7          new String[]{"F.0.5.1","F.0.5.2"},
8          new String[]{"F.0.5.2.1","F.23.6.1","F.24.6.1","F.25.6.1","F.23.b"}
9   );
```

Listing 1"Activation pattern"

Listing 1 shows an example of how an *"ActivationPattern"* is set. In this pattern, composed by seven steps, rules *"F.0"*, *"F.0.1"*, *"F.0.2"*, *"F.0.3"* and *"F.0.4"* (lines 2 to 6) will be executed sequentially (one after the other). Then, the rule *"F.0.5.1"* or *"F.0.5.2"* will be executed (line 7). In this particular case, *"F.0.5.1"* stops the evaluation of the resource allocation strategy as the demand has been satisfied, whilst *"F.0.5.2"* continues the allocation of water resources as the demand has not been satisfied yet.

The accomplishment of the activation pattern is achieved by the introduction of special facts called "*ActivationFlag*". An "*ActivationFlag*" is a special fact that instructs the Drools rules analyser to execute a particular rule. The rules that take part of an "*ActivationPattern*" require an "*ActivationFlag*" with its name to be present inside the Working Memory. This restriction is expressed in the LHS of the rule.

```
1   rule "F.0.5.2.1"
2   when
3      $flag:ActivationFlag(ruleName == "F.0.5.2.1")
4      ...
5   then
6      insert(new ActivationFlag("END"));
7   end
```

Listing 2"A fragment of a rule that requires an "ActivationFlag""

Hence, the Listing 2 shows a fragment of a rule (called "*F.0.5.2.1*") that requires an "*ActivationFlag*" with its name in the Working Memory (line 3) in order to be eligible to be executed.

Once a rule that belongs to an "*ActivationPattern*" has been executed, it must signal this condition to the engine in order to proceed with the next group of rules from the "*ActivationPattern*". To this extent, an special method has been defined. This method is named "*signalRuleExecuted*" and indicates to the engine that the current rule has ended its execution. Thus, the following rules can be evaluated. Upon reception of that call, the system retracts any "*ActivationFlag*" from the Working Memory and proceeds to add an "*ActivationFlag*" for each of the rules contained in the next step of the "*ActivationPattern*".

As an example, Figure 3 shows the procedure followed when an "*ActivationPattern*" is set. For the sake of the example, let us suppose that the activation pattern is like the one described in Listing 3. This "*ActivationPattern*" is composed of an arbitrary number of *N* steps having the 3 first ones the rules "*Rule1*";"*Rule2*","*Rule3*" and "*Rule4*" respectively. Let us assume that those rules are of the form described in Listing 4 where variable *X* denotes the number of the rule and that no other rule exists in the Rules Repository.

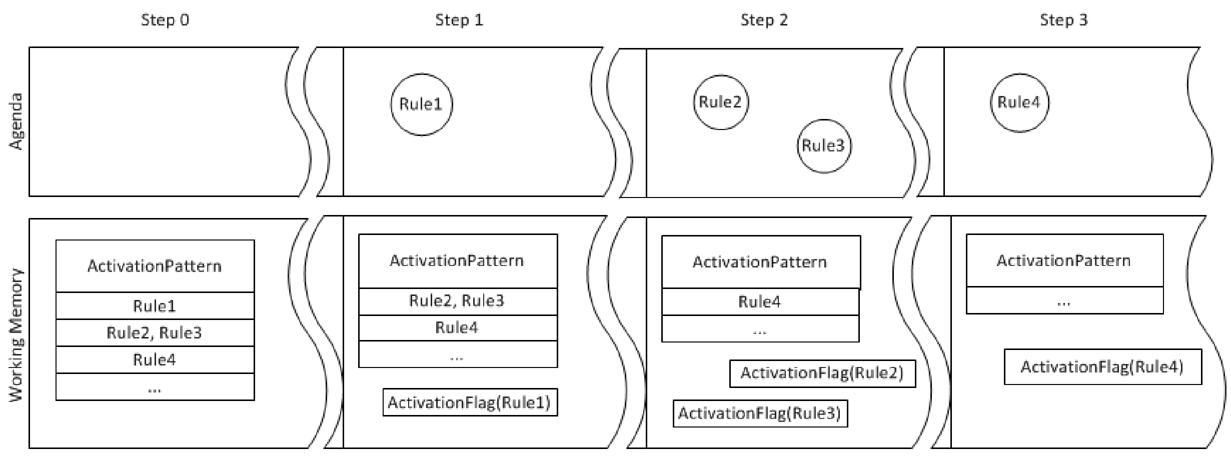

Figure 3 "Procedure followed when an "ActivationPattern" is set"

```
RulesUtils.setActivationPattern(WM,
   "Rule1",
   new String[]{"Rule2","Rule3"},
   "Rule4",
    ...
);
```

Listing 3"An example of activation pattern"

```
rule "RuleX"
when
   $flag:ActivationFlag(ruleName == "RuleX")
   ...
then
   ...
   RulesUtils.signalRuleExecuted($flag,WM);
end
```

Listing 4"Example of rule of the activation pattern"

Based on these assumptions, Figure 3 shows the steps followed during the execution of an *"ActivationPattern".*

Step 0: The *"ActivationPattern"* is inserted into the Working Memory, automatically an activation flag for each rule that compose the first group of rules (only *"Rule1"* in this case) is added to the Working Memory. This causes *"Rule1"* to be inserted into the Drools *"Agenda".*

Step 1: *"Rule1"* is selected from the *"Agenda"* to be executed due to an activation flag for this rule exists. After the end of the execution of the rule, the system is notified of this event and automatically removes the *"ActivationFlag"* for *"Rule1"*. Furthermore, the system removes the top element from the *"ActivationPattern"* and adds the corresponding *"ActivationFlags"* into the Working Memory. In this occasion, two *"ActivationFlags"* are added at the same time *("ActivationFlags"* corresponding to *"Rule2"* and *"Rule3"*) because of the *"ActivationPattern"* contains two rules.

Step 2: Two rules are placed inside the Drools Agenda (*"Rule2"* and *"Rule3"*) given that their LHS is completely fulfilled. Then, Drools rules engine decides which of the two rules is executed. At the end of the execution of one of both rules (*"Rule3"* for example), the system is signaled of that event and proceeds to update the Working Memory. All the *"ActivationFlags"* are then removed (*"Activation Flags"* for *"Rule2"* and *"Rule3"* in this case). This impossibility *"Rule2"* and *"Rule3"* to be fired, as none of their LHS is fulfilled anymore. Then, the *"ActivationPattern"* is updated, removing the collection of rules on top and adding an *"ActivationFlag"* for each rule in that collection (only *"Rule4"* this time).

Step 3: The process of executing the *"ActivationPattern"* continues until this pattern contains no more rules are ready to be executed.

Although the “*ActivationPattern*” technique described above enforces the ordered execution of the rules that form the “*ActivationPattern*”, the execution of the rest of the rules that might be part of the Rules Repository is not restricted at all. That is, it might be possible that, during the execution of an “*ActivationPattern*”, a rule outside this pattern is activated because its LHS is fulfilled.

As described before, the water allocation procedure is an iterative process during which many allocation strategies might be evaluated until the water demand is fulfilled. From a general perspective, the defined allocation procedure starts with a plan that minimizes the abstraction from *“TF8”*, *“TF10”*, *“ST1”*, *“ST2”* and *“ST4”*. If this procedure is unable to provide enough water to fulfill the demand, a new approach is taken where *“ST4”* extraction is maximized. If this new approach is still unable to fulfill the demand, another approach is taken where discharge from *“ST1”* and *“ST2”* is incremented. Each of these iterations modifies the predictions of discharge, water level, water input, etc., of various elements that conform the water distribution network. Given the fact that these values are needed for performing calculations to decide the amount of water demand satisfied, it is required that, for each new iteration of the water allocation procedure, those values are reset to the values prior to the start of the iteration. To solve this problem, three different approaches where evaluated: (i) restarting the execution of the rules evaluation, (ii) making a backup of all the elements of the Working Memory; or (iii) making a backup of the values modified.

“Restarting the execution of the rules evaluation” has been discarded given that it is inefficient in terms of time spent. Given that prior to the execution of the rules that implement the water allocation procedure many other rules have been fired, restarting the whole execution would require to re-execute all those rules again, leading to an increase of the total execution time. Another option evaluated consisted on “*making a backup of all the elements stored in the Working Memory*” at the beginning of the execution of the first allocation strategy and then, if needed, restoring the elements from this backup to try another approach. This approach was discarded given that it requires to store a lot of information (all the facts contained in the Working Memory) when only few were actually modified. The last evaluated approach, and the one finally selected, consists on making a backup only of the modified “*TimeValuePairs*” during the execution of a resource allocation strategy. To accomplish this, the class “*ObservationValuesStorage*” has been defined (see Listing 5).

```java
public class ObservationValuesStorage {
    public Map<String, Double> values;
    public ObservationValuesStorage()
    {
        values = new HashMap<String, Double>();
    }
}
```

Listing 5"Definition of "ObervationValuesStorage" class"

As depicted in Listing 5, this class has only a member named *values* of type "*Map<String,Double>*" where the key ("*String*") contains an id of a "*timeValuePair*" and the value ("*Double*") contains the value associated with that "*TimeValuePair*". Then, a single instance of this class is inserted into the Working Memory and is then accessible for the rules to store values in it.

```
 1   rule "F.0.1"
 2   when
 3       $flag:ActivationFlag(ruleName == "F.0.1")

 4       $DemandSatisfiedObservation: Observation(id ==
     "DemandSatisfiedObservation")
 5       $TF8 : WaterResource(id == "TF8")

 6       $storage: ObservationValuesStorage()
 7   then
 8       //Set TF8Q in one day to be TF8 min
 9       TimeValuePair TF8QTVP =
     RulesUtils.getObservationTVPByDate($TF8,DISCHARGE,PREDICTION_PROCEDURE,D1_DA
     TE);
10       Double TF8MinValue =
     RulesUtils.getObservationTVPByDate($TF8,DISCHARGE_WORKING_MIN,MANUAL_METHOD,
     D1_DATE).getValue();
11       //store the value of TF8Q
12       $storage.values.put(TF8QTVP.getId(),TF8QTVP.getValue());
13       modify(TF8QTVP){setValue(TF8MinValue)}

14       //Increment the demand satisfied
15       TimeValuePair DemandSatisfiedTVP =
     RulesUtils.getTVPByDate((TimeSeriesObservation)$DemandSatisfiedObservation.g
     etObservationResult(),D2_DATE);
16
     modify(DemandSatisfiedTVP){setValue(DemandSatisfiedTVP.getValue()+TF8MinValu
     e)}

17   RulesUtils.signalRuleExecuted($flag,WM);
18   end
```

Listing 6"Example of a rule that stores an observation value"

As an example, Listing 6 shows a rule that stores an observation value as a fact that later in the execution is restored. In this rule, *"TF8"* discharge for day 1 is set to its minimum value and the demand satisfied is incremented with the same value. Given that some prior calculations established the value of discharge of *"TF8"* at a certain value before the execution of this rule, and that it might be necessary to restore that previous value in the future (if the resource allocation strategy is unable to satisfy the water demand), the value of discharge from *"TF8"* at day 1 is stored inside the "*$storage*" fact (line 12). In the LHS of the rule, a statement that requires the existence of the storage fact is made (line 6). Then, on the RHS, before updating the *"TF8"* discharge observation for day 1 (line 13), the current value is stored inside the mentioned storage fact (line 12).

