A pattern-oriented approach to development of a real-time storm sewer simulation system with an SWMM model

Shiu-Shin Lin, Ying-Po Liao, Shang-Hsien Hsieh, Jan-Tai Kuo and Yen-Chang Chen

ABSTRACT

This paper documents a real-time storm sewer simulation system (RTS4) in conjunction with a Storm Water Management Model (SWMM) based on a pattern-oriented approach. The RTS4 is initiated by system analysis ascertaining functional requirements, which is sequentially followed by conceptual model, pattern languages, concrete pattern-based design, implementations and applications. The proposed conceptual model helps sketch out a core software skeleton in relation to prior system requirement analysis. Of the proposed pattern languages, each can be regarded as a 'building block' on which the concrete pattern-based design is built. Finally, the RTS4 is implemented by following the proposed pattern-oriented design. The applicability of RTS4 is demonstrated with respect to storm sewer simulation and real-time operations. The results of the simulation show that the proposed pattern-oriented approach offers a promising basis for software system developments such as RTS4.

Key words | pattern language, pattern-oriented approach, real-time operation, storm sewer systems, storm water management model (SWMM)

Shiu-Shin Lin (corresponding author)
Department of Civil Engineering,
Chung-Yuan Christian University,
Chungli 32023,
Taiwan
E-mail: *linxx@cycu.edu.tw*

Ying-Po Liao
Shang-Hsien Hsieh
Jan-Tai Kuo
Department of Civil Engineering,
National Taiwan University,
Taipei,
Taiwan

Yen-Chang Chen
Department of Civil Engineering,
National Taipei University of Technology,
Taipei,
Taiwan

INTRODUCTION

In order to reduce storm damage in urban areas it is necessary to have physical information, e.g. timely observation data or simulated prediction water stages, that will guide disaster prevention. In Taipei, for example, real-time understanding of predicted migration of sewer flow helps in disaster prevention. A Real-Time Storm Sewer Simulation System (RTS4) gives a physical-based solution, capable of predicting the migration of sewer flows during storm events. RTS4 is able to simulate storm sewer hydrodynamics with hydraulic facilities, thereby providing important data such as profiles of water stage and discharge for each junction and conduit. Determining whether or not junction surcharges occur through simulation is very useful to understand where the higher risk regions are located.

doi: 10.2166/hydro.2010.021

Figure 1 illustrates the possible complexities and challenges involved in developing an RTS4. By solving full de Saint-Venant equations (Chaudhry 1993), the extended transport block of Storm Water Management Model (SWMM-EXTRAN, Huber & Dickinson 1988), a well-developed and widely used sewer model (Park & Johnson 1998; Zaghloul 1998; Tsihrintzis *et al.* 2007), is adopted here as the hydrodynamic numerical kernel in an RTS4. It is responsible for delivering a real-time physical-based solution of sewer flows. However, application of a RTS4 to actual events demands more functions than hydrodynamic modelling alone. Useful supplements would include real-time control, an interactive Graphical User Interface (GUI), Geographic Information System (GIS), visualizations for

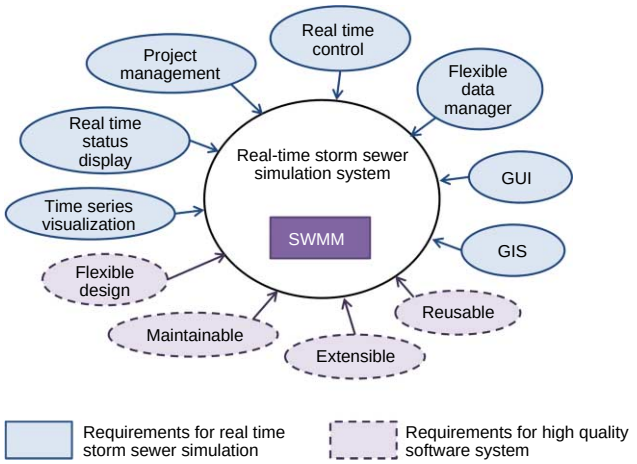

Figure 1 | Complexities and challenges for the development of real time storm sewer simulation system.

real-time status display, flexible input and output, project management and so on. In addition, requirements for high-quality software system developments, i.e. flexible design, maintainable code, extensible functionalities and reusable units and objects also provide complexities and challenges. Accordingly, these systems are generally versatile and delicate. Implementing the interdependent working mechanisms among participant units creates considerable complexities (Lin *et al.* 2006).

