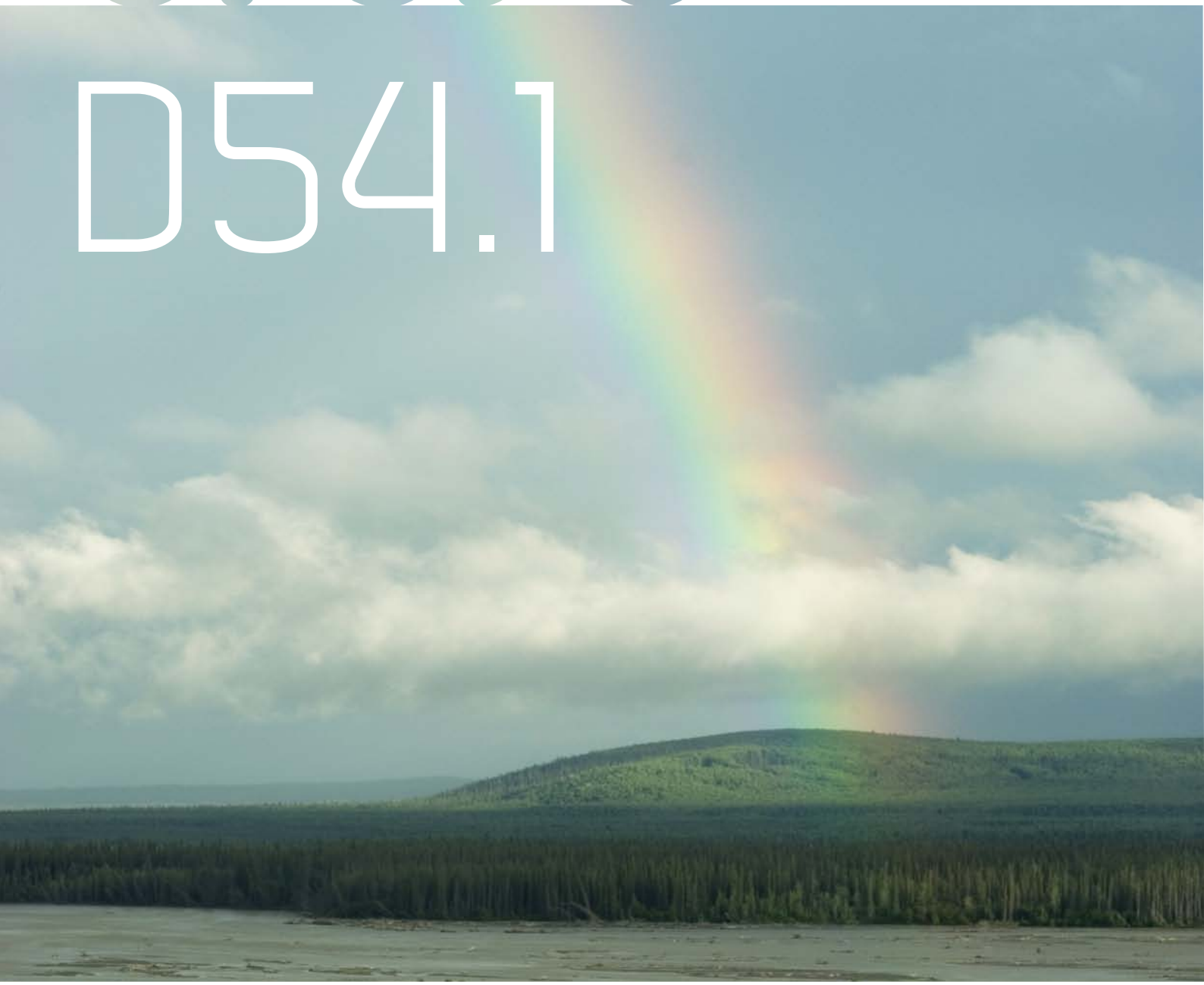
Integrated Decision Support Framework

M.S.MORLEY
Z.KAPELAN
D.A.SAVIĆ

trust

D54.1



TRANSITIONS TO THE URBAN WATER SERVICES OF TOMORROW

TRANSITIONS TO THE URBAN WATER SERVICES OF TOMORROW

Deliverable D54.1: Integrated Decision Support Framework

M.S.Morley, Z.Kapelan & D.A.Savić
University of Exeter, Centre for Water Systems

Abstract

An Integrated Framework for the development of a Decision Support System (DSS) is described, facilitating strategic decision-making for the long-term city metabolism planning problem. A novel methodology for comparing and selecting alternative solutions is presented employing Multi-Criteria Decision Analysis and multiple scenarios handling resulting in a system able to deal with uncertain futures, complementing the modules developed in WP54 and using the same platform (AWARE-P) as the other software applications delivered in WA5. The DSS assists in the decision making process by managing problem definition, structuring/analysis and solving.

This document outlines the Integrated Decision Support Framework upon which the DSS will ultimately be delivered. Succinctly, this is the software environment in which the DSS will be constructed, describing the concepts, data structures, data flows and associations that will be required to bring it to fruition.

Table of Contents

List of Figures

List of Tables

Introduction

The Decision Support System (DSS) developed in WP54 seeks to support long-term, strategic-level planning of Urban Water Systems at the city/system level. This will be achieved through a novel methodology for comparison and selection of alternative solutions, within the framework of long-term transition paths, and amidst multiple decision criteria. The principal output of the work area is a DSS incorporating the multicriteria decision analysis system able to deal with uncertain future scenarios, complementing the modules developed in WP54 and using the same software platform (AWARE-P) as the other software applications delivered in WA5. This application will be also be prepared for standalone use as well as being integrated in the TRUST software platform. The DSS seeks, where possible, to assist in all principal phases of the related decision making process including problem definition, problem structuring/analysis and problem solving. The degree of sophistication of these decision-making tools will adapt to the complexity of the options to be compared, to the availability of data and to the skills and availability of human resources the end-user is willing and able to invest in the implementation and application of this tool.

This document outlines the Integrated Decision Support Framework upon which the DSS will ultimately be delivered. Succinctly, this is the software environment in which the DSS will be constructed, describing the concepts, data structures, data flows and associations that will be required to bring it to fruition.

DSS Concept

The DSS methodology and software architecture presented here employs novel techniques to facilitate the generation and comparison of user-developed Intervention Strategies and to permit the interactive investigation of the behaviour of these Strategies and their effect on the Urban Water System (UWS).

The principal functions of the DSS will be to:

- *accept* and *store* the input data from the user to define the problem to be solved, including defining one or more optional intervention strategies over some analysis horizon, multiple criteria used of evaluation of these strategies and the associated different scenarios of analysis;
- *enable* the user to *assess* the impact of above intervention strategies on the UWS performance;
- *enable* the user to *identify* the preferred intervention strategy by either *ranking* the optional intervention strategies (using MCDA) and/or using "what-if" type analysis to interactively modify and evaluate these strategies.

- *visualize* and *store* the results obtained (e.g. to visualize the multiple criteria profiles over the analysis horizon for the top ranked solution(s), together with e.g. the map denoting which interventions are applied and where for this particular solution, etc.)
- *enable* the user to *access* a simple, context-sensitive help/guidance system.

DSS Components

This section provides an overview of each of the conceptual components identified in the WP54 description together with their links. In the proposed design, several changes have been made to the functional relationships of the proposed modules and concepts – relative to those envisaged in the proposal– as well as the nomenclature. A mapping between the envisaged and proposed components is provided in the Software Architecture section. As outlined in the proposal, the primary Graphical User Interface (GUI) to the DSS will be based on the AWARE-P platform (www.aware-p.org), giving the unified feel and look to the DSS and the other planning software developed in WA5 (D53.2).

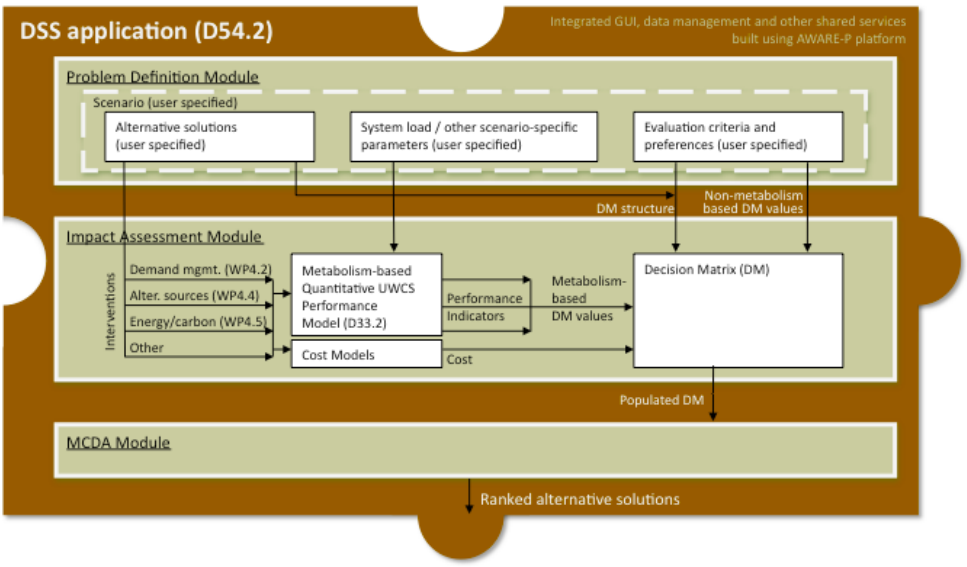


Figure 1: D54.2 The DSS for long-term strategic-level planning

Figure 1 illustrates the form of the Decision Support System as specified in the proposal which comprises three principal modules. Before identifying the setting of these modules in the revised design which is presented in Figure 2 and in more detail in Appendix B, the functionality of each module is described since this remains largely unchanged.