```
1    rule "Reset Observation values"
2    when
3     $flag:ActivationFlag(ruleName == "Reset Observation values")
4     $DemandSatisfiedObservation: Observation(id ==
     "DemandSatisfiedObservation")
5        $storage: ObservationValuesStorage()
6    then
7     for(String id : $storage.values.keySet())
8     {
9      LOGGER.info("Reseting "+id+" to value "+$storage.values.get(id));
10     TimeValuePair tvp = (TimeValuePair)WM.getObjects(new
     IdObjectFilter(id)).iterator().next();
11     modify(tvp){setValue($storage.values.get(id));}
12    }
13
14    $storage.values = new HashMap<String, Double>();
15    RulesUtils.signalRuleExecuted($flag,WM);
16   end
```

Listing 7"Rule Reset Observation values"

All the values stored inside the "*ObservationValuesStorage*" fact can be retrieved by executing the rule "*Reset Observation values*" (Listing 7). The RHS of this rule iterates over all the "*TimeValuePair*" ids from the "*ObservationValuesStorage*" fact (lines 7 to 12) searching the "*TimeValuePair*" inside the Working Memory with the same id as the defined in the rule (line 10). At the moment "*Time Value Pair*" id is found, the correspondent value in that "*TimeValuePair*" is updated with the one from the "*ObservationValuesStorage*" fact (line 11). After all the necessary "*TimeValuePairs*" have been reset, the contents from the "*ObservationValuesStorage*" fact are also reset (lines 14 and 15).

2.1.1 Changing the Working Memory Population Procedure

The RBR procedure for the ACA case defined on the Deliverable 4.1 on Section 4.3.1 has suffered some modifications with respect to the ones introduced on Deliverable 4.2 on Section 3.1. In the implementation proposed in that deliverable, Alibaba was used during the facts definition procedure. Alibaba is a Java library that allows to make SPARQL queries to a Sesame repository. This library is normally used to obtain the results (semantic instances) as Java objects instances. Initially, this option seemed ideal due to it allows to make available the knowledge stored inside the Sesame repository to the rules evaluation engine using a classes structure. This class structure is formed by an exact map with the ontology, allowing to describe the rules in a semantic way. Another major benefit from using that approach was the automatic procedure of obtaining the data from the Sesame repository and inserting it into the Working Memory. With very little effort, one could have the facts stored inside the Sesame repository, available in the Working Memory. Despite the mentioned benefits, this approach was abandoned due to the fact that the Alibaba library was unable to handle the complexity of the ontology being used. The ontology being used models to model the domain is complex in terms of relations between classes (inverse, functional, cardinalities of the elements, etc.) and contains diverse knowledge that has to be inferred. Although Alibaba seemed to be a feasible solution when facing simple ontologies, it has been shown that it is not able to work with more complex ones. The main problems found when trying to map the current ontology were that Alibaba was not able to infer certain information and errors regarding the cardinality of the attributes.

Regarding the lack of information, Alibaba was not able to infer properties and relations like the property "*isObservationOf*" that links an "*Observation*" with the "*FeatureOfInterest*" that it observes, while the Sesame repository along with a reasoner are able to infer them. In terms of the cardinality of the attributes, Alibaba was not able to assign the appropriate cardinality to the Java classes it generates. Thus, attributes that should be modeled using a simple type or class were modeled with collections (e.g. Sets) of that simple type or class. For instance, when modeling the "*id*" attribute associated with each entity that could be modelled using a "*String*". However, Alibaba model the *"id"* using *Set<String>* (collection of strings). Although this problem did not impede the definition of the rules, it increased the complexity of their translation to DRL rules, making them harder to be read or model inside the rules.

```
1   $ST1WaterResource : WaterResource (getIds().iterator().next() == "ST1")
2   $ST1WaterResource : WaterResource (id == "ST1")
```

Listing 8"Simple statement from the LHS of a rule"

In that sense, Listing 8 shows a simple statement from the LHS of a rule where a fact of type "*WaterResource*" with id ST1 is matched. On the one hand, line 1 shows how this statement had to be expressed when modelling the attribute "*id*" as a collection of "*String*". On the other hand, line 2 shows how the same statement is expressed when the cardinality of the attribute "*id*" is correctly set. As can be seen, the upper approach is much more verbose and cumbersome whilst the bottom one is more concise and straightforward. After facing these series of problems, the decision to substitute the use of Alibaba for an ad-hoc ontology-to-object mapping procedure was taken.

Another reason that motivates the changes on the Working Memory population procedure is the change on how the required data is fed to the RBR. Initially, the data regarding the logical model and the observations was designed to be transferred using a Resource Description Framework (RDF) file. This file would contain all the information about the structure of the water distribution network ("*WaterResources*", "*FeaturesOfInterest*", "*Offerings*", etc.) along with the "*Observations*" and their associated "*TimeSeries*". Finally, due to reasons regarding the efficiency of that approach, it was decided to store the data separated from the ontology (Deliverable 3.2- "*WDW 1st prototype*"). After the changes on how the actual "*TimeSeries*" are stored, it was decided to communicate those time series using WaterML2.

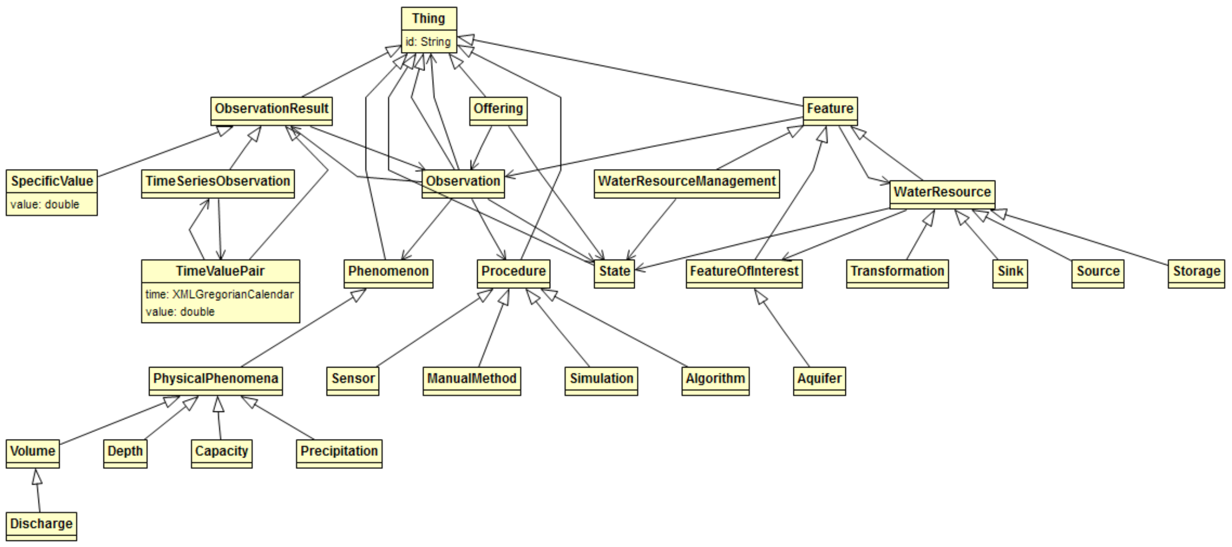

Figure 4 "Model of the information stored in the Sesame repository"

Figure 4 shows a diagram of the Java model classes used to represent the information stored in the Sesame repository. These classes are totally aligned with the ones defined in the ontology (D1.3 "*Generic Ontology for water supply distribution chain*").

Therefore, the new Working Memory population procedure is divided in two steps: WaterML2 parsing and ontology querying. During the WaterML2 parsing step, the input files containing all the *"TimeSeriesObservations"* needed for the execution of the RBR are parsed and the data is stored in-memory. Normally, the input files for the RBR are: *"dailyDemandForecast"*, *"waterVolumeContribution"*, *"predictedWaterVolume"* and *"waterTreatmentPlantsState"*. The file *"dailyDemandForecast"* contains the forecasted values for the daily demand for the next three days. The file *"waterVolumeContribution"* contains forecasted values of resources non-regulated produced by the rain throughout the water supply and distribution chain for the next three days. The file *"predictedWaterVolume"* contains predicted water volume of the different reservoirs and aquifers that compose the logical model for the next three days. The file *"waterTreatmentPlantsState"* contains the status of the different water treatment plants (working or not).

By using an XML parser, the nodes contained in the mentioned WaterML2 files are read and instances of *"TimeSeriesObservation"* are created and populated with the values from the file. These *"TimeSeriesObservation"* instances are then stored inside a *Map* where the key (*String*) is the *"id"* of the time series and the value (*"TimeSeriesObservation"*) is the actual *"TimeSeriesObservation"* instance.

Once the WaterML2 files have been read, information regarding the logical model (*"FeaturesOfInterest"*, *"WaterResources"*, etc.) is extracted from the ontology. To achieve this goal, the ontology is published in a local Sesame repository, and then queries are performed over the semantic graph. The RDF file describing the ontology is published into a local in-memory Sesame repository in order to allow the RBR engine to make queries without the need of communicating with a remote Sesame repository, improving the overall speed of the solution. Once the local Sesame repository has been configured and the ontology is available, the process of retrieving the required information starts. To obtain this information, a series of SPARQL queries are performed against the ontology. SPARQL queries permits to interrogate the ontology about the information stored on this system. After executing the defined query, SESAME retrieves the results in the form of triples ("*subject*", "*object*" and "*predicate*" variables). Then, these triplets need to be translated into instances of the Java class model in order to make them available for the Drools rules analyzer. This process of translation consists of, starting from the result of a SPARQL query in the form of triples, instantiate the pertinent Java class and set the appropriate values to its attributes, according to the information stored in the ontology. Whenever an *"Observation"* is retrieved from the ontology, the *"TimeSeriesObservation"* that contains the results of the *"Observation"* is retrieved from the in-memory representation of the WaterML2 files and linked to it. Once the information required to evaluate the rules is retrieved from the Sesame repository and the Java classes are instantiated, all of these instances are inserted into the Drools Working Memory.

```
CONSTRUCT {
?entityModel WatERPOntology:id ?id ;
a ?type ;
WatERPOntology:hasObservation ?wrObservation .
?wrObservation WatERPOntology:hasProcedure ?wrObservationProcedure .
?wrObservation WatERPOntology:hasPhenomenon ?wrObservationPhenomenon .
?entityModel WatERPOntology:hasFeature ?foi .
?foi WatERPOntology:hasObservation ?observation .
?observation WatERPOntology:hasPhenomenon ?phenomenon .
?observation WatERPOntology:hasProcedure ?procedure .
}
WHERE {
 ?lmodel WatERPOntology:isModelOf ?entityModel .
 FILTER (?lmodel = WatERPOntology:TerLlobregatWaterResourceManagement)
.

 ?entityModel WatERPOntology:id ?id .
 OPTIONAL {
   ?entityModel WatERPOntology:hasObservation ?wrObservation .
   {
   OPTIONAL {
    ?wrObservation WatERPOntology:hasProcedure ?wrObservationProcedure
      .
   } OPTIONAL {
    ?wrObservation WatERPOntology:hasPhenomeno
    ?wrObservationPhenomenon .
   }
  }
}
{
 ?entityModel a ?type .
}
 UNION
 {
```

```
33      ?entityModel WatERPOntology:hasFeature ?foi .
34      {
35       ?foi WatERPOntology:hasObservation ?observation.
36       OPTIONAL {
37        ?observation WatERPOntology:hasProcedure ?procedure .
38       } OPTIONAL {
39        ?observation WatERPOntology:hasPhenomenon ?phenomenon .
40       }
41      }
42    }
43    FILTER (
44     ?type = WatERPOntology:Transport
45     || ?type = WatERPOntology:Sink
46     || ?type = WatERPOntology:Storage
47     || ?type = WatERPOntology:Source
48     || ?type = WatERPOntology:Transformation
49     ).
50    }
```

Listing 9 "SPARQL query"

In Listing 9 is depicted an example of a SPARQL query used for querying the Sesame repository. This query takes advantage of the *"CONSTRUCT"* structure from SPARQL syntax (Prud'hommeaux & Seaborne, 2008). This structure permits to return a single RDF graph formed by the union of RDF graphs generated inside the *"WHERE"* clause (lines 12 to 50). Then, the resultant graph is formed by the variables that matches with the "*CONSTRUCT*" part (lines 1 to 11). Hereafter, this specific SPARQL query returns any *"WaterResource"* of type *"Transport"*, *"Sink"*, *"Storage"*, *"Source"* and *"Transformation"* (lines 44 to 48). For each matching *"WaterResource"* all its *"Observations"* are returned along with their related *"Procedure"* (lines 19 to 22) and *"Phenomenon"* (lines 22 to 25). Additionally, all the *"FeaturesOfInterest"* linked to each *"WaterResource"* are also returned. Those *"FeaturesOfInterest"* are obtained with a *"UNION"* query (lines 31 to 41) that matches any relation between a *"WaterResource"* and a *"FeatureOfInterest"* (line 33). For each of these features of interest, all their related *"Observations"* are returned along with their *"Procedure"* (lines 36 to 38) and *"Phenomenon"* (lines 38 to 40).

2.1.2 Exposing RBR as a Web Service

In order to allow the integration of the RBR Engine with the WatERP framework, the RBR exposes an OGC-WPS server, named as WPS-DSS (see *D2.3* "*Open Interface Specification*" in Section 3.2 on page 24). Following the procedure described in D.4.3 Inference and Simulation Engine 1st prototype section 3, a "*ProcessDescription"* document is provided in order to allow the publication of the RBR Engine in the SOA-MAS architecture. The "*ProcessDescription"* file (see Listing 10) contains the definition of the inputs expected by the RBR and the output it produces.