To meet those requirements for practical problems, a hard-coding approach to accommodate complicated inter-dependent working mechanisms is the obvious solution, but worthless. Our goal is to find a general solution by applying modern methodologies of software development for the development of a versatile real-time storm sewer simulation system based on the well-known hydrological numerical kernel, SWMM-EXTRAN.

For the purposes of this research, we propose a pattern-oriented approach to develop an RTS4. The objective here is not only to outline the languages and patterns that could develop a real-time storm sewer simulation system in a general way, but also to demonstrate feasibility by applying this procedure successfully to the production of an object-oriented system, the RTS4. Creating a well-designed software system then, as addressed in this paper, is very challenging. Object-oriented (OO) analysis and design continues to attract critical interest (Larman 2004). This can be attributed to the fact that its design facilitates reusable, flexible OO software systems that can be consistently maintained. Indeed, 'design patterns' (Gamma *et al.* 1995; Fowler 1996) are one of the most important methodologies, as evidenced by their influence on software design across different domains (Rising 1999; Massingill *et al.* 2000; Zhao *et al.* 2008). The working assumption behind the concept of 'pattern' is that 'problem-solution' pairs can be repeatedly adapted in order to quickly solve many routine tasks that are essentially similar (Alexander *et al.* 1977; Alexander 1979).

The procedure of the proposed pattern-oriented approach in this paper is illustrated in Figure 2. In the following section, based on OO analysis, we first analyze requirements, draw the system boundaries and sketch a constructive conceptual model. According to system boundaries given by OO analysis, a language pattern of an RTS4 is proposed during the design phase.

There are many themes worth exploring within the design phase. Consideration when developing a software system should ideally be given to extendable and reusable design issues at an abstract (rather than a domain or language-specific) level. System patterns become relevant in this context (Buschmann *et al.* 1996) but, in the interests of elucidating a more concrete definition for software patterns, it becomes necessary to highlight the forces (Coplien 2004). In this way, an individual pattern targets one problem to balance the distribution of forces it contains. The situation can become far more complex, however, not least where the collocating of many patterns in manifold relationships occurs in software design. In the bridging of one pattern to another, known as a 'pattern sequence', developers arrange complicated forces in a sequential way where highly interwoven forces can be decomposed into a lower level, thus rendering them more manageable for the purposes of analysis. Likewise, much higher forces relevant to the full scope of the target software system can be decomposed into smaller pieces and redistributed into an intricate pattern sequence.

Based on a top-down approach, the level of design details is refined in accordance with the iterative process to find the forces that produce stress in different levels (from the software level to the concrete design pattern level), thereby resolving our proposed pattern languages by balancing the forces for each level. As a result, pattern

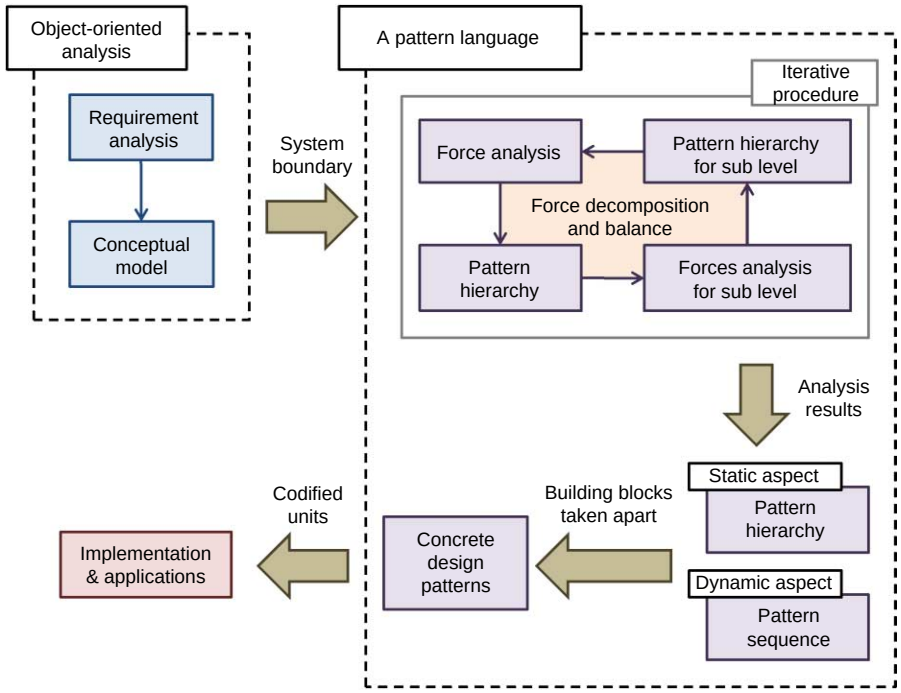

Figure 2 | A pattern-oriented approach to software design and development.