Problem Definition module

This module enables the user to define, in a structured way, the strategic planning problem analyzed. More specifically, the module enables the user to define a number of alternative solutions (i.e. intervention strategies) to be analyzed, together with the multiple criteria that will be used to evaluate them, all under a number of different, user-specified scenarios. An alternative solution is defined here as a single intervention strategy to be applied to the city's urban water systems over some long-term analysis horizon (e.g. 30 years). The interventions in this set will be selected by the user from a pre-specified list of different individual intervention types stored in the relevant DSS database and by providing additional information on individual intervention quantity and timing.

The evaluation criteria (also selected by the user from the pre-specified list created based on the work done in WA3-4) will be used to quantitatively assess each alternative solution analyzed. The criteria will be based primarily on the outputs of the Quantitative UWCS Performance metabolism-based USCW performance model (hereafter WaterMet^2) developed in WP3.3,i.e. various performance indicators related to the UWCS performance, either directly (e.g. total annual energy used) or indirectly (e.g. carbon footprint) and the associated intervention costs. In addition, the Problem Definition Module should allow the user to specify additional criteria that cannot be quantified by using the WaterMet^2 model (e.g. on a scale 1-5, 1 being the best and 5 being the worst, what is the social acceptance of the analyzed alternative solution, etc.). This functionality in the module also permits the incorporation into the DSS of Risk Indicators not handled directly by WaterMet^2 as well as additional costs that will need to be calculated to fully evaluate the model.

The concept of *scenarios* will be used here to represent different possible future climate, urbanization and other changes. This will be achieved by allowing the user to control, i.e. specify different values in different scenarios for a number of pre-selected model parameters (e.g. future population, different per capita consumption, historic/synthetic rainfall, percentage impervious area, air temperature, etc.), all defined over the long-term analysis horizon. A list of these parameters will be finalised in collaboration with WaterMet^{2}model developers.

Impact Assessment module

The Impact Assessment module will be developed to enable the quantification of the impact of each optional intervention strategy on the future UWCS performance. To undertake this, this module will make repeated use of the WaterMet^{2}model (D33.2) developed in WP33. The database of individual interventions modelled will be created and populated with different intervention types (e.g. construction of new rainwater harvesting storage) and the associated default cost models/parameters (e.g. cost per unit tank volume) and other data. Each intervention modelled will also have an associated list of WaterMet^2 model parameters that will be updated every time the impact of that intervention on the future city's metabolism is evaluated. The above information required will be obtained from the respective WA4 work packages. Once selected from the database, in addition to the above information, the user will specify the intervention size (e.g. 20ML of new tank storage)

and the area of its application (e.g. north zone of the city's distribution system). This will be repeated for every intervention of each alternative solution considered. The resulting UWCS performance (calculated by the WaterMet^2 model) and the corresponding intervention costs will then be used to populate the Decision Matrix. The structure of this matrix will be defined based on the information provided in the Problem Definition module. The rows in this matrix correspond to optional intervention strategies and the columns correspond to the multiple criteria used to assess these strategies.

The Impact Assessment module encapsulates the whole of the WaterMet^2 Metabolism Model as developed in WP33 along with further data structures necessary to couple that model to the DSS as required In addition, this module can execute the WaterMet^{2}metabolism model repeatedly to perform the evaluation for each of the assessment timesteps and to apply any variable scenarios.

Multi-Criteria Decision Analysis module

This module facilitates user selection of the best alternative solution (i.e. set of interventions) to be applied to the analysed city over some pre-specified, long-term analysis horizon (e.g. 2010-2040). This will be achieved by ranking the alternative solutions by applying the relevant Multi Criteria Decision Analysis (MCDA) method to the Decision Matrix data (see Tasks 54.3 and 54.2). Two well-known MCDA methods will be implemented in the DSS for this purpose: (a) the Compromise Programming (CP) method (Zeleny, 1973) and the Analytical Hierarchy Process (AHP) method (Saaty, 1980). The two methods were selected because of their widespread use (especially the AHP method) but also because they use different ranking technologies and, also, allow users to express their preferences in a different way. In the CP method, user preferences are specified as multiple evaluation criteria weights making this method more suitable for use by less experienced users. In the AHP method, user preferences are specified via the pairwise criteria-importance comparisons. This requires more experience to configure and employ the method. The DSS will enable the user to select the method to use when solving a particular problem, including the possibility to use both methods on the same problem and then compare results (e.g. to see if there an alternative solution that is ranked highly regardless of the MCDA method used).

Required DSS Functionality

The principal tasks required of the DSS comprise:

Environment Configuration

The user will be assisted in defining the Environment configuration – i.e. the definition of the problem to be analyzed (WP54.2). This assistance will take the form of:

- Defining a time horizon for the analysis, along with the intermediate times at which Interventions may take place.
- Defining Scenarios which comprise varying input parameters to the WaterMet^2 model or to Custom Indicators defined outside of WaterMet^2.
- Selecting the Evaluation Criteria from the list of available Indicators along with defining User Preferences that are to be used when ranking the proposed Intervention Strategies.

Baseline model evaluation

For the subset of Performance Indicators (normal, risk or cost) the WaterMet^2 model performance is evaluated (WP54.3). During evaluation, each of the scenarios defined in the Environment Configuration is applied in turn and Performance Indicators derived for each.

Intervention Strategy generation

The user will be helped to generate one or more Intervention Strategies by specifying a set of Interventions that are undertaken at the pre-determined times defined in the Environment Configuration. Each intervention is presented to the user in terms of a model input that it can influence, a time at which it takes place and the magnitude/nature of the intervention made.

Intervention Strategy evaluation

Through repeated execution of the WaterMet^2 model each Intervention Strategy is evaluated to determine its effect on UWCS performance. This is achieved by, firstly, applying each Scenario defined in the Environment Configuration in turn and also applying each Intervention in the Strategy in turn – at the appropriate timestep. This process results in a series of indicator values, for each timestep and scenario, representing the performance of the system.

Ranking

Having created two or more Intervention Strategies, the principal role of the DSS is to undertake an automatic ranking of the Strategies using a Multicriteria Decision Analysis (MCDA) technique (WP54.4). Two such techniques are implemented: Compromise

Programming (CP) and the Analytic Hierarchy Process (AHP) although the framework design should not preclude other techniques to be applied, including optimization. The ranking is performed according to the Evaluation Criteria that have been identified in the Environment Configuration and according to any specified User Preferences. The DSS should produce the ranked list of Intervention Strategies along with the metrics that have used in the ranking process.

Interactive modification and evaluation

The user will be supported in interactively modifying an Intervention Strategy and submitting it for re-evaluation. Any number of Intervention Strategies can be created by the DSS and existing Strategies can be cloned and modified to assist in "what-if?" analysis, allowing variations of Strategies to be submitted to the evaluation and raking processes.

DSS Software Framework

This section describes the design of the implementation of the Integrated Decision Support System Software Framework. An overview of the architecture is presented and contrasted to that outlined at the proposal stage and how best to implement the DSS in terms of programming language and interfacing terms. Each of the software constituents is described in detail in terms of its relationship to the other components as well as their interaction to fulfil the identified use cases. Finally, some conclusions are drawn as to the direction of the first steps in the implementation of D54.2 – i.e. the DSS software itself.

Implementation Specification

Platform/Development Language

The non-UI elements of the Decision Support System will be implemented in ANSI C++. The extant components of the WaterMet^2 Metabolism Model have been developed in C/C++. Given the tight integration required between the DSS and WaterMet^2, it is sensible to pursue the DSS development in the same language.

Employing ANSI C++ in the development should ensure that the DSS remains platform independent. During development, the DSS will be implemented on a Microsoft Windows platform although it is envisaged that the entire DSS will be able to be recompiled and run on any platform that provides suitable software support – including Linux. At the time of writing, the only external dependencies anticipated in the design of the DSS framework is the provision of an HTTP server - which implements the "Flexible Interface" outlined below – and access to a Relational Database Management System (RDBMS).

Extensibility is accommodated through the polymorphic characteristics of many of the concepts employed in the DSS design. For example, the abstract Ranking class can be derived from to provide alternative Ranking methodologies to complement the provided Multicriteria Decision Analysis (MCDA) approach. The sole constraint on these extensions is that they can be linked to C/C++ code through source or object files in some form.

Flexible Interface

The requirement to adopt the Baseform/Aware-P platform for the presentation (View) layer of the DSS enforces a clear separation between the Control and View layers of the application architecture – given the different programming languages employed by each. As the View layer may, indeed, be running on a different platform to the DSS it is necessary to adopt a control interface that is platform independent and sufficiently robust to handle the moderate data transfer requirements of the DSS. Whilst standards such as COM (Rogerson, 1997) and CORBA (OMG, 2011) are available, these would be difficult to adapt for use with the Aware-P platform without significant work on the platform side. Experience with a large-scale DSS integrating a number of disparate systems, developed as part of the

Neptune project (Morley *et al.,* 2009), has shown that a combination of HTTP requests and XML data transfer can effectively implement such an interface.