```
     ...
  <ProcessDescription statusSupported="false"     storeSupported="false"
wps:processVersion="1.0">
```

```
<ows:Identifier>gr.iccs.waterp.dms.process.WaterAllocationSimulationProcess<
/ows:Identifier>
       ...
       <DataInputs>
      <Input minOccurs="1" maxOccurs="1">
        <ows:Identifier>logicalModel</ows:Identifier>
        ...
      </Input>
      <Input minOccurs="1" maxOccurs="1">
        <ows:Identifier>dailyDemandForecast</ows:Identifier>
        ...
      </Input>
      <Input minOccurs="1" maxOccurs="1">
        <ows:Identifier>waterVolumeContribution</ows:Identifier>
        ...
         </Input>
      <Input minOccurs="1" maxOccurs="1">
        <ows:Identifier>predictedWaterVolume</ows:Identifier>
        ...
      </Input>
      <Input minOccurs="1" maxOccurs="1">
        <ows:Identifier>waterTreatmentPlantsState</ows:Identifier>
       ...
      </Input>
    </DataInputs>
    <ProcessOutputs>
      <Output>
        <ows:Identifier>waterAllocationSimulationResult</ows:Identifier>
       ...
      </Output>
    </ProcessOutputs>
  </ProcessDescription>
  ...
```

Listing 10 "ProcessDescription file"

The inputs required by the RBR engine are *"logicalModel"*, *"dailyDemandForecast"*, *"waterVolumeContribution"*, *"predictedWaterVolume"* and *"waterTreatmentPlantsState"*.

The parameter *"logicalModel"* contains the RDF file describing the ontology that represents the structure of the logical model for the particular case.

The parameters *"dailyDemandForecast"* contains the forecasted values for the daily demand for the next three days. This data is transferred using a WaterML2 document.

The parameter *"waterVolumeContribution"* contains the forecasted values of resources non-regulated produced by the rain throughout the water supply and distribution chain for the next three days. This data is transferred using a WaterML2 document.

The parameter *"predictedWaterVolume"* contains the predicted water volume of the different reservoirs and aquifers that compose the logical model for the next three days. This data is transferred using a WaterML2 document.

The parameter "*waterTreatmentPlantsState"* contains the status of the different water treatment plants (working or not). This data is transferred using a WaterML2 document.

The output produced by the RBR, as stated in the "*ProcessDescription"* file under the *"ProcessOutputs"* node, is a WaterML2 document named *"waterAllocationSimulationResult"* containing the information regarding the simulated discharge from various "WaterResources" and the fulfilled demand. More precisely, the *"waterAllocationSimulationResult"* contains *"ObservationMembers"* describing the water abstraction from the Storages (*"ST1"*, *"ST2"*, *"ST3"* and *"ST4)*, the amount of *"RNR"* used and the amount of water treated by the desalination plant (*"TF10"*). All these "ObservationMembers" contain a time series describing the amount of water abstracted/used/treated for each of the three following days. These *"ObservationMembers"* are also denoted with an attribute *"parameter"* containing the resource priority. The priority of the water resources describes the order on which water is extracted from those resources in order to fulfill the demand. This priority (defined in paragraph *F Defining Resources Priorities* of the rules) is an integer value ranging from *1* to *8* where *1* indicates the highest priority (from where water is going to be extracted first) and *8* the lowest (from where water is going to be extracted last). In order to include this information inside a WaterML2 document, the *"parameter"* attribute of "*OM_Observation"* has been used. This attribute, as defined in ISO19156 section 7.2.2.5 (International Organization for Standardization, 2010), allows to annotate an "*OM_Observation"* with a key-value pair used to describe an arbitrary event-specific parameter. Additionally, the output WaterML2 contains an observation member for each storage (*"ST1"*, *"ST2"*, *"ST3"* and *"ST4)* with a time series describing the evolution of its water volume for the next three days (according to the simulated water abstraction).

```
<wml2:observationMember>
    <om:OM_Observation gml:id="Obs_ACA_ST1_Discharge1Day_RBRSimulation"
parameter="priority=1">
      <om:resultTime>
        <gml:TimeInstant gml:id="WatERP.resTime.1">
          <gml:timePosition>2016-01-01T00:00:00+00:00</gml:timePosition>
        </gml:TimeInstant>
      </om:resultTime>
      <om:procedure
        xlink:href="http://www.waterp-
fp7.eu/WatERPOntology.owl#RBRSimulation"
        xlink:title="RBRSimulation" />
      <om:observedProperty
        xlink:href="http://www.waterp-
fp7.eu/WatERPOntology.owl#Discharge1Day"
        xlink:title="Discharge1Day" />
      <om:result>
        <wml2:MeasurementTimeseries gml:id="WatERP.ACA.Ts.13000">
          <wml2:defaultPointMetadata>
            <wml2:DefaultTVPMeasurementMetadata>
              <wml2:uom code="m3" />
              <wml2:interpolationType

xlink:href="http://www.opengis.net/def/waterml/2.0/interpolationType/Continu
ous"
                xlink:title="Instantaneous" />
            </wml2:DefaultTVPMeasurementMetadata>
          </wml2:defaultPointMetadata>
          <wml2:point>
            <wml2:MeasurementTVP>
              <wml2:time>2016-01-01T23:59:59+00:00</wml2:time>
              <wml2:value>500.00</wml2:value>
            </wml2:MeasurementTVP>
          </wml2:point>
          <wml2:point>
            <wml2:MeasurementTVP>
              <wml2:time>2016-01-02T23:59:59+00:00</wml2:time>
              <wml2:value>500.00</wml2:value>
            </wml2:MeasurementTVP>
          </wml2:point>
          <wml2:point>
            <wml2:MeasurementTVP>
              <wml2:time>2016-01-03T23:59:59+00:00</wml2:time>
              <wml2:value>500.00</wml2:value>
            </wml2:MeasurementTVP>
          </wml2:point>
        </wml2:MeasurementTimeseries>
      </om:result>
    </om:OM_Observation>
  </wml2:observationMember>
```

Listing 11 "Result file snippet"

Listing 11 shows an example of *"ObservationMember"* from the result file. The observation, identified with the id *"Obs_ACA_ST1_Discharge1Day_RBRSimulation"* refers to the simulated water discharge from *"ST1".* It contains a node *"procedure"* that node defines the procedure used to generate this observation. In this particular case, the procedure is *"RBRSimulation"* which is defined inside the ontology "*http://www.waterp-fp7.eu/WatERPOntology.owl#RBRSimulation*".

The node *"observedProperty"* defines that phenomenon observed by the observation is of type *"Discharge1Day"* which is defined by the concept *"http://www.waterp-fp7.eu/WatERPOntology.owl#Discharge1Day"* from the ontology.

Finally, the shown *"ObservationMember"* contains a node of type *"MeasurementTimeseries"* that contains the time series of the water discharge from *"ST1"* for the next three days after the simulation.

2.2 Final version of the RBR Procedure

After describing the changes performed over the RBR procedure, the final version can (see Figure 5) be divided in four steps: (i) construction of the Rules Repository, (ii) population of the Working Memory, (iii) execution of the Drools Engine and (iv) communication of the Results.

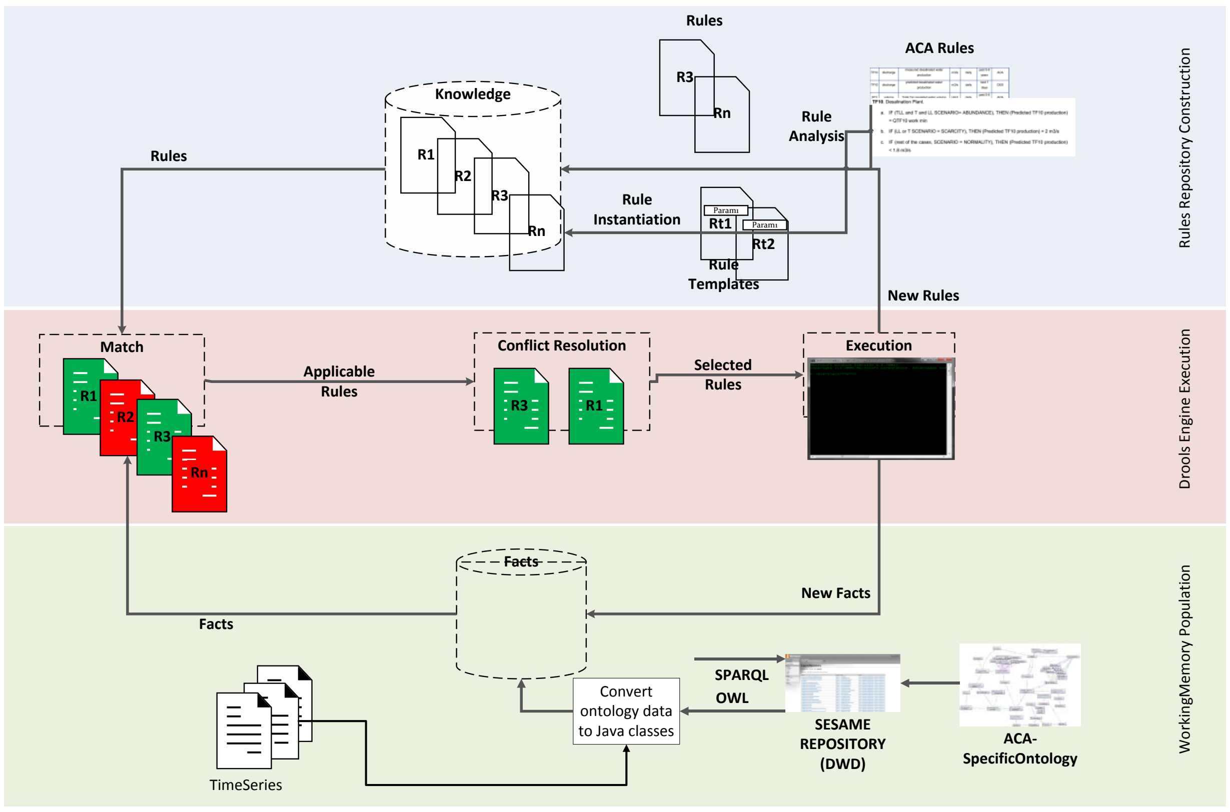

Figure 5 "Final RBR execution procedure"

2.2.1 Construction of the Rules Repository

Upon reception of a request from the SOA-MAS to make a simulation for a given scenario, the RBR procedure starts by populating the rules repository with all the rules. Those rules are the translation of the expert knowledge from water managers into rules that can be understood by the Drools Rules Engine (expressed in DRL). All that knowledge is inserted into the Drools Working Memory in order to make the engine aware of it. The process of populating the Rules Repository consists in reading all the files that contain the rules definitions and inserting them into the Drools Rules Repository. There are two types of rules, literal rules and template rules.

Literal rules are those that model the exact knowledge of the domain expert. Those rules are self-contained in the sense that no extra information is needed to make them ready to be evaluated by the rules engine.

On the contrary, template rules are an abstraction of the knowledge of the domain expert. Those rules group a subset of the reference rules that are very similar in terms of structure but each has minimal differences. Those differences are represented in terms of variables of the template. For instance, if three rules to determine the status of a reservoir have the same structure but differ in the range of water levels on which the reservoir is considered to be in scarcity, abundance or normality status, the rules could be modeled as a template (Deliverable 4.2 on Section 3.2 in page 25). The template would have a parameter that identified the upper and lower water levels and the status assigned to the reservoir when its water level falls within that range. When executing the RBR each template rule is instantiated to create an arbitrary number of N literal rules. This process of instantiation is performed by providing to the engine the rule to be instantiated along with a collection of parametrizations (one parametrization for each literal rule to be generated). Then, the engine replaces the parameters from the rule with the provided values and inserts the new rule into the rules repository.

2.2.2 Population of the Working Memory

Continuing with the process of executing the RBR procedure, once the Rules Repository has been filled with all the rules (literal rules and instantiation of template rules), the next step is to populate the Working Memory. This process consists on retrieving the knowledge from the ontology and the WaterML2 file and inserting this knowledge in the Drools Working Memory.

As explained before, the SOA-MAS provides to the RBR an RDF file with the ontology that models the case being analyzed and a series of WaterML2 files containing the observations results. To insert the needed information into the Drools Rules Engine, the first step is to parse the WaterML2 files and store its contents in-memory. Later on, the provided RDF file is published inside a local Sesame repository and relevant information is extracted from it. Finally, all the Java model classes instantiations created during these steps are stored inside the Working Memory.

2.2.3 Executing the Drools Engine

After the Rules Repository has been filled with all the rules (literal and templates instantiations) and the Working Memory has been populated with all the required information from the ontology, the Rules Engine execution starts.

The Drools execution (The JBoss Drools Team, 2012) consists of an iterative process on which, for each step the system evaluates which rules are eligible for being executed given the current state of the Working Memory (matching process). After this step, the list of rules eligible for being executed (Agenda) is updated, removing all the rules whose predicated defined in the LHS are no longer fulfilled with the status of the working memory and adding new rules that meet this condition. After the activation agenda is updated, the rules contained on it are sorted according to their priority (salience) and the first one is executed. The execution of a rule can lead to modifications of the Working Memory, those modifications are due to new knowledge inferred by the RHS. This iterative process ends once there is no more knowledge to be inferred and there LHS of any rule is fulfilled.

2.2.4 Communication of the Results

The main goal of the RBR is to decide the best water allocation strategy to fulfill the demand of the next three days taking into account political, environmental and socio-economical aspects. This water allocation strategy decides how much water is extracted/used/treated from each of the

"*WaterResources*" that compose the logical model and the order (priority) with which that resources are used.

The primary "*WaterResources*" from which water can be used are the storages (*"ST1"*, *"ST2"*, *"ST3"* and *"ST4)*, the non-regulated resources (*"RNR"*) introduced in multiple points throughout the model (*"So3"*, *"So4"* and *"So5"*), the desalination plant (*"TF10"*) and the water treatment plants (*"TF7"* and *"TF8"*).

The result produced by the RBR is a collection of time series describing the daily amount of water contributed by each of these *"WaterResources"* to fulfil the each daily demand (day 0, day 1 and day 2). These results are returned using a WaterML2 document.

After the evaluation of the rules has ended, the Drools *"WorkingMemmory"* contains the observations that describe the selected water allocation strategy. In order to return these observations as a result of the RBR execution, the required information has to be extracted from the *"WorkingMemory"* and stored inside a WaterML2 document. To this extent, a series of queries over the *"WorkingMemory"* are performed. As an example, Listing 12 shows how the required observations are extracted from the Drools *"WorkingMemory"*. In this example the *"WorkingMemory"* is queried for the observation describing the simulated discharge from *"ST1"*.