languages comprise two fundamental units. Pattern sequences can be regarded as one of them, which give the dynamic aspect of a pattern language; on the other hand, ramified pattern hierarchies can be regarded as another fundamental unit of pattern languages, which give the static aspect. In contrast to a focus on only one target design problem, pattern languages consequently aim at holistic support for one specific domain problem by merging different patterns into a concrete design to fulfil requirements and to complete conceptual models (Buschmann *et al.* 2007).

Both pattern sequences and pattern hierarchies are gathered by many concrete design patterns. Taken apart from them, eventually those concrete design patterns can be regarded as building blocks. Each concrete design pattern generally has its corresponding implementation in terms of a designate programming language, which implies that they are to be codified units. We propose a C++-based means of implementing RTS4 to demonstrate the feasibility of our proposed pattern-oriented designs, inclusive of details about program implementation issues such as development environments and tools. To demonstrate the applicability of RTS4, we introduce and discuss various applications before drawing conclusions.

A REAL-TIME STORM SEWER SIMULATION SYSTEM (RTS4)

For the first step, a system analysis must analyze the requirements in the specific domain problem and ascertain the common needs for real-time storm sewer simulations. These requirements and system descriptions are outlined and organized into a conceptual model to help define system boundaries.

Description of RTS4

RTS4 is a software system developed to demonstrate an implementation version based on various designs, patterns and pattern languages. Its main function is to perform a real-time storm sewer simulation, including additional operations for urban drainage systems. The necessary requirements can be described as follows.

1. *Real-time simulation*: The most important requirement is to trigger a hydrodynamic model that can simulate sewer flows under different conditions in real time (Lin *et al.* 2004, 2005, 2006). Input data, including possible real-time control conditions, can be factored

into this model so that solutions for the corresponding sewer flows can be immediately calculated.

2. *Real-time monitoring:* Real-time monitoring and display of the states for hydrodynamic modelling is very important when ensuring that engineers are able to conduct a timely check of system conditions i.e. details of junction surcharging, water stages or discharges for each junction and conduit.
3. *Real-time operation:* Real-time operation simulations for urban drainage system are an inevitable development in that engineers may want to know how sewer flow conditions correspond not only to real-time rainfalls and boundary conditions, but also to additional real-time operational states (Lin *et al.* 2005). In particular, for large cities such as Taipei, operations at pump stations and gate stations are critical when it comes to reducing the risk of inundation in storm events. Considering a 'what-if' analysis for a real storm event, engineers need to determine how well the sewer flows respond to their testing of operational variables (such as pumping rates or on/off states for pumps) in a timely manner that enables them to counteract future damage.
4. *Flexible data I/O management:* From these prior functionalities involving intrinsic simulation computing, the importance of flexible management for data input/output for models and working projects arises such as readily rebuilding or modifying SWMM-EXTRAN input data.
5. *Effective GUI:* Covering all of the requirements described above, an effective GUI is required for (1) real-time simulation state display; (2) user-friendly building and updating of input data; (3) showing output final results; (4) project management and corresponding simulation parameter setup; (5) interactively manipulating hydrological facilities to perform real-time operations; and (6) viewing sewer system layouts in an interactive GIS environment and dynamic visualization of information.

The conceptual model

Regarding the above-described requirements, a couple of constructive concepts are then incorporated into the software system. In order to sketch the core software skeleton, the conceptual model (also referred to as the 'domain model' in object-oriented analysis) is gradually given more concrete form through the inclusion of assorted conceptual objects and their interactions for this domain-specific problem (Fowler 1996). Regarded as one of the architectural patterns (Fowler 2003), a domain model provides a reference point for presenting the complex rules and logics for a specific domain problem. In other words, each object in a domain model acts on behalf of an individual concept, not necessarily equal to a real class in system implementations despite class diagrams using UML notations (Object Management Group 2005).