The key advantages to adopting such an interface are that both sides can adopt a very simple, invariant, API which passes XML data as required. It is not required for the DSS to expose a complex API which permits access to all of the data members within. Instead the complexity of the DSS data structures is embodied in the definition of an XML schema which is used to describe the data types and structures therein. The schema is a "living document" which can be modified by developers working on both sides of the interface as requirements become clearer during the implementation process. This can take place without the API to the DSS needing to be reworked at each juncture.

In addition to allowing greater platform independence, the use of HTTP/XML allows the UI and operational constituents to operate not only on different platforms but also on completely disparate systems. Thus individual instances of the DSS control mechanism may be run on remote computers as necessary to meet load requirements – though it should be noted that implementing a load management scheme is beyond the scope of this project.

Enforcing such a rigid separation between the User Interface and the functional components of the DSS further permits the DSS to be embodied in other settings if desired, such as a standalone application at a remove from the Aware-P platform. Indeed, for testing purposes and to demonstrate the efficacy of the approach as envisaged in the proposal, the DSS will initially be developed with a temporary, loosely connected Graphical User Interface (GUI) that will permit the interactive configuration of the DSS with the appropriate Environment settings and selection of Risk and Performance Indicators – using exactly the same XML interface as will be utilised by the deployed version of the DSS on the Aware-P platform. This exercise will also inform the design of the interface on the Aware-P platform.

Open Source

The source code for the DSS framework will be "open-sourced" – license type to be advised – which will permit developers to reuse and extend the functionality of the DSS through recompilation. The type of license adopted will have a bearing on the ability to extend the DSS on a commercial basis and it is recommended that a licensing strategy is adopted that does not preclude commercial exploitation whilst ensuring that the core functionality of the DSS is publically available and extensible.

The DSS is itself likely to make use of a number of open source software libraries to streamline the development process. It is important that the selection of these libraries does nothing to preclude the licensing of the DSS as a whole on more restrictive terms. At the time of writing, it is envisaged that the DSS will employ open source components for emitting and parsing properly formed XML data as well as managing access to the RDBMS (see the following section).

Database Requirements

The DSS has modest database storage requirements – largely as a repository for persistent data that it will be advantageous to preserve from one instance of the DSS to another – e.g. the library of available indicators of system performance. Data pertaining to the results developed by the DSS is formatted as XML data for the purposes of informing the user interface. This data could be stored in offline files if the need to do so is identified.

The Aware-P platform (Coelho & Vitorino, 2011) provides its own Data Management interface and, in the interests of avoiding unnecessary functional duplication, it is recommended that the DSS should use this as a data repository in the first instance. This is subject to a C/C++ callable interface being available to achieve this. In the event that this proves impossible or is unwieldy in practice, a local database will need to be used for the storage of this data. This may either take the form of an open source RDBMS (e.g. PostgreSQL(http://www.postgresql.org/), MySQL (http://www.oracle.com/mysql)) or in the form of local storage of XML data if this has acceptable processing overheads.

Software Architecture

An overview of the DSS software architecture is presented in Figure 2.

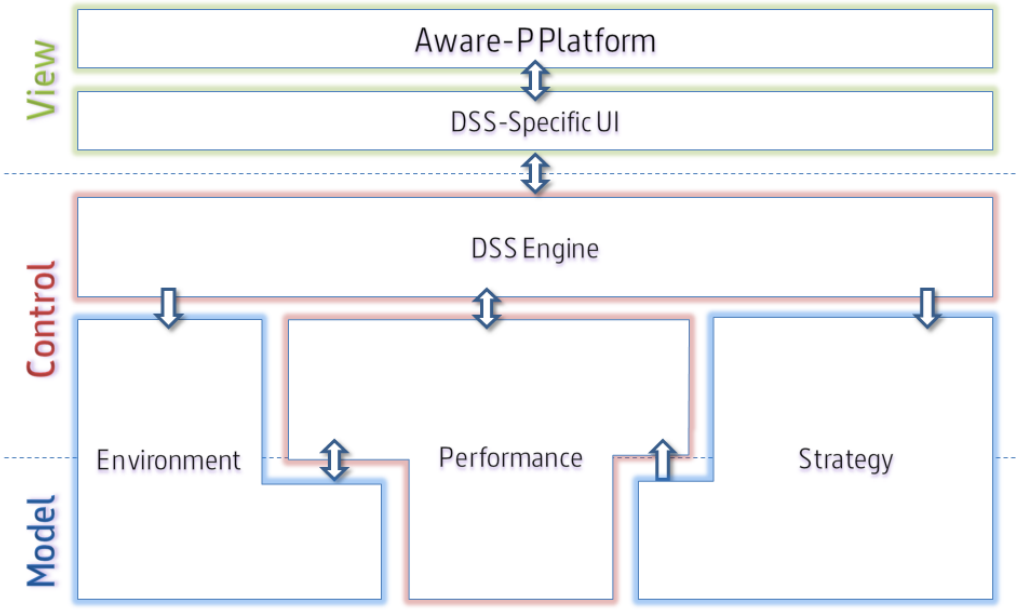


Figure 2: DSS Constituents

The software architecture is divided into three conceptual layers, Model, Control and View which correspond to Modelling Functionality, Operations and User Interface respectively. This document is largely concerned with the Control and Model layers of the application identified the above Figure. The View layer is to be implemented on the Aware-P platform and is subdivided into two conceptual units. The DSS-Specific UI comprises the dialogs and

displays specific to modifying the inputs of the DSS and rendering the results as, for example, tables. The top-level "Aware-P" platform element represents the advanced visualization services provided by the platform in terms of network display, charting etc.

Turning to the Control and Model layers, it can be seen that the nomenclature has been altered with respect to the decomposition of the DSS as presented in the proposal (Figure 1) to more closely align the key concepts with the packages in which they lie. Four key Packages are introduced that implement the DSS. An overarching DSS Engine package is employed as the interface between the other procedural elements of the DSS – in the Control layer – and the user interface above. All function calls and data transfers are marshalled through this layer as a single point of contact. The remaining three packages transcend the Control/Model boundary, containing elements applicable to both domains. The Environment Package is broadly equivalent to the Problem Definition Module as outlined in the proposal. However, the key difference is that the concept of Alternative Solutions has been extracted and has been placed in the Strategy Package along with the Ranking concept, originally in the MCDA module. The third package, Performance, incorporates the Impact Assessment Module as well as the WaterMet^2 model as a self-contained entity.

Table 1: Mapping of DSS Constituents from Proposal to Design

Proposal Module	**Constituent**	**Design Packages**
Problem Definition Module	Alternative Solutions	Strategy Package
	System Load	Environment Package (Scenarios)
	Evaluation Criteria	Environment Package
Impact Assessment Module	Cost Models	Performance Package
	Metabolism-based	Performance Package / WaterMet^2 Package
	Decision Matrix	Performance Package
MCDA Module		Strategy Package

The original Module concept names are mapped to the Packages in the revised design, shown in Figure 2, as seen in Table 1, above.

Interface to Aware-P

At the time of writing there remains uncertainty as to the best mechanism for linking the WP54 Decision Support System with the Aware-P/Baseform visualization platform to be employed by WP53.

Where appropriate, the DSS framework will seek to reuse extant or forthcoming modules developed for the Baseform/Aware-P platform including those not directly related to user interface or visualization. As a result, the boundary between Aware-P and the other components in the DSS as seen in Figure 2 may be revised accordingly to accommodate such integration. For example, as previously mentioned, the DSS should attempt to use the data management provision of Aware-P where practicable. In addition, the presence of an existing Performance Indicator library on the Aware-P platform itself should be leveraged if possible. It seems likely that an additional level of indirection will be required to map, hopefully on a one-to-one basis, the indicators employed by WaterMet^2/DSS and those that are used by Aware-P for the purposes of reporting and visualization.As has been intimated, in order to safeguard the path to the future integration of the DSS with the platform components developed elsewhere, it is envisaged that an XML schema will be employed for the control and configuration of the DSS. This, coupled with a user-facing HTTP server will facilitate both online and offline (file-based) configuration and execution of the DSS. Incoming XML documents or requests are passed to an XML Parser which will permit the use of single messages which configure individual facets of the DSS Object– as might be produced by an end-user configuring the system through a GUI - as well as compound messages which can be used to configure and execute the DSS in a single step – as might be employed by a file-based mode of operation.

Data output from the DSS will conform to its own XML schema, applied by an XML Encoder, which, if necessary, can be modified by using an XSLT (eXtensible Stylesheet Language Transformation) to convert the output data into whatever format suits the Aware-P platform best. The information output from the DSS is envisaged to include, details of the Intervention Strategies, Rankings and the Performance Indicator Values (i.e. time series) for each of the Risk and Performance Indicators selected for analysis by the end user.

Software Components

This section describes each of the conceptual classes of the DSS in turn and should be read with reference to the individual package Conceptual Class Diagrams as well as the overview diagram which can be found in the pull-out in Appendix B.