```
Observation ST1DischObs = getObservation("ST1", "Discharge1Day", "RBRSimulation");
```

Listing 12 "WorkingMemory observation retrival"

Once all the needed information has been extracted from the Drools *"WorkingMemory"*, a WaterML2 document containing that information is constructed. As described in *Section 2.1.2*, the resulting document contains an *"ObservationMember"* for each of the previously mentioned *"WaterResources"* (*"ST1"*, *"ST2"*, *"ST3"* ,*"ST4","RNR", "TF10", "TF7" and "TF8").* Each observation member contains a time series describing the amount of water abstracted/used/treated for each of the three following days. These *"ObservationMembers"* are also noted with an attribute *"parameter"* containing the resource priority (from 1 to 8).

```
WaterML2 wm = new WaterML2(
"waterAllocationSimulationResult",
"Water allocation simulation result",
new DateTime().toString(DateTimeFormat.forPattern("yyyy-MM-
dd'T'HH:mm:ssZZ")),
"DMS",
dt.toString("yyyy-MM-dd'T'HH:mm:ssZZ"),
mdt.toString("yyyy-MM-dd'T'HH:mm:ssZZ"));
List<ObservationMember> omList = new
ArrayList<WaterML2.ObservationMember>();
wm.setOmList(omList);

omList.add(wm.new ObservationMember() {{
 setId("Obs_ACA_ST1_Discharge1Day_RBRSimulation");
12     setInterpolationType("AveragesSucc");
13     setPhenomenon("Discharge1Day");
14     setProcedure("RBRSimulation");
15     setTimePosition(endDt.toString("yyyy-MM-dd'T'HH:mm:ssZZ"));
16     setUom("m3");
17    setParameter("priority="+((SpecificValue)(ST1PriorObs.getObservationRe
      sult())).getValue());
18    setPointList(new ArrayList<WaterML2.Point>());
19    }});

20    TimeSeriesObservation ST1DischObsTS =
      ((TimeSeriesObservation)ST1DischObs.getObservationResult());
21    for(TimeValuePair tvp : ST1DischObsTS.getValues()) {
22       omList.get(0).getPointList().add(wm.new
      Point(tvp.getTime().toString("yyyy-MM-
      dd'T'HH:mm:ssZZ"),tvp.getValue().toString()));
23    }
  ....
```

Listing 13 "Construction of WaterML2 document"

Listing 13 shows an example of how the resulting WaterML2 document is constructed. In this example, a WaterML2 instance is created with the *"waterAllocationSimulationResult" id* (line 1).

Then, this WaterML2 instance is composed by a list of *"ObservationMembers"* that is created in lines 8 and 9. Then, a new instance of *"ObservationMember"* is added (lines 10 to 19). This new observation observes the *"Discharge1Day"* phenomenon (line 13) with the RBRSimulation procedure (line 14). The attribute *"parameter"* is assigned the attribute *"priority"* with the value from *"ST1PriorObs"* (line 17). This observation, generated during the execution of RBR engine, contains the value of priority assigned to the *"WaterResource" "ST1"*. Later on (lines 21 to 23), the *"ObservationMember"* is populated with the *"TimeValuePairs"* coming from *"ST1DischObs"*. This observation, associated with the *"ST1"* contains the water discharge simulated for the next three days.

2.3 Implementing Tests to Validate the Results of the RBR

In order to validate the correctness of the results provided by the RBR application in the ACA case and apart from the unitary tests that have been implemented to validate certain rules, tests that englobe all the rules have also been defined. These new tests comprise all the rules defined in Deliverable 4.1 and the focus on validating the correct interrelation between them. In order to test the Inference Engine under all the possible conditions of the system, a custom data set has been created. This custom data set contains observations of the demand and water level predictions with lectures on a daily basis and spanning throughout a whole year. Special effort has been put when creating the data set to make sure it contains a representation of all the possible scenarios that the inference engine could face (water scarcity, abundance and normality). The testing procedure has been focused on iteratively evaluating the rules on day at a time (for each 365 days) and validating that the engine produces the appropriate results. Detailed information about the testing procedure can be found on D7.5.3- "*Implementation of the Water DSS*" that is going to be presented at same time as this deliverable.

In order to simplify the test procedure, it has been decided not to use the ontology as a definition of the logical model. Instead, an ad-hoc instantiation of the classes has been performed. For the same reason, the series of observations also come in a text file with a tabular format, and not in WaterML2 formatted documents.

```
1    Sink Si1 = new Sink("Si1");
2    kSession.insert(Si1);
3    RulesUtils.addObservationWithTVPair(Si1, DISCHARGE, PREDICTION_PROCEDURE,
     dataSheet.getTimeSeries(startRow, numRows, 'K'), wm);
```

Listing 14 "Instantiation of the test case"

As an example, Listing 14 shows a fragment of the Working Memory ad-hoc population process in which the "*WaterResource*" *"Si1"* is manually instantiated (line 1) and populated with an observation (line 3). The method "addObservationWithTVPair" (line 3) creates an observation of the provided type (*"Discharge")* with the given procedure (*"Prediction"*) and fills it with the provided time series. The method "*getTimeSeries*" reads the test data file and constructs a time Series with the values found between rows "*startRow*" and "*startRow+numRows*" under the given column (K).

Date	**ST1**	**ST2**	**ST3**	**So3**	**So4**	**So5**	**So6**	**So7**	**So8**	**Si1**	**Si2**
	(hm^3)	*(hm^3)*	*(hm^3)*	*(m^3/s)*	*(m^3/s)*	*(m^3/s)*	*(masl)*	*(masl)*	*(m^3/s)*	*(hm^3)*	*(hm^3)*
01/01/2016	*50.00*	*40.00*	*180.00*	*1.00*	*1.00*	*1.00*	*-2.00*	*0.00*	*3.83*	*0.50*	*6.00*
02/01/2016	*49.82*	*39.82*	*180.00*	*1.00*	*1.00*	*1.00*	*-2.00*	*0.00*	*3.83*	*0.50*	*6.00*
03/01/2016	*49.65*	*39.65*	*180.00*	*1.00*	*1.00*	*1.00*	*-2.00*	*0.00*	*3.83*	*0.50*	*6.00*
04/01/2016	*49.47*	*39.47*	*180.00*	*1.00*	*1.00*	*1.00*	*-2.00*	*0.00*	*3.83*	*0.50*	*6.00*
05/01/2016	*49.30*	*39.30*	*180.00*	*1.00*	*1.00*	*1.00*	*-2.00*	*0.00*	*3.83*	*0.50*	*6.00*

Table 4 " Values of the CSV test data file in tabular format"

The information to populate the Drools Working Memory has been acquired from a Comma Separated Values (CSV) file (test data file) where for each *"WaterResource"*, the different time series are defined (see Table 4). In particular, the first column contains the date when the observations are performed, columns *"ST1"*, *"ST2"* and *"ST3"* contains the observations of water volume of each storage. Columns *"So3"*, *"So4"*, *"So5"* and *"So8"* contain the amount of water that comes from the sources. Columns *"So6"* and *"So7"* contain the values regarding the water level of the aquifer and sea respectively. Finally, *"Si1"* and *"Si2"* show the demand predicted for Urban and industrial use and *"Canal de la dreta"* respectively.

During the execution of one iteration of the test procedure, the rules analyser evaluates the water resources allocation strategies and selects the first one that meets the water demand. In order to reach to a conclusion, multiple rules are executed and new facts are introduced into the working memory. To be able to understand the path followed (sequence of rules executed) by the rules engine and see the inference performed on each step, the rules engine has been configured to produce a verbose output.