The five described requirements (excluding the last since 'effective GUI' cannot be issued individually) can be conceptualized as individual objects. The conceptual model presented in a more concrete form with interactions is depicted in Figure 3. In relation to Requirements 1 to 4, the proposed conceptual model is comprised of five major individual objects: a Hydrodynamic Model, Visualizer, Main Controller and an Input File Auxiliary, while User features as the main actor.

The User initially interacts with the Main Controller containing the GUI interfaces that perform all the functions. The Main Controller not only mediates to all other parts, but is also accountable to Requirement 3 which contains the control panels and related forms needed to perform the real-time operations for sewer systems. As for sewer flow conditions, the Hydrodynamic Model is related to Requirement 1. In this instance, real-time modelling is completed via internal numerical iterations, triggered by the Main Controller who also passes compulsory real-time rainfall data and boundary data and/or real-time sewer operation variables. A Main Controller is associated with one Hydrodynamic Model, and possibly more than one Visualizer. These might include: (1) a couple of charts for displaying real-time simulated data for selected junctions and (2) a GIS to display the layout of sewer systems. Finally, (although this is usually done first), the Input File Auxiliary is used to fulfil part of Requirements 4 and 5. This helps manage SWMM-EXTRAN input data, which can finally be exported to the Main Controller. To fulfil the rest of Requirement 4, the Main Controller flexibly manages working projects and input data entities by collaborating

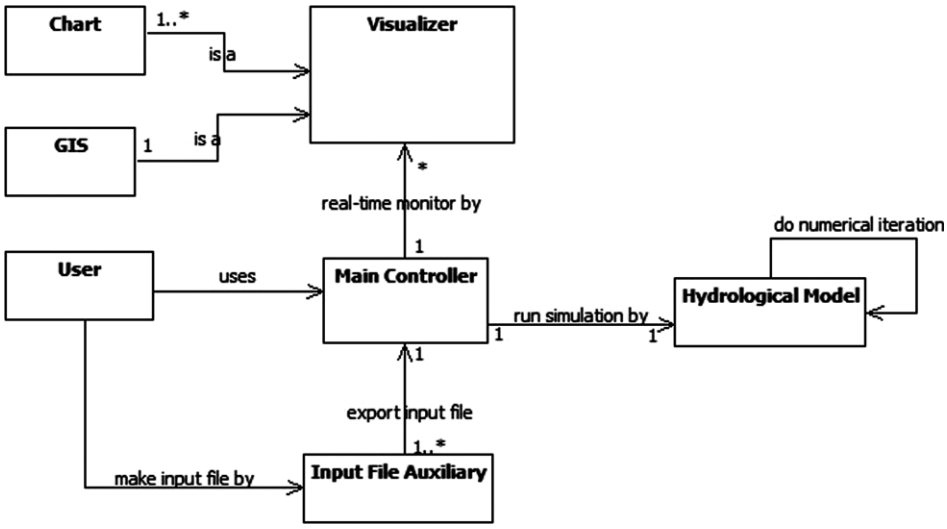

Figure 3 | Conceptual model for RTS4.

with the Input File Auxiliary. This is also because of the sharing of data entries among other parts, e.g. input data feeding for the Hydrodynamic Model.

THE PATTERN-ORIENTED APPROACH

The pattern-oriented approach is an iterative process to refine the level of detail for the designs. By using a top-down method, designs can be gradually narrowed down and eventually detailed as codified units. As noted in the introduction, the regulation of forces is a primary concern for patterns and pattern languages. To resolve the designs from a bird's-eye view to a worm's-eye view, an exertion of force is the most important activity driving progression through the pattern hierarchy. Global forces resulting from the union of component forces in each lower level are first found and decomposed into sublevels. To balance these, subsequent patterns are applied. Given the split by higher level forces, the sub-problem areas arise due to new contexts emerging from pattern application, and these require subsequent balancing. The iterative process can eventually ramify the pattern in which the hierarchy is applied, from top to bottom level. The basic elements of this ramification are referred to as 'patterns' which function as 'building blocks' that can be directly codified.

Forces and pattern languages

To blueprint the proposed pattern languages, it is convenient to draw up a two-stage presentation by first introducing a static view and then following with a dynamic view. In the first stage, the static view shows how the ramifications of pattern-applying hierarchies evolves by force decompositions, and how the forthcoming sub-contexts of each level form the topology of those patterns used.