Each package is introduced and the key relationships between the concepts in that package are explored. Where appropriate, the key public functions for each concept are described.

Further functional description can be seen in the individual package Conceptual Class Diagrams. Dashed lines on the Class Diagrams indicate relationships or concepts that are not fully understood at the time of writing and which will become clearer as the development of the WaterMet^2 metabolism model progresses.

DSS Engine Package

The DSS Engine represents the interface of the control layer of the DSS to that of the User Interface. It comprises a single instantiated class, DSS, which is responsible to responding to requests for data and control functions from the user interface and to dispatch and marshal information from the subsidiary packages. This package also includes a number of abstract class types which are common to the other implementation packages but which are not used directly.

DSS

This class is responsible for instantiating all of the subsidiary objects in the DSS as a whole and for marshalling and dispatching the messages for their interaction with the user interface. The key functions for this class are the same as the publically accessible functions for the "Manager" classes in the Environment and Strategy packages as well as the "Evaluator" class in the Performance package. It can be considered that the DSS object will effectively respond to HTTP requests to call these functions.

StreamableXML

The abstract StreamableXML class is the basis for many of the classes in the conceptual class hierarchy – being able to read and write itself to an XML stream. All derivatives of this class are expected to implement the key functions outlined below to allow XML documents to be generated and parsed for each descendant class. Each instance of a StreamableXML object is guaranteed to have a unique identifier.

Key functions

getXML	Abstract function to read a class from a provided XML document.
setXML	Abstract function to write a class to an XML document.

StreamableXMLContainer

StreamableXMLContainer is itself a derivative of the generic StreamableXML class, adding functions to allow it to store other derivatives of StreamableXML. This permits a nested hierarchy of objects to be stored, searched and read/written from/to XML documents. Containers may be used to store references to other objects or to act as their owners.

Key functions

add	Abstract function for creating a new StreamableXML object appropriate to the type of container.

remove	Removes an object from the container additionally deallocating any resources associated with it if this container is the object's owner.
clone	Creates a copy of a given object in the container.
copy	Creates a copy of a given object and stores it in a nominated container.
count	Returns the number of elements stored in the container.
find	Determines whether an object with the given ID is present in the container.
getXML	Returns an XML document containing all of the child elements by calling their individual getXML functions.
setXML	Abstract function to read child elements from an XML document by calling their individual setXML functions.

Environment Package

The "Environment" Package largely corresponds to the functionality envisaged in WP54.2, the "Problem Definition" module, with the exception of the Alternative Solutions concept. Its primary responsibility is the initial configuration of the system including the initial states for the WaterMet^{2}model, the definition of any variant scenarios that are to be considered as well as describing the time horizon over which the analysis is to be undertaken. With the exception of the Model Inputs, most of the data members in the Environment Package are dynamically configured by the user interface which is able to read/write XML data to configure the Environment according to the user's preferences.

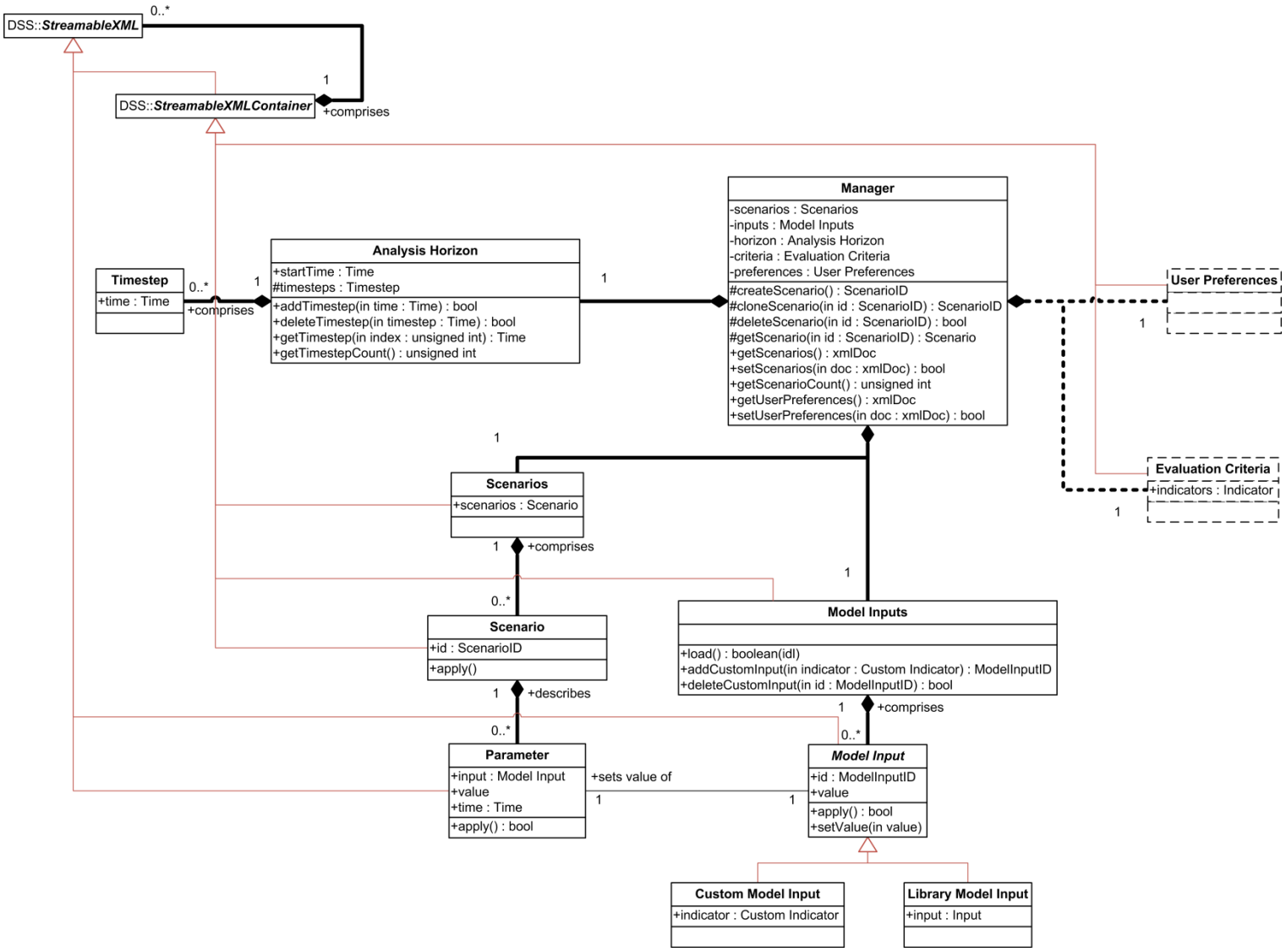


Figure 3: Conceptual class diagram for Environment Package

Environment::Manager

The manager class is the public face of the Environment package and all control of its subsidiary objects is marshalled through an instance of this class.

Key functions

getScenarios	Returns an XML document representing the current configuration of the Scenarios object and all the contained Scenarios by calling Scenarios::getXML().
setScenarios	Takes as a parameter an XML document that has been received from the user interface and which contains the required specification of the Scenarios object. This is achieved by calling the setXML() function of the Scenarios container.
getUserPreferences	Returns an XML document representing the current configuration of the User Preferences object by calling User Preferences::getXML(). The output of this function is intended to be used to populate the preferences section of the user interface.
setUserPreferences	Takes as a parameter an XML document that has been received from the user interface and which contains the required specification of the User Preferences. These are set by calling the UserPreferences::setXML function.

Environment::User Preferences

A single, publically accessible, User Preferences object is instantiated by the Manager object. This object is responsible for storing any user-defined preferences, particularly qualitative data that will be used to inform the procedure implemented by the Ranking class and its derivatives. At the time of writing, the details of the types of User Preferences that might be expressed are unknown; however, it is likely that this class will be derived from StreamableXMLContainer and act as a repository for a more specialized User Preference class which will be derived from as necessary to accommodate different types of preference.

Key functions

getXML	Returns the current configuration of the User Preferences as an XML document.
setXML	Receives an XML document and parses it, according to the appropriate schema, to configure the User Preferences object. When called, the existing contents of the User Preferences object are discarded.

Environment::Analysis Horizon

The Manager class maintains a single Analysis Horizon object which is used to describe the time frame over which the evaluation of the system is to be undertaken as well as

identifying intermediate timesteps at which interventions are permitted to take place. The timesteps are represented by a collection of Timestep objects.

Environment::Timestep

In conjunction with a Analysis Horizon object, timestep objects reference times within the analysis horizon at which the state of the network is evaluated. In addition, the Influences associated with Interventions are defined as taking place at these intermediate timesteps.

Environment::Scenarios

The Manager class instantiates a single container object of the Scenarios class to store instances of the Scenario object which represent varying initial, or other time-variant, conditions for the system to be considered during evaluation.

Key functions

apply	Applies a given Scenario by calling the Scenario's own apply() function.
getXML	Returns the description of all of the Scenarios by calling each Scenario's getXML function in turn. This function is ordinarily employed by the getScenarios() function of the Manager object.
setXML	Receives an XML document and parses it, constructing Scenarios as appropriate by creating new Scenario objects as applicable and then calling their setXML() functions. If a call to this function succeeds, the existing Scenarios contained herein are discarded and replaced with those defined in the XML document.