```
2015-05-12 16:29:11,740 INFO [RulesAnalyzer] - -------------------------Rule:
E.22.a-------------------------
2015-05-12 16:29:11,740 INFO [RulesAnalyzer] - TLL and LL are in ABUNDANCE or
NORMALITY-> PREDICTED QTF3 ENV >=4.5
2015-05-12 16:29:11,741 INFO [RulesAnalyzer] - INSERTED: TimeValuePair
id:Obs_TF3_DischargeEnvironmentalMin_PredictionProcedure_TS_0 time:2016-10-
26T02:00:00.000+02:00 value:4,500000
2015-05-12 16:29:11,742 INFO [RulesAnalyzer] - INSERTED:
org.bdigital.alim.waterp.dss.ie.rbr.model.TimeSeriesObservation@5f872ecf
2015-05-12 16:29:11,746 INFO [RulesAnalyzer] - INSERTED:
org.bdigital.alim.waterp.dss.ie.rbr.model.Observation@2944bbd3
2015-05-12 16:29:11,746 INFO [RulesAnalyzer] - ---------------------------------
-----------------------------
```

Listing 15 "Output produced when executing a rule"

In this line, Listing 15 shows an example of the produced output when executing a rule from the Rules Repository. In this particular case, the rule *E.22.a* is executed. As a consequence of the execution of the rule, a new "*Observation*" is inserted into the Working Memory. This observation contains a *"TimeSeriesObservationResult"* formed with a *"TimeValuePair"* with time *2016-10-26T02:00* and value *4.5*. At this time, Drools provides the possibility to add some code that is executed whenever a change occurs in the Working Memory (when a fact is inserted, modified or retracted). This code is provided to the Rules Engine via an instance of a class that implements the interface *"WorkingMemoryEventListener"*.

```java
public class CustomWorkingMemoryEventListener implements
WorkingMemoryEventListener{
Logger logger;

public CustomWorkingMemoryEventListener(Logger logger)
{
 this.logger = logger;
}

public void objectInserted(ObjectInsertedEvent arg0) {
  logger.info("INSERTED: "+arg0.getObject().toString());
}

public void objectRetracted(ObjectRetractedEvent arg0) {
  logger.info("RETRACTED: "+arg0.getOldObject().toString());
}

public void objectUpdated(ObjectUpdatedEvent arg0) {
  logger.info("UPDATED: "+arg0.getObject().toString());
}
}
```

Listing 16 "Implementation of WorkingMemoryEventListener"

Listing 16 shows the custom implementation of "*WorkingMemoryEventListener"* used for visualizing the Drools execution process. This class provides an implementation of each of the methods from the interface (*"objectInserted"*, *"objectRetracted"* and *"objectUpdated"*). For each method, a message containing the description about the event signalled is written to a logger.

Although the shown output provides a good insight of the rules evaluation process, when implementing the rules it has been necessary to provide a way to stop the execution at a certain point and evaluate the state of the system (WorkingMemory, Agenda, etc.). Although, Drools offers the possibility to add breakpoints on the RHS parts of the rules in order to debug them, it has not been possible to do so, as Drools would not stop on those breakpoints. Apparently, many other developers have faced this same issue.

To overcome this problem, a custom build of the Drools Rules engine has been done. Internally, Drools convert the RHS of each rule in a Java class with a method that is executed whenever the rule is executed (the LHS of the rule is fulfilled).

```
rule "E.15.c"
when
  not
  (
  $TLLState : Offering(id == "TLLStateOffering" , state == ABUNDANCE) and
  $TState : Offering(id == "TStateOffering" , state == ABUNDANCE)  and
  $LLState : Offering(id == "LLStateOffering" , state == ABUNDANCE)
  )
  not
  (
  $TState : Offering(id == "TStateOffering" , state == SCARCITY)  or
  $LLState : Offering(id == "LLStateOffering" , state == SCARCITY)
  )
  $TF10 : WaterResource (id =="TF10")
then
  LOGGER.info("REST OF THE CASES -> PREDICTED QTF10 PROD <1.8m3/s");

  TimeValuePair tvp =
RulesUtils.getObservationTVPByDate($TF10,DISCHARGE,PREDICTION_PROCEDURE,D2_DATE);

  modify(tvp){setValue(1.7)};
end
```

```
public class Rule_E_15_c {
    private static final long serialVersionUID = 510l;

public static void defaultConsequence(org.drools.spi.KnowledgeHelper drools,
org.bdigital.alim.waterp.dss.ie.rbr.model.WaterResource $TF10,
org.drools.FactHandle $TF10__Handle__ ,   javax.xml.datatype.XMLGregorianCalendar
D2_DATE ,  org.bdigital.alim.waterp.dss.ie.rbr.model.Discharge DISCHARGE ,
org.slf4j.Logger LOGGER ,  org.bdigital.alim.waterp.dss.ie.rbr.model.Algorithm
PREDICTION_PROCEDURE  ) throws Exception { org.drools.runtime.rule.RuleContext
kcontext = drools;
    LOGGER.info("REST OF THE CASES -> PREDICTED QTF10 PROD <1.8m3/s");

               TimeValuePair tvp = 
RulesUtils.getObservationTVPByDate($TF10,DISCHARGE,PREDICTION_PROCEDURE,D2_DATE);

               { org.bdigital.alim.waterp.dss.ie.rbr.model.TimeValuePair __obj__
= (tvp); org.drools.FactHandle __obj____Handle2__ = 
drools.getFactHandle(__obj__);__obj__.setValue(1.7); drools.update(
__obj____Handle2__, 9223372036854775807L ); };
}

}
```

Listing 17 "The implementation of rule E.15.c"

Listing 17 shows the implementation of rule E.15.c using DRL (top) and how the Drools transforms the RHS into a Java class (bottom).

By default, Drools places these classes inside the system temporal directory and cannot be accessed. Although, Drools offers the possibility to store those classes in a location specified by the user. Using this feature, we could add the folder where the compiled rules were placed inside the build path of the Java project and this allowed adding breakpoints to that code.

3. Case-Based Reasoning (CBR)

This section is aimed at depicting the final version of the CBR. Mostly, the development of the CBR was finished in the Deliverable 4.2. However, several issues were found during the CBR deployment in the pilots (see Section 3.1). By solving these issues, the CBR has been enhanced in terms of memory leakage, better integration with the visualization environment and enhancements of the recommendations and alerts referring the validity of certain pumping scheduling. Hence, the final version of the CBR is depicted in Section 3.2, where the whole CBR process is described according to the features that each phase contains according to the present and past enhancements. As a main highlight of this section, the CBR is ready to be used in the demo-sites, offering energy reduction by means of planning of suitable pumping schedules based on lived-past cases merged with water distribution manager experience

3.1 CBR Enhancements deriving from pilot's

This part of the document is focused on describing the changes performed over the CBR due to the pilot's deployment process. In the deployment process, several issues were found during the execution of the test (D7.5.2- "*Implementation of the Water DSS*"). Mainly, the encountered issues were related with the memory consumption leakage during the CBR execution (see Section 3.1.1). Other minor changes have been performed in reference to the path configuration for input/output files, the EPANET configuration in an external machine, the configuration of the database, and the configuration of a directory to store temporal files generated by the CBR. These minor changes were derived from the deployment of the CBR in the demo-sites, requiring to make configurable several parameters with external tools and results (see Section 3.1.2). Moreover, one relevant change performed over the CBR has been regarding the case adaptation and evaluation phase where a rule-based learning has been applied to check the infrastructure safety and water quality during the case simulation (see Section 3.1.3).

3.1.1 CBR Enhancements Derived from the Test Execution (WP7)

The CBR has been tested during this last period of time in the framework of the WP7. This test was focused on evaluating the main planning and learning capabilities of the CBR. According to this, the test was focused on evaluating the pumping planning and the energy consumption gain for a specific data set (composed by measured demand, tank volume and adopted pumping planning for a whole year) by comparing the proposed CBR energy consumption (the proposed CBR pumping planning) with the real energy consumption (the real pumping planning). Furthermore, the test also captures RAM memory usage and CPU cycles for each test with the aim of evaluating the resources consumed by the CBR and compare the CBR efficiency with the computational efforts.

Derived from these tests, several memory leakages were detected in the CBR execution. More concretely, the breakdown point of the memory consumption is caused by the continuous execution of EPANET to acquire the energy consumption of the scenario (even though the energy was calculated in previous phases of the CBR). For that reason, this last version of the CBR has a remodelled "*retrieval*" phase.

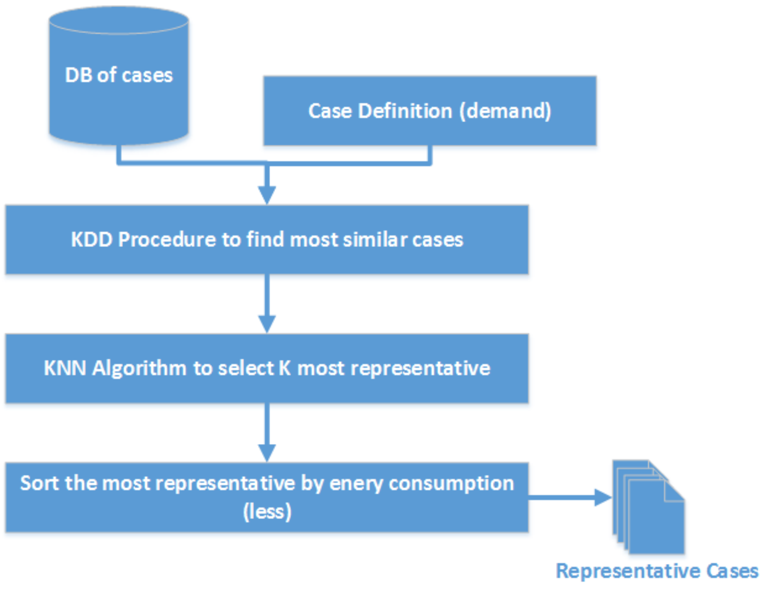
DB of cases
Case Definition (demand)
KDD Procedure to find most similar cases
KNN Algorithm to select K most representative
Sort the most representative by enery consumption (less)
Representative Cases

Figure 6 "Diagram of the new Retrieval phase"

The new "*Retrieval*" phase (see Figure 6) consists of acquiring most similar cases from the database of cases according to the "*K*" parameter defined for the KNN algorithm. Once the most similar cases are retrieved, they are sorted with respect to the past-lived energy consumption (from lower to higher). In order to assure the availability of an energy consumption parameter in the database of cases, for those representative cases, the energy consumed has been calculated in the hydraulic model (EPANET). Then, this energy consumption has been introduced in the SQL file designed to initialize the database of cases. For the rest of cases, those that are learnt during the execution of the CBR, the energy consumption will already be present in the database of cases given that it is calculated during the CBR adaptation and learning step.

The improvements in the "*Retrieval*" phase have enhanced the CBR, by removing unnecessary hydraulic model simulations that generate huge consumption of memory. Furthermore, this decision will also permit to save computational time in larger hydraulic scenarios (reducing the computational time in several minutes and even hours).

3.1.2 CBR Enhancements from the Deployment in the Demo-sites

During the deployment of the CBR in the demo-sites, several changes in the implementation have been performed. These changes refer to the incorporation of configuration files to make the installation of the

CBR easier. Furthermore, the CBR has been enhanced during the deployment in the demo-sites by returning more detailed parameters from the hydraulic model according to the visualization in the OMP.

Regarding the configuration of the CBR in the demo sites, the CBR has been improved with the incorporation of a configuration file ("*.conf*" file). This configuration file (see Listing 18) is a Java properties file (Properties) composed by "*keys*" and "*definition of the key*". On the one hand, the "*keys*" (bold coloured) refer to specific words/tags that are used in the code to define several files, folders and/or paths where the hydraulic model is located. On the other hand, the "*definition of the key*" defines the specific path, file or folder that the CBR will use to store or read information from, or execute specific tools.