Domain Model is the entry point. While regarded as a conceptual model in the analysis phase, it also remains a form of architectural pattern. The global, top-level force for real-time storm sewer simulation system development demands a proper solution by which to model very intricate concept rules. Domain Model is a good solution to wrap individual independent logic units and to enable the easy visualization of inter-dependent relationships, as well as providing subsequent designs with appropriate guidance. Two sub-forces arise from the global force. One is a flexible framework for application implementations, permitting the loose coupling of user interfaces and logical operations in addition to ensuring that any changes resulting from user interfaces do not disturb the logical layer. The other is a highly coherent design for logic rules for specific domain problems such as real-time storm sewer simulation.

As Figure 4 shows, two branches are accordingly expanded from Domain Model. Figure 4 applying for user interface separations is Model-View-Controller pattern, and for internal logic units partitioned from the whole problem space, a Domain Object pattern. A Model-View-Controller pattern provides the applications with the skeleton for a robust framework with the potential to decouple user interfaces (Viewer), logical rules and domain-specific contents (Model) and for dispatching functionalities and data flows to the centralized controller. For this application, those functionalities related to domain-specific problems are partitioned from the Model-View-Controller and completed by Domain Object. There are several consequent forces inside the two branches. The two sub-hierarchical pattern systems evolved are shown in Figures 3 and 4.

Figure 5 pertains to the Domain Object pattern, wherein more details are generated by two further sub-forces. Regarding logic contents for the real-time simulation problem, sub-forces are created in terms of two styles: (1) dynamic procedure dominant and (2) static state dominant.

Considering the former (such as a simulation actor), a design that provides domain-specific operations at the abstract level to encapsulate the details of implementation needed to facilitate polymorphism is required. An Explicit Interface pattern is the best candidate for carrying out domain-specific operations. It aims to provide a set of abstract public functions with a view to completing dynamic operations (not static data or states).

In contrast, for static state dominant content such as simulated data, the most urgent force is concerned with whether individual static states can be identified quickly. Here the Value Object pattern can issue the efficiency

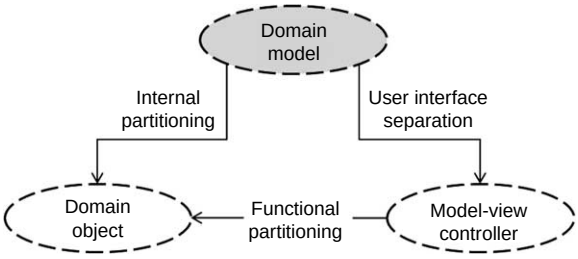

Figure 4 | Pattern topology—domain model.

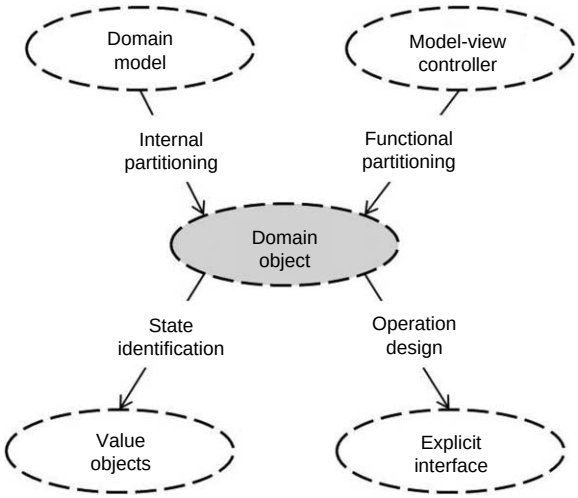

Figure 5 | Pattern topology—domain object.

performance. Each Value Object type consists of only 'values', with a specific type to identify the current static state, in which no extra operations create overheads. Simply speaking, Value Object is similar to using a naked array. Value Object types not only offer very good performance but also a more meaningful type of information to foster clearer identifications of state.

As shown in Figure 6, in contrast to domain logic contents the other branch pertinent to Model-View-Controller aims at application. Derived from the Domain Model, our design of the overall architecture is focused upon the Model-View-Controller pattern, intended to decouple user interface designs and domain logic contents. By extension, four decomposed forces are invoked by these three sub-parts (Model, View and Controller) in addition to data handling and notification of its updating. Those consequent forces and its corresponding pattern solutions are described in the following sections.