Environment::Scenario

A Scenario object represents the concept of a group of input Parameters which define initial conditions for the metabolism model or state changes that occur at fixed times during the analysis horizon.

Key functions

apply	Applies the Scenario to the inputs of the metabolism model by calling each contained Parameter's apply() method.
getXML	Returns the description of all of the contained Parameters by calling each Parameter's getXML function in turn.
setXML	Receives an XML document and parses it, constructing Parameters as appropriate by creating new Parameter objects as applicable and then calling their setXML() functions. If a call to this function succeeds, the existing Parameters described by the Scenario object are discarded.

Environment::Parameter

As they are currently defined, a Parameter object describes a variation in a Model Input that can take place at a given time within the Analysis Horizon. Each instance of a Parameter is linked to a single Model Input object.

Key functions

apply	Applies the Parameter to the associated input of the metabolism model by calling that input's own apply() function.
getXML	Returns an XML document containing the description of the time, change in value and Model Input associated with this parameter.
setXML	Populates the attributes of this parameter by parsing the passed XML document. Any Model Inputs passed must be extant within the Model Inputs collection of the Manager object.

Environment::Evaluation Criteria

A single, publically accessible, Evaluation Criteria object is instantiated by the Manager object. This object is responsible for identifying which of the Performance::Indicator objects in the DSS are to be used as comparators for the ranking procedure implemented by the Ranking class and its derivatives. Each Indicator object identified in the Evaluation Criteria must exist within the DSS and care must be taken by the user interface to validate the selected entries against the extant indicators.

Key functions

getXML	Returns the current configuration of the User Preferences as an XML document.
setXML	Receives an XML document and parses it, according to the appropriate schema, to configure the User Preferences object.

Environment::Model Inputs

This object is populated when it is instantiated by reading from a local database all of the model inputs that are exposed by the WaterMet^2 metabolism model (stored as Library Model Inputs) as long with any customized model inputs that have been created which are used to directly modify values of Indicators. This latter type is stored as Custom Model Inputs.

As with most of the container types employed by the DSS, Model Inputs is derived from the DSS::StreamableXMLContainer class which allows its contents to be easily rendered to XML for transfer to the user interface.

Key functions

getXML	Returns an XML document containing the specification of all of the contained Model Inputs.

Environment::Model Input

The abstract Model Input class has two derivations for accommodating predefined and custom model input types:

Key functions

apply	Applies the current value of the Model Input to the associated WaterMet^2 Input (for Library Model Input) or Custom Indicator (Custom Model Input).
getXML	Returns an XML document containing the specification of the Model Input and its associated WaterMet^2 Input or Custom Indicator.

Environment::Library Model Input

In addition to the attributes and methods inherited from the Model Input class, the Library Model Input introduces an association with a single Input exposed by the WaterMet^2 metabolism model. Thus changes made to this type of model input are directly reflected in the inputs of the metabolism model.

Environment::Custom Model input

A Custom Model Input class is provided to permit the modification of indicators that do not have direct analogues within the WaterMet^2 metabolism model. Each Custom Model Input is associated with a single Custom Indicator object whose value it modifies. This allows the DSS to be extended to include new indicators to be considered by the ranking methodology without necessitating an extension to the metabolism model itself.

Performance Package

This package corresponds to the Impact Assessment Module envisaged by the proposal and is responsible for the management of the system performance and risk indicators and for performing the model evaluation. Evaluation takes part in the WaterMet^2 Metabolism Model, developed as part of WP33, which is encapsulated as a sub-package.

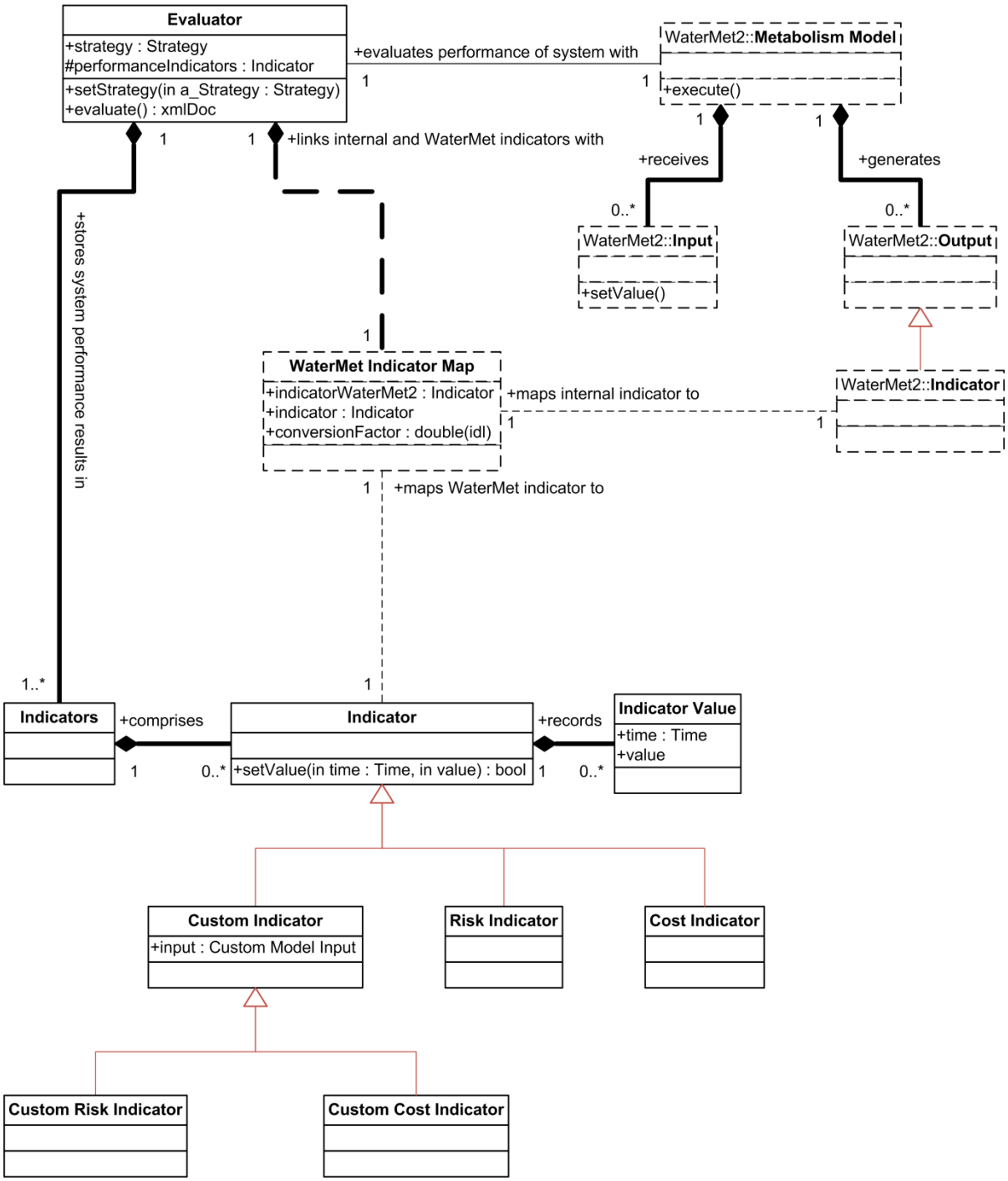


Figure 4: Conceptual Class Diagram for Performance Package

In addition to the indicators produced from the outputs of the WaterMet^2 model, the Performance package is also responsible for the derivation of additional cost and risk indicators. These indicators are accommodated through the use of "Custom Indicator" concepts which, further, may be mapped directly to model inputs.

Performance::Evaluator

The Evaluator class is the principal component of the Decision Support System and a single instance is instantiated by the DSS object when the system is initialised. This object's role is to evaluate the performance of the system for a given Problem Definition (i.e. Environment Package) and set of Interventions (i.e. Strategy Package).

Key functions

setStrategy	Selects the current strategy to be analysed. If non-null, calls the Strategy's apply() function to configure the inputs of the metabolism model to reflect the interventions undertaken in the strategy.
evaluate	Evaluates the performance of the system for the current strategy (if any) over the analysis horizon for each of the Scenarios defined in the global Environment::Scenarios object. If no Strategy has been selected for evaluation, the results of the evaluation are stored in the objects in the Performance::Indicators container – representing the baseline system state. Otherwise the results are stored in Performance::Indicator objects associated with the current strategy.

Performance::Indicators

The Evaluator object is responsible for instantiating a single Indicators container which is populated at system startup from a database table containing details of all of the Performance and Risk Indicators available to the Decision Support System. Following a successful run of the metabolism model, the Indicators object will contain Indicator objects derived from the WaterMet2 Output for each indicator over each timestep and for each scenario defined in the Environment Package.

The Indicators container derives from the StreamableXMLContainer class which will allow details of the available Indicator objects therein to be published to the user interface.

Key functions

getXML	Returns an XML document containing the specification of all of the contained Indicator objects by calling their own getXML() functions in turn.

Performance::Indicator

The basic Indicator class represents a single performance indicator for the model. Time series data are accommodated as a collection of Indicator Value objects.