```
# CONFIGURATION OF EPANET MODEL AND PROGRAM EXECUTION.
epanetEXEFile= C:\\Program Files (x86)\\EPANET2\\epanet2d.exe
# CONFIGURATION OF MYSQL
mysqlDriver= com.mysql.jdbc.Driver
mysqlLocation= localhost/cbr_knowledgebase
mysqlUser=root
mysqlPassword=

# CONFIGURATION OF RAPIDMINER
rapidminerProcessDir=MachineLearningProcess
rapidminerOutputsDir=C:\\RapidMinerCBRModels

#CONFIGURATION TEMP CBR FILES
cbrTemporalyDir=temp
```

Listing 18 "CBR configuration file"

In reference to other improvements performed during the deployment in the demo-sites, the information returned by the CBR regarding the evaluation and simulation procedures has been detailed (see Table 5). In reference to the returned information for the evaluation procedure, the returned case information is composed by (i) the *case id* as identifier of the case, (ii) the proposed *pump scheduling* (at 10 min time scale) based on the similar past cases, (iii) the *aggregated demand* for all sinks (at hourly time scale) and (iv) the *specific demand* for each of the defined sinks. This last time series is gathered from the forecasts provided by the demand management systems. With regards the simulation procedure, the returned information is: (i) the energy consumption generated based on the simulated pumping scheduling, (ii) the alerts and errors generated during the hydraulic model execution, (iii) the pump pressure (at 10 minutes time-scale) to describe the performance of each pump, (iv) the tank head (at 10 minutes time-scale) to show the water manager the behaviour of the main tank to accomplish the predicted demand and (v) the aggregated pump pressure (10 minutes time-scale) by each water work.

Returned Information by the CBR	
Evaluation Procedure	**Simulation Procedure**

Case ID	*Energy Consumption of the simulation*
Pump Scheduling	*Alerts and Errors generated by EPANET*
Aggregated Demand for the whole scenario	*Pump Pressure obtained from the simulation*
Specific demand for each defined Sink	*Tank Head (Level) simulation*
	Aggregated pump pressure by each water work

Table 5 "CBR output in both evaluation and simulation procedures"

As a conclusion, the enhancements performed in the CBR during the deployment have been used to make easier the integration of the CBR with the whole architecture using the SOA-MAS as a bridge. Furthermore, the CBR is able to provide detailed information regarding the evaluation and simulation procedures by returning relevant information about the scenario simulation using certain pump scheduling and detailed information regarding the proposed pumping scheduling by the CBR.

3.1.3 CBR Enhancements in the "Adaptation and Evaluation Phase"

The "*Adaptation and Evaluation*" phase has been enhanced in terms of the simulation functionality of the CBR. As depicted in the Deliverable 4.2 on Section 4.5 in page 48, the simulation engine permits the water distribution manager to perform changes over the proposed pumping scheduling obtained in the evaluation phase in order to make this proposed pumping scheduling fit better with the actual situation.

One of the main weaknesses during the simulation phase is the incorporation of rules inside the INP file of EPANET. These introduced rules force the simulation engine to accomplish some mandatory restrictions. This enforcement can even change the pumping planning or other configurations of the distribution network. To deal with this situation, the water manager always receives some warning messages referring to the changes performed over the pumping scheduling instead of crucial errors. A first step toward the enhancement of the simulation engine is to incorporate the capability of providing more realistic results over the simulation. To achieve this goal the defined rules in EPANET have been removed and a RBR over policy and infrastructure constraints have been introduced.

The main aim of the RBR is to check some policy and infrastructure constraints of the water distribution network based on the resultant information from the simulation generated by the hydraulic tool (EPANET).

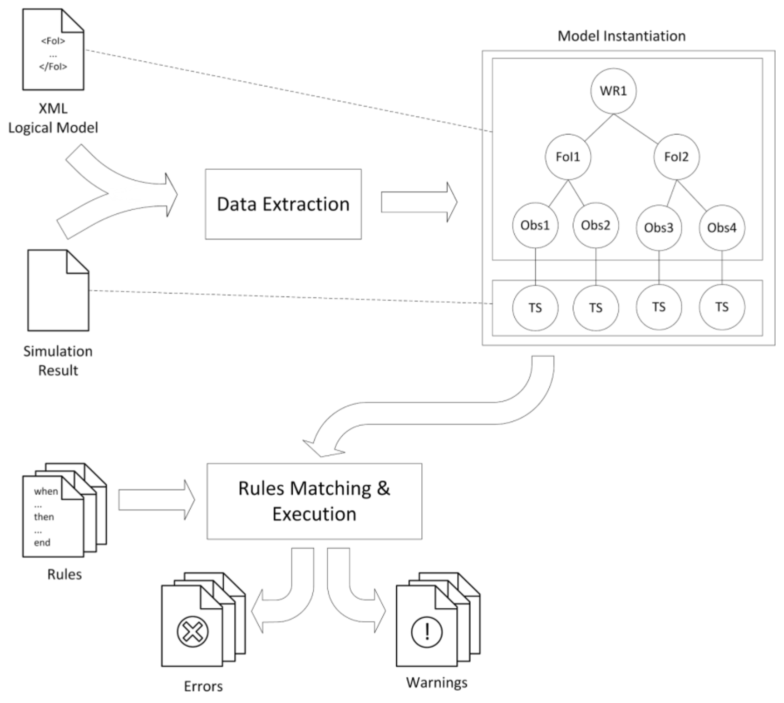

Figure 7 "Information flow of the RBR applied to the CBR"

Figure 7 depicts the process of the RBR applied to the CBR. Mainly, this process can be summarized into four steps: (i) the Data Extraction aimed at retrieving simulation information from the EPANET output file and an XMML file, (ii) Model Instantiation that uses the extracted information to populate Drools environment with facts; (iii) the Working Memory instantiation that combines the generated facts with the rules in order to introduce rules into the Working Memory for their posterior execution; and (iv) Drools Rules Matching and Execution process execution aimed at executing rules until no rules were available in the Working Memory (rules can activate or deactivate new rules). As a result, Error and Warning messages are provided to the water distribution manager.

According to the description of the RBR, its main objective is to apply the water distribution manager experience over a set of facts that represent an actual situation of the distribution network. The RBR incorporates a set of rules that represent policy and infrastructure restrictions which is called Rules Repository. These rules were defined in the Deliverable 4.1 in Section 3.2.1 on page 72. As an example of these rules, the Listing 19 depicts the rule GEN2. This rule enforces the pressure in all the pumps within the distribution network to be in range of 4 and 6 bars all the time. The rule GEN2 is based on the expert knowledge and its goal is to maintain the pressure of the network to avoid pipe sedimentation that can produce water leakage.

GEN2: The pressure in the supply network has to be between 4 bar and 6 bar (400 kPa and 600 kPa) everywhere at all times.

Listing 19 "Example of a rule from Rules Repository (rule GEN2)"

All natural language rules have been translated into DRL. This translation is not an automatic procedure, and it has to be done using the same rules generation process that the one followed in Deliverable 4.2 Section 3.2. DRL rules contain two differentiable parts, the LHS or conditional part (when part in Drools) and the RHS or action part (then part in Drools). Using the same example as before, the result of the conversion of rule GEN2 (Listing 19) into Drools define a rule that is fired whenever any pump from a certain water work does not satisfy a pressure range. In general, water managers write the rules focusing on the objective that has to be accomplished, for instance rule GEN2 states that the pressure value has to be between a certain range. To translate the rules and in order to provide warning and alarm messages when restrictions are not satisfied, rules are coded in order to be fired whenever the range is violated and not when the model accomplishes the desired values/parameters. The above mentioned situation happens in GEN2 rule and it can be seen how in Listing 19 water manager states a pressure value to satisfy and in Listing 20 the translation states that the rule has to be fired whenever that range is not satisfied.

Listing 20 shows a translation example of GEN2 rule. The main restriction is represented in line *"$PressureTVP: TimeValuePair(timeSeriesObservation==$TSO , (value>6 || value<4))",* which says that if a *"TimeValuePair"* belongs to a certain kind of *"timeSeriesObservation*" and also is outside the range of 4 and 6 then it satisfies this part of the rule. The rest of restrictions of the rules represent the pertinence of *"TimeValuePair*" to the values of the desired pumps. This pertinence is described hierarchically in a WaterML2 file. So, from top to bottom (from more general to more specific) one can see how in *"$SN: WaterResource(id=="SupplyNetwork")"*, the RBR must select the *"WaterResource*" with *"id"* equals to *"SupplyNetwork"* and stores the result in a variable called *"$SN"*. Then, in line *"$WW: WaterWork(parent == $SN)"* the RBR must select all *"WaterWorks"* which its parent is the result stored in *"$SN"* and finally the result is stored in *"$WW".* In line *"$FoI: FeatureOfInterest(waterResource == $WW)"* the RBR must select all *"FeatureOfInterest"* which *"waterResource"* is any of the results stored in *"$WW"* and it stores the result in *"$FoI"*. In line *"$Pressure: Observation(feature==$FoI, phenomenon == PRESSURE, procedure == SIMULATION)"*, the RBR must select all *"Observations"* which feature is any of the results in *"$FoI",* but also its phenomenon must equals *"PRESSURE"* and its procedure must equals *"SIMULATION".* In this rule, *"PRESSURE"* and *"SIMULATION"* variables are constants. The result of the restriction is stored in *"$Pressure".* In line *"$TSO: TimeSeriesObservation(observation == $Pressure)"*, the RBR must select all *"TimeSeriesObservation"* which observation is any of the results in *"$Pressure",* the result is stored in *"$TSO".* Each fact within the Working Memory that satisfies all the restrictions previously explained, violates GEN2 rule and the RBR fires the RHS part of the rule. This RHS part is line *"insert(new RuleError(String.format("Pump %s has a value of %f on %s", $FoI.getId(), $PressureTVP.getValue(), $PressureTVP.getTime().toString())));",* and it inserts a *"RuleError"* with an specific message to the Working Memory. When all rules are fired, and after the RBR process is finished, the Inference Engine inspects the Working Memory retrieving all *"RuleErrors"* and *"RuleWarnings"* which are delivered to the water manager.

```
rule "GEN2"
when
 $SN: WaterResource(id=="SupplyNetwork")
 $WW: WaterWork(parent == $SN)
 $FoI: FeatureOfInterest(waterResource == $WW)
 $Pressure: Observation(feature==$FoI, phenomenon == PRESSURE, procedure ==
SIMULATION)
 $TSO: TimeSeriesObservation(observation == $Pressure)
 $PressureTVP: TimeValuePair(timeSeriesObservation==$TSO , ( value>6 ||
value<4))
then
 insert(new RuleError(
 String.format("Pump %s has a value of %f on
%s",$FoI.getId(),$PressureTVP.getValue(),
$PressureTVP.getTime().toString())));
end
```

Listing 20 "Example of a rule translation (translation of rule GEN2)"

Extending the GEN2 rule example, the general process results in a translation of the whole expert knowledge into Drools Rules that are stored in the Rules Repository. Then, the information from simulation is extracted from EPANET output file and it is represented as elements of a WaterML2 file. This representation is converted into facts which are finally introduced in the Working Memory. Those facts that match the LHS of any of the rules in Rules Repository fire the RHS part of the rule. Generally, the RHS part result in the introduction of new facts of the kind "*RulesError*" or "*RulesWarning*".

The WaterML2 file mainly describes water resources and observations involved in the rules. This WaterML2 file is provided by the SOA-MAS that indeed acquire this information from the WatERP-KnowledgeBase. As an example, in Listing 21 there is a snippet from an XML file that represents a possible instantiation of the logical model. In this particular case, there is a "*WaterResource*" with "*Id*" "*DW_WW*". This water resource has two features of interest "*DW_WW_MP1*" and "*DW_WW_MP2*". Each "*FeatureOfInterest*" has an "*Observation*" "*DW_WW_MP1_Pressure*" and "*DW_WW_MP2_Pressure*" that observe the "*Phenomenon*" "*Pressure*" with the "*Simulation*" "Procedure" of "*DW_WW_MP1*" and "*DW_WW_MP2*" respectively. Those observations refer to the table "*Node*" and column "*Pressure*" of the simulation result file (see Listing 22).