External library relevance

For real-time storm sewer simulation, several models need to be incorporated with this system. If the external library is based on sequential codes or function-based subsystems such as SWMM, Wrapper Façade is the most preferred pattern to commit the wrapping of such subsystems into a set of delegation interfaces to be manipulated in our developed system. Moreover, focusing on another type of

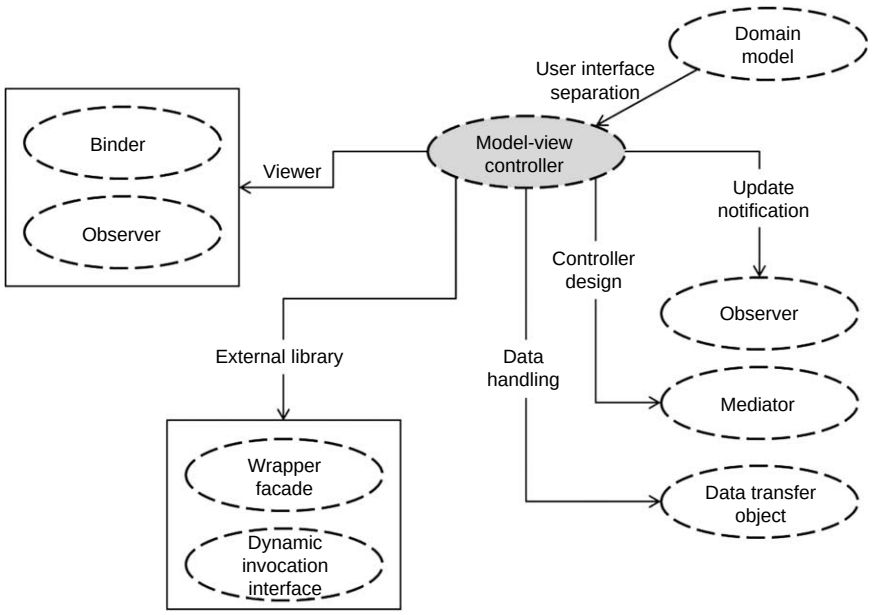

Figure 6 | Pattern topology—model view controller.

external library by dynamic accessing without available source code, Dynamic Invocation Interface is the reasonable candidate solution. Similar to 'reflection' technique, a Dynamic Invocation Interface can retrieve meta-information of invocations in run time such as function names and function accessing pointers. Because function callings are determined in real time, the simulation engine retains very high flexibility and extendibility irrespective of which implementation is involved.

Independent viewer and flexible design for content updating

For 'Viewers' in Model-View-Controller, often representing real-time data such as real-time junction stages or graphical user interfaces, the forces hinge on ensuring flexibility for decoupling representations and business logics. The handling of signal updating for contents achieves fluency. With signal directions, forces can be decomposed further into two sub ramifications, one for one-way direction and the other for two-way direction.

For one-way signal direction, information from logic contents is fed into representations without mutation and the Observer pattern presents as a known public solution, which reaches a state of balance. However, this is not true for two-way signal direction. For this type, logic contents are not only represented in user interfaces, but are likely to be modified after user interactions. Such conditions can include input auxiliary user interfaces that should input current values of SWMM input fields from the logic layer. These values may be changed and fed into logic contents whose user modifications are unsynchronized, due to event-driven user interfaces such as VCL or Visual Basic. Another pattern proposed in this paper, namely the Binder pattern, helps balance this strain. Aiming at event-driven user interface frameworks, a Binder pattern can effectively resolve the specific force for two-way direction signal updating.

Mediation in controller

In the Model-View-Controller pattern, the 'Controller' part represents the most important aspect due to the core of the system being centralization. The greatest force here concerns how to mediate each component of this system, models, viewers or other I/O units and manage them to ensure conflict-free inter-working. Many candidates offer feasible solutions. Concerning the high degree of inter-dependency between each functionality in real-time storm sewer simulations, a preferred solution is to choose a centralized (or 'radial topology' style) for our target design

i.e. a MEDIATOR pattern. The controller is designated as the MEDIATOR. Each developed unit communicates only with the Mediator, and indirectly interacts with other parts via a set of public interfaces provided by the MEDIATOR. This indirect access ensures the opportunities for MEDIATOR to manage inter-communications globally and to avoid corrupt critical sections, signal flows or accessing rules arising from local unit-to-unit interactions.