Performance::Risk Indicator

As per the Indicator class. This represents a specialisation that differs from its ancestor in name only and acts as a placeholder for the implementation of a risk-specific class.

Performance::Cost Indicator

As per the Indicator class. This represents a specialisation that differs from its ancestor in name only and acts as a placeholder for the implementation of a cost-specific class.

Performance::Custom Indicator

Custom Indicators are those Indicators that are not directly computed by the WaterMet^2 metabolism model. Instead, the indicator values are supplied through an association with a Custom Model Input – which in turn derives its value from an Intervention's Influence.

Performance::Custom Risk Indicator

As per the Custom Indicator class. This represents a specialisation that differs from its ancestor in name only.

Performance::Custom Cost Indicator

As per the Custom Indicator class. This represents a specialisation that differs from its ancestor in name only and acts as a placeholder for the implementation of a cost-specific class with direct association with a Custom Model Input.

Performance::Indicator Value

The Indicator Value class associates a single value with a single timestamp and is used, in aggregate, to represent time series data for an individual Indicator.

Performance::WaterMet^2 Indicator Map

In order to maintain an abstracted interface between the DSS and the metabolism model during development, a mapping class is used to associate the indicators employed internally by the DSS (i.e. the Performance::Indicator objects) and those output by the metabolism model. Ultimately, this association should become redundant with the WaterMet^2 Indictors being a subset of those available to the DSS as a whole.

WaterMet^{2}Sub-Package

The WaterMet^2 metabolism model is developed as part of WP33. Given that this is on-going work the package is illustrated here in skeleton form only with the key concepts that will be necessary to ensure integration with the wider Decision Support System. As such the internal interface between WaterMet^2 and the DSS framework is undefined at the time of writing, given the different potential modes of linkage (remote, DLL, source code) – which ultimately depends upon how the WaterMet^2 model is ultimately deployed. However, future developments in this interface will not affect the external programming interface presented to the user interface of the DSS.

WaterMet^2::Metabolism Model

Key functions

execute	Runs the metabolism model for the currently specified set of inputs..

WaterMet^2::Input

The Input class represents all of the parameters that can be set within the WaterMet^2 model. This class will likely become specialised as the input requirements of the model become clearer.

Key functions

setValue	Sets the parameter to the given value for a given time.

WaterMet^2::Output

Output from the metabolism model is represented by the abstract Output class which is expected to encapsulate all output forms from the model, including time-series data. Accordingly, a number of specialisations will be required as the implementation of the metabolism model evolves.

WaterMet^2::Indicator

A specialised Indicator derivation of the abstract Output class is proposed to accommodate single value and time-series indicator value.

Strategy Package

This package encapsulates the concepts required to develop, modify and rank alternative solutions (Strategies) for the system. A single Manager object is instantiated by the global DSS object to control the generation and ranking of strategies in response to requests from the User Interface.

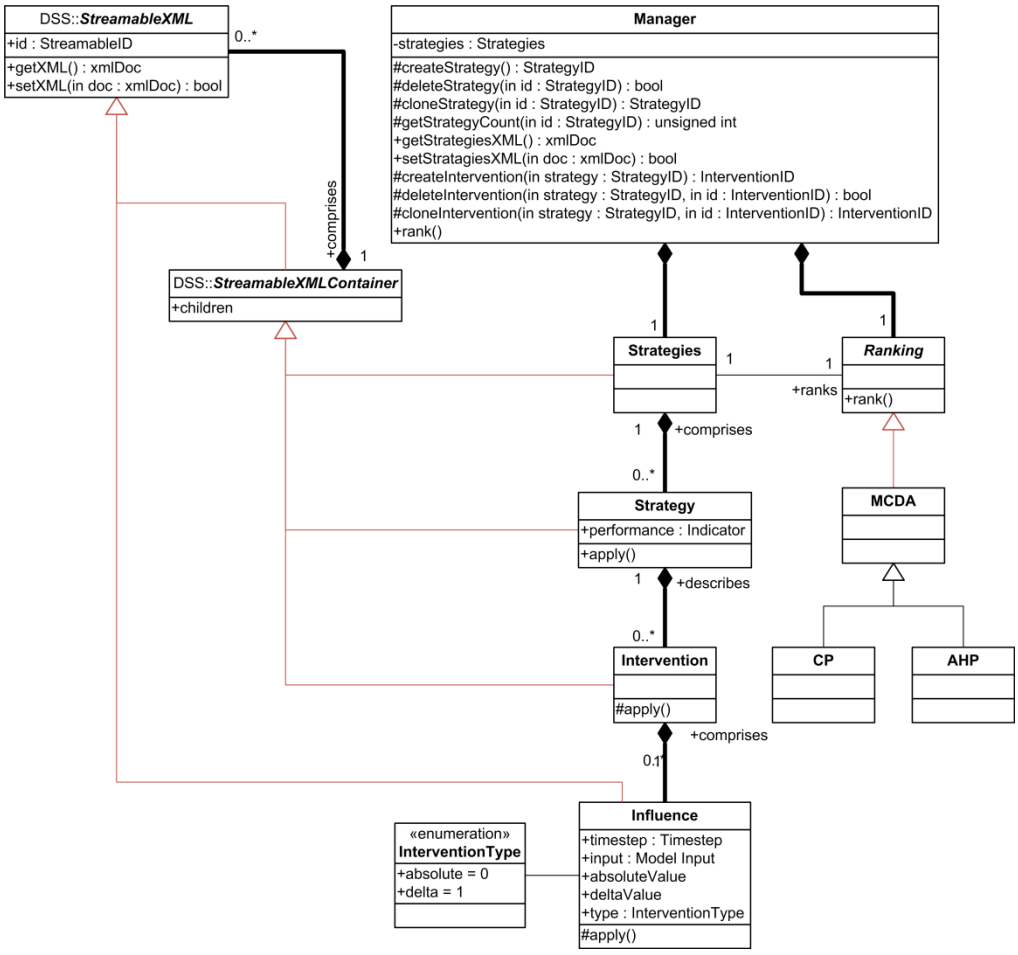


Figure 5: Conceptual class diagram for Strategy Package

Strategy::Manager

The Strategy Manager implements the public interface to the Strategy Package.

Key functions

getStrategiesXML	Returns an XML document containing a description of all of the defined Strategies, Interventions and Influences – along with the results generated by evaluation.
setStrategiesXML	Parses a supplied XML document to construct
rank	Calls the rank() function of a Ranking object to rank all of the defined Strategies.

Strategy::Strategies

A container class derived from DSS::StreamableXMLContainer for storing individual Strategy objects. A single Strategies object is instantiated by the global Stategy::Manager object.

Key functions

getXML	Returns the description of all of the Strategies by calling each Strategy's getXML function in turn. This function is ordinarily employed by the getStrategiesXML() function of the Manager object.
setXML	Receives an XML document and parses it, constructing Strategies as appropriate by creating new Strategy objects as applicable and then calling their setXML() functions. If a call to this function succeeds, the existing Strategies contained herein are discarded and replaced with those defined in the XML document.

Strategy::Strategy

Defines, as a set of Interventions, an option for the modification of the behaviour of the system.

Key functions

apply	Applies each of the contained Intervention objects to the metabolism model by calling the apply() function of each Intervention..
getXML	Returns an XML document containing the description of the Strategy and all of its constituent Intervention objects.
setXML	Populates the attributes of this Strategy and Intervention Objects therein by parsing the passed XML document. Any existing contents of the Strategy are discarded following the successful conclusion of this function call.

Strategy::Intervention

An Intervention describes an individual operation undertaken to modify the behaviour of the system. Since a single Intervention may affect several different system parameters, each Intervention comprises a number of Influence objects which may be used to describe individual effects on the Model Inputs.

Key functions

apply	Applies the Intervention to the metabolism model by calling the apply() function of each of the Influence objects associated with the Intervention.
getXML	Returns an XML document containing the description of the Intervention along with all of the subsidiary Influence objects.
setXML	Populates the attributes of this parameter by parsing the XML document parameter. Any Model Inputs passed must be extant within the Model Inputs collection of the Manager object.

Strategy::Influence

An Influence object is used to associate the effect of an Intervention with an individual Model Input. Thus each Intervention comprises one or more Influences on the Metabolism Model directly, through association with Library Model Inputs, or indirectly via Custom Model Inputs. As well as describing the change in a given Model Input, each Influence is associated with a Timestep at which it is deemed to take effect.

Key functions

apply	Applies the Influence to the associated Model Input by calling that Input's own apply() function.
getXML	Returns an XML document containing the description of the time, change in value and Model Input associated with this Influence.
setXML	Populates the attributes of this Influence by parsing the passed XML document. Any associated Model Inputs passed must be extant within the Model Inputs collection of the Environment::Manager object.

Strategy::Ranking

An abstract class responsible for ranking all of the Strategy objects contained in the Strategies collection, utilizing the preferences stated in the Environment::User Preferences object and according to the criteria described in the Environment::Evaluation Criteria object.

Key functions

rank	Ranks all of the strategies contained within the Strategies object,

Strategy::MCDA

An abstract class as a placeholder for a Multicriteria Decision Analysis algorithm (Köksalan *et al.*, 2011) as a descendant of the Ranking class. Two descendent classes are provided, implementing Compromise Programming (CP) and the Analytic Hierarchy Process (AHP), respectively.