```
<WaterResources>
 <WaterResource id="DW_WW">
  <FeaturesOfInterest>
    <FeatureOfInterest id="DW_WW_MP1">
     <Observation id="DW_WW_MP1_Pressure" phenomenon="Pressure"
procedure="Simulation">
      <TimeSeries table="Node" elementId=" P1_DW1" column="Pressure"/>
     </Observation>
     ...
    </FeatureOfInterest>
    <FeatureOfInterest id="DW_WW_MP2">
     <Observation id="DW_WW_MP2_Pressure" phenomenon="Pressure"
procedure="Simulation">
      <TimeSeries table="Node" elementId=" DW_WW_MP2" column=" Pressure"/>
     </Observation>
     ...
    </FeatureOfInterest>
   <FeaturesOfInterest>
  <ParentWaterResource id="SupplyNetwork"/>
    </WaterResource>
 ...
</WaterResources>
```

Listing 21 "Snippet of an XML logical instantiation example"

The EPANET output file is used to extract specific simulated measurements regarding the water resource management. The simulation result file describes the simulation results of the proposed pumps schedule in terms of energy consumption and status of the nodes and links that conform the distribution network (see Listing 22).

```
Node Results at 0:00:00 hrs:

----------------------------------------------

                    Demand      Head  Pressure

 Node                 m3/h         m         m

----------------------------------------------

 HBLUR             1018.83    163.69      3.89  Tank


Link Results at 0:00:00 hrs:

----------------------------------------------

                      Flow  Velocity  Headloss

 Link                 m3/h       m/s    /1000m

----------------------------------------------

  P1_HW1               0.00       0.00       0.00  Pump

  P2_HW2               0.00       0.00       0.00  Pump

  P3_HW3               0.00       0.00       0.00  Pump

  P4_HW4               0.00       0.00       0.00  Pump

  P5_MW1             676.03       0.00     -56.86  Pump

  P6_MW2               0.00       0.00       0.00  Pump

  P7_MW3               0.00       0.00       0.00  Pump

  P8_MW4               0.00       0.00       0.00  Pump

  P9_RW1               0.00       0.00       0.00  Pump

  P10_RW2              0.00       0.00       0.00  Pump
```

```
P11_RW3             0.00       0.00       0.00  Pump
P12_RW4          1763.09       0.00     -67.24  Pump
P39_DW4           434.30       0.00     -58.97  Pump
P38_DW3           205.37       0.00     -58.20  Pump
```

Listing 22 "Extract of an EPANET simulation result file"

Incoming information is extracted using the Data Extraction process that permit the results of the simulation of the CBR and the semantic model of the water distribution network to be instantiated into Drools facts. The Data Extraction process consists of two parsers, one for the EPANET output file and the other for the WaterML2 representation of the logical model. On the one hand, EPANET parser reads the EPANET output file and creates a data table with the values of each variable in each time instant indexed by the pump's identifier. On the other hand, the WaterML2 parser iterates over the hierarchy described in the file. Once it identifies a "*TimeSeries"* node, it extracts the "*table"* value*, "elementId"* value and *"column"* value. Using these extracted values, it can retrieve the required information from the data table obtained during EPANET parser execution. Once this process finishes, facts can be created and introduced into the Working Memory. As a result, the Drools Working Memory is feed with specific information that will permit to fire rules from the Rules Repository. Then, Drools execution process starts.

At the beginning of the Drools execution, the framework builds a RETE network using the rules in the Rules Repository. This network is used to match the incoming facts in the Working Memory. Once all the facts are introduced, those rules that have the LHS part satisfied are placed into the Agenda. The Agenda is in charge of resolving the RHS part execution. At this moment, rules with the same LHS part may be placed into the Agenda. For these cases, Drools has to sort the rules execution based on the priorities of the rule (criticality of the rule). After this process, rules are executed one by one. During this execution, some rules can introduce facts that fire new rules. Drools executes rules until no more facts are generated. At the end of the execution, the Working Memory contains a new set of facts as the kind *"RulesError"* and *"RulesWarning"* which contain a text that defines each Error or Warning. The *"RulesErrors"* represent the violation of some of the restrictions introduced by the expert knowledge. Similarly, *"RulesWarnings"* represent recommendations that are not satisfied in the actual simulation result.

As an example, Listing 23 shows the output of the execution of the rule GEN2 with an arbitrary model instantiation. As can be seen, the rule is fired two times: one because the pump "*DW_WW_MP1*" violates the range with a value of 6.5 bar; the other because the pump "*DW_WW_MP2*" violates the range with a value of 2.5.

```
2015-05-20 13:55:06,854 INFO [RulesAnalyzer] - ---------------------------------------------
----------------

2015-05-20 13:55:06,854 INFO [RulesAnalyzer] - ---------------------------------------------
```