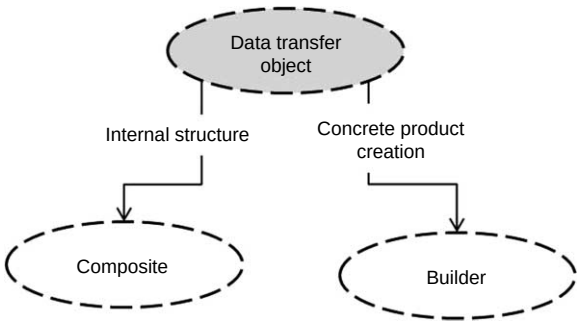

Figure 7 | Pattern topology—data transfer object.

Flexible data handling and update notification

As well as Model, Viewer and Controller, another significant issue should also be considered: internal data handling and update notification. This is important in other similar physical model-based software systems. For example, taking SWMM as the computation kernel, RTS4 should manage complicated I/O data for models, real-time states or real-time signals for storm sewer control. Data handling therefore becomes an important issue and forces which arise here need to be flexibly managed to provide good 'signal roadways' for update notifications. Firstly, a DATA TRANSFER OBJECT pattern is the preferred candidate when it comes to wrapping miscellaneous or messy pieces of information or data into object-style concrete forms that are more amenable to management. Secondly, data update notifications can provide a good avenue of signal passing through the OBSERVER pattern. Despite the above qualifications, a slight difference is that the observer role is no longer the visual component but the data object itself.

Notwithstanding the role that the DATA TRANSFER OBJECT pattern can play in the management of data, consideration should be given to insufficient force balancing. As shown in Figure 7, there are two further component forces: how to provide a supportive and flexible internal data structure for potent management and how to effectively create concrete data entries for final documents. Simply put, two major issues should be taken into account when considering data handling: maintenance management and creational behaviour for data entries.

For the first issue, in providing a flexible tree structure the COMPOSITE pattern is highly capable of accommodating the intrinsic arrangements of data entries. This implies a good ability to buttress the possible higher variety of model-based input/output data fields. Due to the whole-part relationship of tree structures, however complicated the internal structures, the root element (or any element in the arbitrary level of sub-trees) can be wrapped into a single concrete object and easily carried by a DATA TRANSFER OBJECT as a result. The balancing forces are sufficient, not only due to the management of intricate data structures but also to the provision of straightforward data transfer. For the second issue, the creation of data entries aimed at the generation of final products from the internal data structure should be carefully designed to avoid unmanageable codes. In other words, to conform to the variety of formats in the final product the BUILDER pattern should become a feasible candidate to export to other data sources with an extensible and flexible design.

In the second stage, the dynamic view demonstrates the pattern sequences using symbolic notation. While the force decompositions indirectly construct the static topology of patterns used, the forces can also be regarded as the 'road signs' to dynamically voyaging inside the topology. In forces, the regulations that define the approvals or inhibitions determine which pattern sequences are most possible, feasible and reasonable. Those pattern sequences can be represented using symbolic systems. Presently, there is no symbolic system to illustrate specific pattern sequences.

However, two candidate symbolic systems are suitable when presenting pattern sequences. They are the Communicating Sequential Processes (CSP) model (Hoare 1978; Hoare & Jifeng 1999) and Backus–Naur Form (BNF)-like notations. To draw highly abstract concepts (e.g. context-free grammar), BNF is a so-called meta-syntax primarily

proposed to describe the syntax of formal languages by using a set of formal expressions (e.g. another formal syntax). Hence, BNF-like notations can be adopted to represent the grammar for pattern languages e.g. pattern sequences. The symbolic system adopted here is one of the BNF-derived notations, in which ∅ denotes an initial state; → imperative sequential composition; $\rightarrow^{O}$ optional sequential composition; | alternation; and () pattern grouping.

The two most possible pattern sequences are listed below. The first is related to modelling and simulation running and the second to content updating, data handling and rebuilding. They can theoretically be combined into one single passage by applying alternating or optional symbols, but here we prefer to separate them into two passages for clarity.

∅ → Model View Controller → ^{O}((Explicit Interface|(Value Objects → Explicit Interface)) → ^{O}(Wrapper Façade → ^{O}Dynamic Invocation Interface)).

The above symbolic system conveniently represents the pattern sequence as one essential part of pattern languages, which we can reduce to literal writing in plain prose as below.

Model View Controller must be applied, and may optionally be followed by Explicit Interface or alternatively by Value Objects, and must be followed by Explicit Interface. Explicit Interface may optionally be followed by Wrapper Façade, which if applied may optionally be followed by Dynamic Invocation Interface.