Modes of Operation

This section describes in detail the process and data flow required to facilitate each of the use-cases identified in the Required DSS Functionality section above. These descriptions should be viewed in conjunction with the Sequence Diagram in Appendix A which provides an overview of the operation of the key evaluation process in the DSS.

Define Environmental Configuration

This use case covers several modifications that may be made to the objects in the Environment Package, each of which is superficially similar. These are:

- Definition of a Analysis Horizon and intermediate Timesteps
- Definition of Scenarios
- Definition of User Preferences
- Definition of Evaluation Criteria
- Enumeration of Model Inputs
- Definition of Custom Model Inputs

Data flow

For the definition of Scenarios, User Preferences, Evaluation Criteria and Custom Model Inputs it is expected that the User Interface will call the appropriate getXML() function to retrieve an XML document containing the current state of those items and to populate the User Interface display elements accordingly. If the user undertakes any modifications then a revised XML document is sent from the User Interface to the appropriate setXML() function.

Notes

Definition of the any of the Environmental parameters may take place synchronously at any time – as long as the model is not being evaluated.

At present each of the major concepts in the package has its own function to retrieve and set its contents via an XML document. If appropriate, this could be supplemented by a function in the Environment::Manager which would permit all of the configuration elements to be retrieved or set at once.

Evaluate Baseline Urban Water Model

In order to evaluate the effect of strategies on the system, it is necessary to run the unmodified model to establish a baseline.

Data flow

The User Interface should call the applyStrategy() function with a null parameter to ensure that the model has no interventions applied to it. This should be followed by a call to the evaluate() function. This function returns an XML document containing the contents of the Indicators object belonging to the Evaluator. In this document all of the Indicator result values for the baseline scenario can be found.

Notes

This operation can only be undertaken if the model is not already being evaluated. Calling evaluate() blocks all other DSS operations until the resulting XML document is returned.

Generate Intervention Strategies

The end user will need to specify intervention strategies to determine their effect on the performance of the system. A function is provided to clone an existing strategy to act as a starting point for a new strategy.

Data flow

The bulk of this functionality is devolved to the User Interface layer which is responsible for generating an XML document that contains the required specification of all of the Strategies. It is envisaged that the UI will call the getStrategiesXML() function to populate the user interface elements, allow the user to interactively define the strategies and then to call the setStrategiesXML() function to configure the Strategies object accordingly. In order to maintain referential integrity, the UI will also need to retrieve the list of Model Inputs that can be associated with a given Influence as well as a list of the Timesteps at which Interventions are permitted to take place.

Notes

This operation can only be undertaken if the model is not already being evaluated.

Evaluate Impact of Intervention Strategy

The evaluation of the effect of an individual strategy on the system is similar to that of evaluating the baseline system – albeit with the application of the strategy's interventions beforehand.

Data flow

Initially, the User interface will need to retrieve a list of the available Strategies by using the getStrategiesXML() function. It will then prompt the user to select which of the strategies is to be evaluated.

The User Interface should then call the applyStrategy() function with the selected Strategy as a parameter to ensure that the Interventions of the given Strategy are applied to the model. This should be followed by a call to the evaluate() function. This function returns an XML document containing the contents of the Indicators object belonging to the selected Strategy.

Rank Intervention Strategies

This operation evaluates in turn each of the defined Strategies and applies a ranking methodology to order the Strategies in order of the preferences expressed in the Environment Package.

Data flow

The User Interface need only call the rank() function to begin the ranking process. This function returns an XML document describing all of the Strategies, their ranking order and scores.

Notes

By using the evaluate() function repeatedly, the rank() function blocks the DSS – no other DSS operations can be undertaken until the result has been returned.

Interactively Modify and Evaluate Intervention Strategy

The end user may elect to modify an existing strategy and to force its revaluation as well as, if desired, submitting the modified Strategy to be included in a re-ranking of all of the Strategies.

In order to preserve an existing strategy, the user may also opt to copy a strategy before modifying it. A function is provided within the DSS to clone an existing Strategy; however this may be more expediently achieved in the User Interface itself.

Data flow

As with the creation of the Strategies in the first instance, the interactive functionality is devolved to the User Interface layer which is responsible for generating an XML document that contains the required specification of all of the Strategies. It is envisaged that the UI will call the getStrategiesXML() function to populate the user interface elements, allowing the user to interactively select a particular Strategy and to modify its interventions and then to call the setStrategiesXML() function to reconfigure the Strategies object accordingly. The setStrategy() function will then need to be called for the selected Strategy and then the evaluate() function to update the results stored in its collection of Performance Indicators. The full scenario results are returned as an XML document in the call to evaluate().

Notes

This operation can only be undertaken if the model is not already being evaluated.

Future Considerations

Interface to WaterMet^2

Although the use of the "Flexible Interface" underpinned by XML removes the need to explicitly define the entirety of the DSS API at the design stage, it does little to assist the developers of the User Interface as to understanding what the GUI requirements of the DSS may be. To that end, it is anticipated that the first task undertaken as part of D54.2 will be the outline specification of the XML schemas that will be used to describe the data transferred to and from the User Interface. This will require close collaboration with the developers of the WaterMet^2 model as, ultimately, the specification of the data inputs and outputs need to conform to the requirements of that model. This, however, should not be a problem as most of the WaterMet^2 model development will take place at Exeter.

Direct-coupled interface

As alluded to in the proposal, the DSS will also be supplied with a basic user interface to allow it to be run in a standalone fashion as well as for the purposes of local testing.

It is proposed that one of the early activities undertaken in the implementation of the DSS is a mock-up of UI elements for each of the use cases which should also seek to inform the development of the Aware-P-based user interface that will be employed ultimately.

Summary

This document describes the design of an Integrated Framework for the development of a Decision Support System to facilitate decision-making for the long-term city metabolism planning problem. Detailed functionality and related input and output data flows for each of the DSS modules to be developed is presented and the software architecture is defined in terms of a number of packages which subsume the originally envisaged modules:

- The Problem Definition module (Task 54.2) which forms the bulk of the Environment package in the design.
- The Impact Assessment module (Task 54.3) – analogous to the Performance package in the framework.
- The MCDA Ranking module (Task 54.4) which is now represented in the Strategy package.

These modules will ultimately be integrated together into the DSS software (Task 54.5) by developing the Graphical User Interface (GUI) and the relevant data management structures and related services – leveraging the cross-platform strengths of XML. All this, in turn, will enable the solution of the long-term city metabolism planning problem by means of problem structuring/definition, analysis and resolution. The DSS framework and the software will be based on the AWARE-P software platform (www.aware-p.org) as well as being provided with a standalone interface.

Glossary

Analysis Horizon	The period of time over which the performance of the Urban Water System is to be evaluated. Interventions are considered to be undertaken at discrete, pre-defined timesteps within the analysis horizon whilst values for the UWS performance indicators are, ordinarily, returned for the entire horizon.
Indicator	A measure of some aspect of the performance, cost or risk of an Urban Water System. The indicator may evaluate the system or a part of it, or an element of the system. In the context of this document, indicators are generated either by the WaterMet^2 model, the DSS itself or by associated custom inputs.
Input	An input into the WaterMet^2 model or a Custom Indicator calculated out with the model.
Intervention Strategy	A user defined set of *Interventions* over some analysis horizon.
Intervention	A measure undertaken to improve the performance of the network, defined in terms of type of intervention, location or asset affected, quantity or magnitude, timing and an associated cost model.
Multi-Criteria Decision Analysis	A family of analytical techniques for evaluating multiple criteria in decision making. Criteria may frequently be conflicting and result in a "trade-off" with themselves and other criteria or may be qualitative in nature and require the preference of a decision maker.
Ranking	A mechanism for comparing Intervention Strategies according to user-provided preferences and evaluated according to a subset of the available Indicators.
Scenario	A set of Inputs that represent one, potential, initial condition of the UWS– nominally, those conditions that are outside the control of the decision maker through applying interventions - or changes to its state over the analysis horizon. Scenarios can be used to model, for example, varying loading UWS conditions (demand and/or rainfall) over the analysis horizon due to climate change and/or urbanisation.

References

COELHO, S. T. &VITORINO, D. 2011.*AWARE-P: a collaborative, system-based IAM planning software*. IWA 4th LESAM, 27-30 Sep, MülheimAn Der Ruhr, Germany.

KÖKSALAN, M., WALLENIUS, J., &ZIONTS, S. 2011. *Multiple Criteria Decision Making: From Early History to the 21st Century*. Singapore: World Scientific.