```
----------------
2015-05-20 13:55:06,854 INFO [RulesAnalyzer] - --------------------START RULES EXECUTION-------
----------------
2015-05-20 13:55:06,854 INFO [RulesAnalyzer] - ------------------------------------------------
----------------
2015-05-20 13:55:06,854 INFO [RulesAnalyzer] - ------------------------------------------------
----------------
2015-05-20 13:55:06,861 INFO [RulesAnalyzer] - ---------------------Rule: INCLUDE GLOBALS------
---------------
2015-05-20 13:55:06,866 INFO [RulesAnalyzer] - ------------------------------------------------
---------------
2015-05-20 13:55:06,866 INFO [RulesAnalyzer] -
2015-05-20 13:55:06,866 INFO [RulesAnalyzer] -
2015-05-20 13:55:06,866 INFO [RulesAnalyzer] - --------------------------Rule: GEN2------------
--------------
2015-05-20 13:55:06,869 INFO [RulesAnalyzer] - INSERTED: ERROR: Pump DW_WW_MP1 has a value of
6,500000 on 2012-01-01T03:00:00+01:00
2015-05-20 13:55:06,869 INFO [RulesAnalyzer] - ------------------------------------------------
---------------
2015-05-20 13:55:06,869 INFO [RulesAnalyzer] -
2015-05-20 13:55:06,869 INFO [RulesAnalyzer] -
2015-05-20 13:55:06,869 INFO [RulesAnalyzer] - --------------------------Rule: GEN2------------
--------------
2015-05-20 13:55:06,869 INFO [RulesAnalyzer] - INSERTED: ERROR: Pump DW_WW_MP2 has a value of
2,500000 on 2012-01-01T02:00:00+01:00
2015-05-20 13:55:06,870 INFO [RulesAnalyzer] - ------------------------------------------------
---------------
```

Listing 23 "Example of GEN2 rule execution"

As a conclusion, the incorporation of a RBR in this phase of the CBR has permitted to efficiently check the accomplishment of the policy restriction and infrastructural constraints that lives the water distribution network of SWKA. This restrictions and constraints can be easily adapted to other water distribution networks by modifying the data sources (facts) and updating the rules repository (rules adapted to the newest water distribution). Therefore, the simulation is enhanced and more detailed information is provided to the water distribution manager in order to select and adapt the more suitable case for the lived water distribution scenario.

3.2 Summary of the Final Version of the CBR

This section is focused on briefly describing the final version of the case-based reasoning (see Figure 8). The main reason to implement the CBR relies on a lightweight model capable of offering efficient pump schedules for a water distribution network. These efficient pump schedules are achieved by learning from past cases and adapting the similar past solutions to newer situations. The comparison between past and current situations is performed using the "*case*". A "*case*" is a set of variables that permit to abstract certain situation into "*Case Description*" (variables for problem definition) and "*Case Solution*" (variables for solving certain problematic). In the case of water distribution, the "*case*" has been defined to represent the water distribution management (see Table 6). Then, the "*Case Description*" is composed by those variables that permit to discriminate situations in water distribution as the demand level (specific and aggregated) and tank head. The "*Case Solution*" are the variables that represent the solution to the problematic. In the water distribution management, the case solution is composed by the pump scheduling, the energy consumption produced by the pumps where they are satisfying the demand, the pumping pressure generated by the past pumping experience based on the demand, and the solution validity to indicate that the provided solution accomplish infrastructural and quality constraints.

Case Definition	Variable name	Variable measurement unit	Nature of variable	Type of variable
Case Features and Problem	Aggregated Demand	m^3/h	Continuous variable	Array of Doubles composed by the demand value during 24h [1x24]
	Specific Demand	m^3/h	Continuous variable	Array of Doubles composed by the specific demand values during 24h [3x24], where 3 is the number of sink zones in Karlsruhe.
	Tank Volume	m^3	Continuous variable	Array of Doubles composed by HB Luss tank during 24h [1x24]
	Current energy consumption	kwh	Continuous variable	Unique value [1x1]
Case Solution	Pumping Scheduling	Boolean	Discrete variable	Array of Booleans composed by 15 pumps during 24h [15x24]
	Pumping Pressure	m^3/h	Continuous variable	Array of Doubles composed by the pumping pressure value during 24h for the 15 pumps[15x24]
	Solution Validity	Boolean	Discrete variable	Unique value [1x1]
	Energy Consumption	kwh	Continuous variable	Unique value [1x1]

Table 6 "Description case table for SWKA case"

In the CBR, the lived experiences (past cases) are stored in a database of cases. This database of cases has been implemented in MySQL as depicted in Deliverable 4.3 in Section 2.1 on page 19. In order to improve the initial reasoning of the CBR, this database of cases has been initialised with representative cases for SWKA water distribution network. These representative cases are classified by demand in low demand cases, mid demand cases and high demand cases. In several studies performed in SWKA, these cases will permit to achieve between *6-8%* of energy savings (see Deliverable 4.2 in Section 4.2 on page 37). In the CBR, the lived experiences (past cases) are stored in a database of cases. This database of cases has been implemented in MySQL as depicted in Deliverable 4.3 in Section 2.1 on page 19. In order to improve the initial reasoning of the CBR, this database of cases has been initialised with representative cases for SWKA water distribution network. These representative cases are classified by demand in low demand cases, mid demand cases and high demand cases. In several studies performed in SWKA, these cases will permit to achieve between *6-8%* of energy savings (see Deliverable 4.2 in Section 4.2 on page 37).

When a new case (current situation) is defined in the system, the database of cases is used to acquire more similar past cases according to the incoming case ("*Retrieval*" phase). Most similar selected cases are studied to select the most adequate cases based on the energy consumption ("*Reuse*" phase). Further, these cases are adapted using the water manager and the simulation capabilities of the CBR to finally apply the case and update the database of cases.

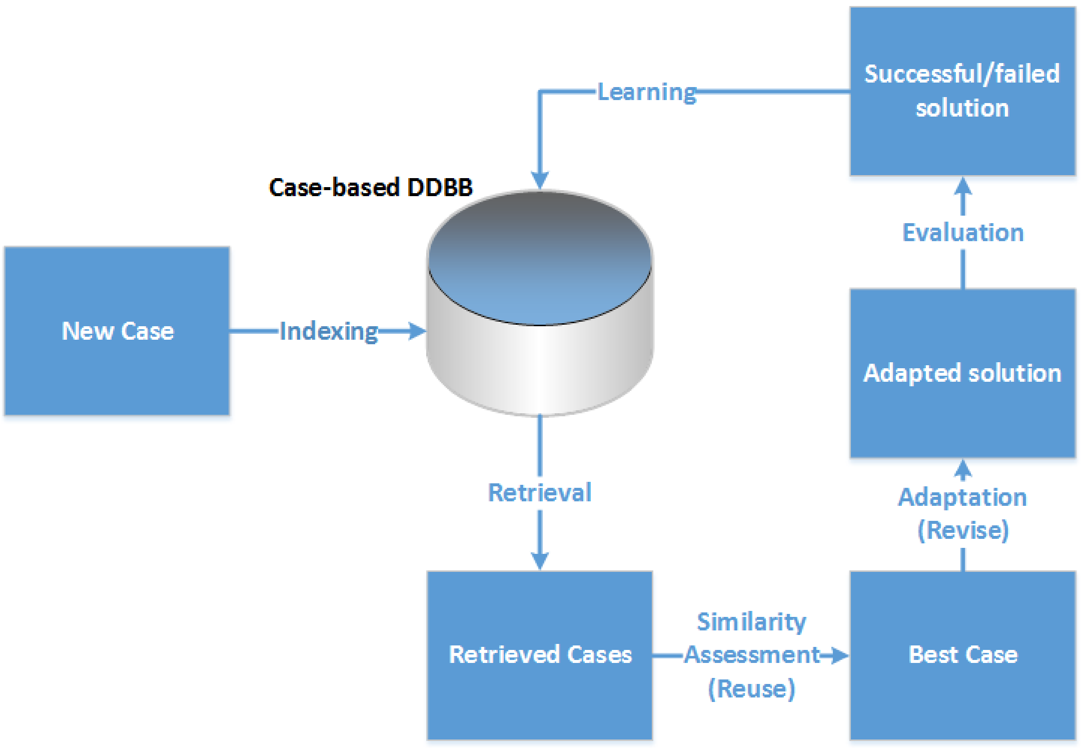

Figure 8 "CBR process"

3.2.1 Retrieval Phase

The retrieval phase was motivated by the analysis of the past experience in order to select most similar cases to the new defined case ("New Case"). Based on the case definition (see Table 6), the similarity between cases is understood as the comparison of the demand time series, main tank level behaviour and the expected energy consumption. To compare these variables, the KNN procedure calculates the shortest K distances (most similar K cases) to the incoming case based on a similarity function. The similarity function calculates the rate of likeness between cases using a weighted similarity for the demand time series and tank level behaviour time series and the energy consumption (see Deliverable 4.2 in Section 4.4 on page 46). In order to calculate the similarity, a KDD procedure was constructed following the procedure described in Figure 9. This KDD procedure was composed by a time-series windowing technique to adapt the collected time series to the process; and a clustering technique to group the time series by taking into account the variance, mean and covariance of the time series (see D4.2 in Section 4.2 on page 41). As a result of this process, a vector composed by the clustering results regarding with the demand and main tank behaviour and the expected energy consumption, are returned. Then, the weighted similarity make use of this generated vector to compare the demand and tank volume clusters with the incoming cases and return the most similar cases.

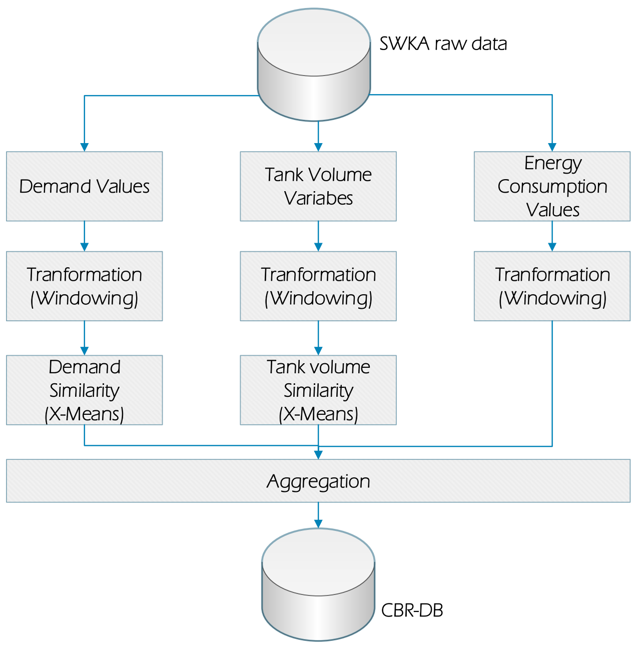
SWKA raw data
Demand Values
Tank Volume Variabes
Energy Consumption Values
Tranformation (Windowing)
Tranformation (Windowing)
Tranformation (Windowing)
Demand Similarity (X-Means)
Tank volume Similarity (X-Means)
Aggregation
CBR-DB

Figure 9 "KDD procedure"

The "Reuse" phase of the CBR uses the similar K retrieved cases in order to estimate non-energy efficient cases from the most similar ones (Deliverable 4.3 in Section 2.2 on page 21). To obtain more relevant cases, the energy consumption extracted from each case has to be greater than 0. In case the energy consumption where less or equal than 0, a non-suitable solution produced by a wrong learning experience is retrieved. Therefore, this case has to be removed from the solution set. The cases from the final solution set is sorted in terms of energy efficiency, from those with less energy consumption (greater than 0) to those with higher energy consumption. As a result of this phase, the water distribution manager receives a set of cases ordered by energy consumption.

3.2.2 Adaptation and Evaluation Phase

The “Adaptation and Evaluation Phase” is focused on selecting one of the most similar cases and adapting this case in order to establish a suitable solution for the current situation. With this aim, several simulations with a hydraulic tool (e.g. EPANET) are performed using the simulation functionality of the CBR. The water distribution manager selects one of the cases from the returned list of cases and performs continuous modifications and simulations in an iterative process over it. During this process, the water manager applies its knowledge to adapt the case (pumping schedule) and simulates the results using an hydraulic tool (e.g. EPANET). The adaptation of the cases might lead to pumping schedules that break infrastructural and water quality constraints or best practices (see Section 3.1.3) making those pumping schedules invalid or sub-optimal. In order to detect these problems, the results of the EPANET simulation are validated against the water distribution manager expertise using an RBR procedure. This procedure, using the WatERP-KB, the output file of EPANET and the water manager expertise expressed in form of rules (DRL), generates a set of Warning and Errors to be considered by the water distribution manager performing the adaptation.

Once the water distribution manager selects and adapts the case, this case is stored into the CBR-DB in order to perform the learning of the CBR. Hence, the learning in the CBR-DB is driven by the water distribution manager that update the previous experience with case modification. This aspect permits the learning in two ways: from the water distribution manger to the CBR (case modification and adaptation) and from the CBR to the water distribution manager with the case selection according the predicted demand (retrieve phase) that provides accurate scheduling where water manager applies general solutions.

4. Conclusions and Future Work

This section summarizes the conclusions obtained from the work developed under Task 4.3 "*Simulation and tuning of the WDSS rule set and inference engine*" and Task 4.4 "*WDSS adaptation*" for the period of time correspondent to month 24 (M24) until moth 33 (M33) of the WatERP project. As described during this document, the work has been focused on the development of the Inference Engine by completing the RBR module and also enhancing de CBR module. To complete the RBR module, changes have been performed to the first prototype presented in Deliverable 4.3 (changes in the working memory population procedure) and the remaining rules have been completed. Additionally, the RBR module has been exposed as a service integrated inside the SOA-MAS architecture.

The WatERP-DSS's main objective is to support the manager's decisions in matching water supply and demand, optimizing energy efforts, minimizing storage and treatment while ensuring that consumer and environmental needs are met. Therefore, the main objectives of the WatERP-DSS are *(i)* to generate suitable recommendations and alerts that enhance the water manager's daily operations by avoiding water resource mismanagement and inefficient energy strategies in the water supply and distribution chain; and *(ii)* to reduce the economic impact associated with the mismanagement of water resources.
The progress of the WatERP-DSS requires the development of the Inference Engine, which includes the complete implementation of the RBR and CBR modules to accomplish the mentioned objectives. Regarding the RBR case, efforts have been made to: (i) improve the Working Memory population procedure, (ii) adapt the rules to ontology changes, (iii) implement the remaining rules, (iv) integrate the RBR with the SOA-MAS architecture. In order to improve the Working Memory population procedure, Alibaba has been replaced by an ad-hoc ontology-to-Java mapping procedure that overcomes all the problems faced with the initial approach. The new approach uses SPARQL queries to obtain triples from the ontology which are later on converted into Java objects and enhanced with information (TimeSeries) coming from WaterML2 formatted documents. Due to changes made to the ontology (naming of entities and relations) the already implemented rules have been revised and adapted with those changes. The remaining rules have been implemented using a newly designed mechanism (*"activationPatterns"*). This mechanism allows to instruct Drools to execute the rules following a certain pattern in a sequential manner. This technique permits to model the water manager decision process using Drools. Lastly, thanks to the publication of the RBR as a web-service, the engine is now able to communicate with the SOA-MAS architecture.

As a result of all the work done, the final implementation of the RBR is capable of receiving simulation request from the SOA-MAS architecture and then applying the water manager knowledge in order to provide an efficient water allocation strategy. This expert knowledge application has been developed using the Drools engine, which provides a powerful interface to build a Rules Repository where restrictions and recommendations are written. Taking into account the logical representation of a real scenario by means of the Drools Working Memory, Drools is in charge of executing the satisfied set of rules, which ultimately result in a water allocation strategy that minimizes the water misuse.

In the CBR case, despite the fact that the development was already finished in this Deliverable some enhancements have been introduced. During the test of the CBR some memory problems were detected within the "*Retrieval*" phase. This phase consumed a lot of memory as it made an intensive use of the hydraulic simulation tool (EPANET). To tackle this problem the "*Retrieval*" phase has been modified in order to have less dependency on EPANET tool. The new "*Retrieval*" phase acquires most similar cases from the database of cases according to a parameter defined in the new procedure. Once the most similar cases are retrieved, they are sorted with respect to the past-lived energy consumption (from lower to higher).

In order to improve the "*Adaptation and Evaluation*" phase within the CBR module, an RBR has been integrated. This RBR applies the expert knowledge of the water manager over the results of the hydraulic simulation tool. This enhancement permits to apply a wide range of restrictions and recommendations and also to adapt the out coming results to better communicate the conclusions to the water manager. To adapt the RBR, two parsers have been developed: one to extract data from simulation tool (EPANET) and the other to parse the WaterML2 logical model. Moreover, the rules from water manager expertise have been implemented and the RBR has been integrated to the actual version of the CBR.

In summary, this Deliverable has presented both final versions of the CBR and the RBR. **In the case of RBR the changes made to the initial prototype have been described and the final version the architecture has been explained. Through this development, it has been shown that the RBR is a suitable tool to apply the water manager's expert knowledge over the actual status of the supply network. In the case of the CBR, it has been shown that it is a suitable tool to solve the complex problematic of distributing water into a city by using the lived experience to recommend suitable cases that can be applied to satisfy current water demand.**

5. References

Bali, M. (2013). Drools JBoss Rules 5.X Developer’s Guide.

Forgy, C. L. (1982). Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*. doi:10.1016/0004-3702(82)90020-0

International Organization for Standardization. (2010). *Geographic Information: Observations and Measurements* (p. 57). Geneva.

OSGeo-Live. (2011). 52°North WPS Quickstart. Retrieved from http://live.osgeo.org/en/quickstart/52nWPS_quickstart.html

Prud’hommeaux, E., & Seaborne, A. (2008). SPARQL Query Language for RDF. *W3C Recommendation*, *2009*, 1–106. doi:citeulike-article-id:2620569

Rossman, L. A. (2000). EPANET 2: users manual. *Cincinnati US Environmental Protection Agency National Risk Management Research Laboratory*, *38*, 200. doi:10.1177/0306312708089715

Sesame Community. (2015a). Programming with Sesame. Retrieved May 05, 2015, from http://rdf4j.org/sesame/2.8/docs/programming.docbook?view

Sesame Community. (2015b). Sesame project web page. Retrieved May 08, 2015, from http://rdf4j.org/

Sottara, D., Mello, P., & Proctor, M. (2010). A configurable Rete-OO engine for reasoning with different types of imperfect information. *IEEE Transactions on Knowledge and Data Engineering*, *22*, 1535–1548. doi:10.1109/TKDE.2010.125