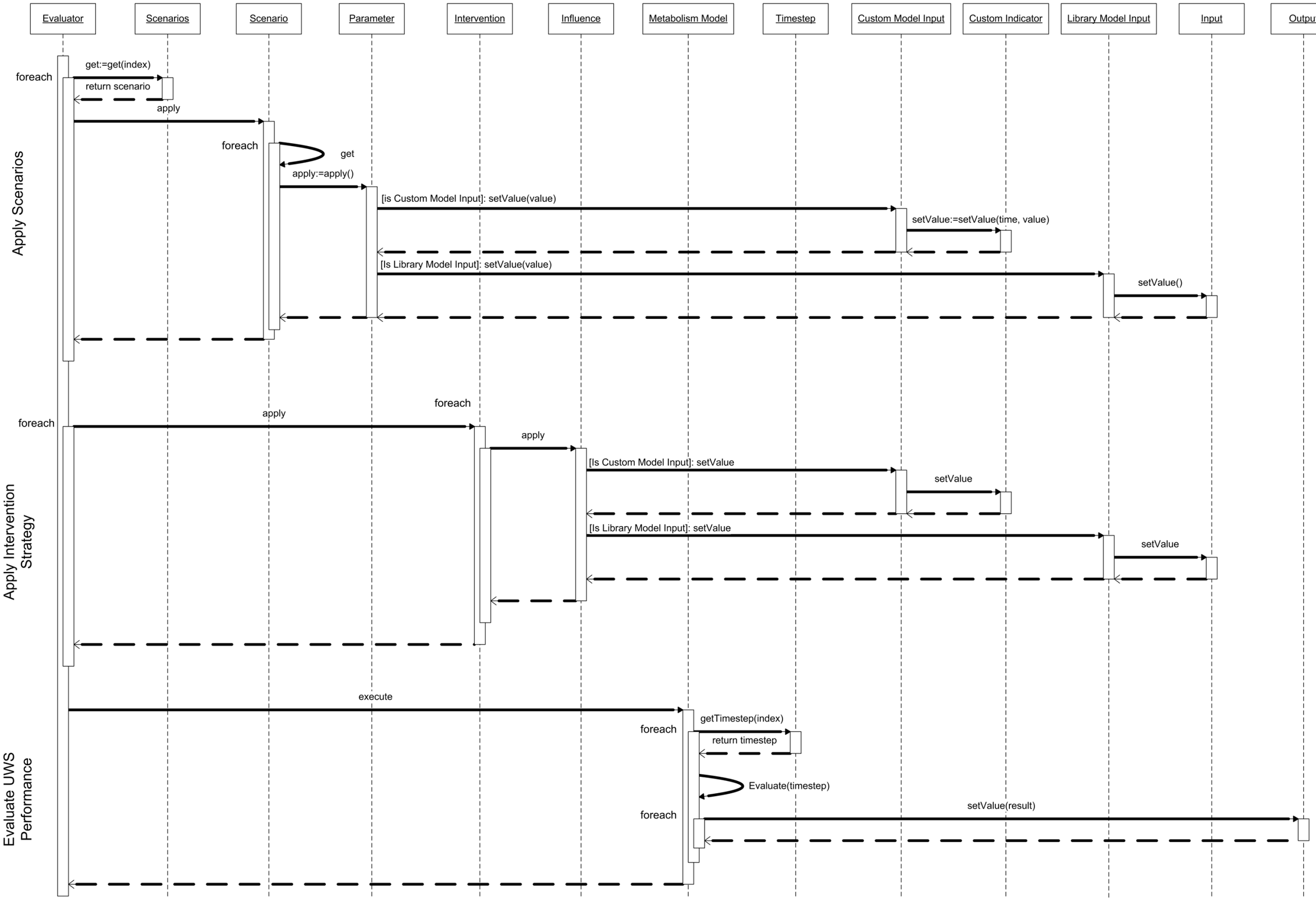
MORLEY, M.S., BICIK, J., VAMVAKERIDOU-LYROUDIA, L.S., KAPELAN, Z. &SAVIĆ, D.A. 2009. Neptune DSS: A Decision Support System for Near-Real Time Operations Management of Water Distribution Systems. In: BOXALL, J.B. &MAKSIMOVIĆ, C. (eds.),*Integrating Water Systems*. Proceedings of Computing and Control in the Water Industry (CCWI) 2009, Sheffield, UK. *pp249-255.*

OMG (OBJECT MANAGEMENT GROUP). 2011. *The Official Common Object Request Broker Architecture (CORBA) Standard.* http://www.omg.org/spec/CORBA/ (accessed 10 March 2012).

Rogerson, D.E. 1997. *Inside COM: Microsoft's Component Object Model.*Microsoft Press, Redmond, U.S.A.*376pp.*

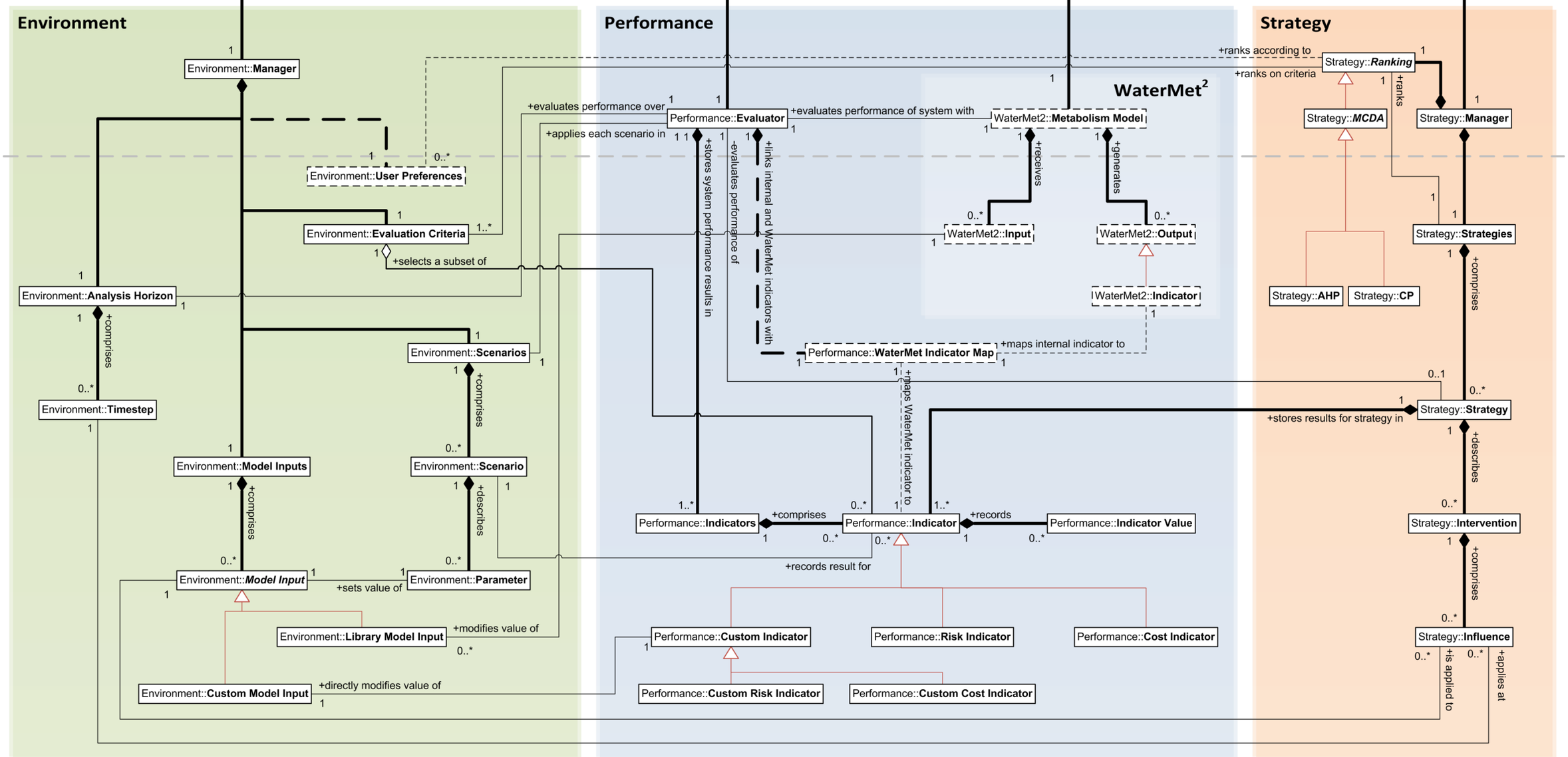
SAATY, T.L. 1980. *The Analytic Hierarchy Process: Planning, Priority Setting, Resource Allocation*, ISBN 0-07-054371-2, McGraw-Hill, New York, U.S.A. *287pp.*

ZELENY, M. 1973. Compromise Programming. In:ZELENY, M. & COCHRANE, J.L. (eds.), *Multiple Criteria Decision Making.*University of South Carolina Press, Columbia, U.S.A.*pp373-391.*


Evaluator
Scenarios
Scenario
Parameter
Intervention
Influence
Metabolism Model
Timestep
Custom Model Input
Custom Indicator
Library Model Input
Input
Output
Apply Scenarios
foreach
get:=get(index)
return scenario
apply
foreach
get
apply:=apply()
[is Custom Model Input]: setValue(value)
setValue:=setValue(time, value)
[Is Library Model Input]: setValue(value)
setValue()
Apply Intervention Strategy
foreach
apply
foreach
apply
[Is Custom Model Input]: setValue
setValue
[Is Library Model Input]: setValue
setValue
Evaluate UWS Performance
execute
foreach
getTimestep(index)
return timestep
Evaluate(timestep)
foreach
setValue(result)

DSS UI

DSS Engine



Integrated Decision Support Framework
D 54.1

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 265122.

SEVENTH FRAMEWORK PROGRAMME