This can be understood to mean that every simulation run is triggered by the main application layout, Model-View-Controller. In turn, Explicit Interface is provided for better initial access to each simulation run. Alternatively, where performance issues quickly identify static states necessary for simulation or meaningful types of information that simulations involve, forces will probably bring out a Value Object prior to an Explicit Interface. After triggering, forces move forward into ambidextrous situations, according to the kind of external library to be applied. The Wrapper Façade is applied to ensure good integration with procedure-based subsystems, the real implementations of which can possibly be located elsewhere for compatibility with the application of a Dynamic Invocation Interface.

The second is an example that helps outline how force regulations result in a pattern sequence, as described below.

∅ → ^{O}(Binder → ((Data Transfer Object → Composite)|(Mediator → Data Transfer Object → Composite)) → Observer → ^{O}Builder → (Data Transfer Object → Composite)).

As with the first example, this can be regressed to a prosaic description, as follows.

Binder may be optionally applied. If applied, Binder must be followed by Data Transfer Object, which must be followed by Composite. Alternatively, if applied, Binder must be followed by Mediator, which must be followed by Data Transfer Object, which must be followed by Composite. Composite must be followed by Observer, which may be optionally followed by Builder. If applied, Builder must be followed by Data Transfer Object, which must be followed by Composite.

As far as practical implementation considerations are concerned, most GUIs are completed by selected widgets. Binder is therefore the first pattern taken here to provide a solution. Binder does not involve in particular coordination for selected widget toolkits, but provides another way to preserve the flexibility of toolkit changes. After that, forces enabling better data handling facilitate subsequent application of the Data Transfer Object. A Composite always follows, or it can be denoted as an immutable pattern composition: <Data Transfer Object, Composite>. This composition carries data entries transferred by a Binder where possible modifications would make Mediator

intervene prior to this composition, involving other local logic units. In doing so, complicated data accessing can be carried out while avoiding direct interaction, which may otherwise have fostered unmanageable relationships. After modification, OBSERVER is the most sensible solution to execute update notification for corresponding contents. Accordingly, BUILDER might be involved to perform concrete data entry regenerations, whose data manipulation is taken by the pattern composition <DATA TRANSFER OBJECT, COMPOSITE > again.

Concrete design by patterns

In this section, the concrete design view is laid out by the pattern topologies and sequences proposed in the previous section. Each pattern can be seen as a single 'vocabulary' in pattern languages, as well as the 'leaf' of the ramification of the applied pattern. That implies an atomic element in design, in order to ensure that it can be directly applied as one building block in the concrete design structure to be codified. Therefore, in this section, we demonstrate how concrete design structures can be stretched out following procedures in pattern languages for the RTS4.

Any pattern language only provides a general holistic solution rather than a specific one. That makes the concrete design structure varied and dependent on implementation favours. Conceptually, for one pattern, any structure can be accepted as long as it does not contravene the organizing principles. We therefore only propose one version to adopt into implementations.

First of all, pattern languages reveal the entrance point starting from the MODEL-VIEW-CONTROLLER pattern. This is depicted in Figure 8 which comprises three main blocks: a main controller, models and viewers. The main idea depicted in Figure 9 is to wrap and trigger models by EXPLICIT INTERFACE, aggregated by a controller which

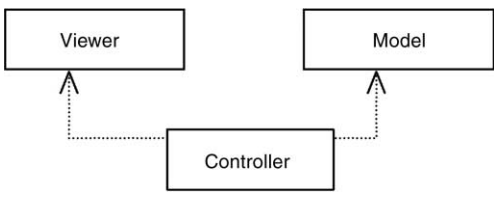

Figure 8 | Design structure—model view controller.

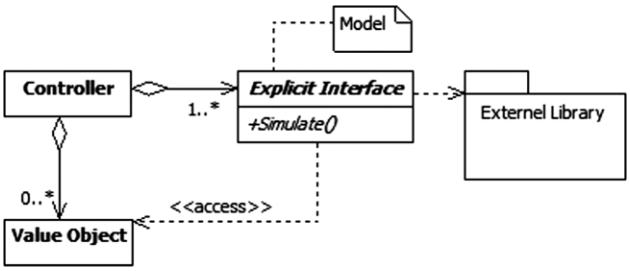

Figure 9 | Design structure—explicit interface-value object.

handles VALUE OBJECT that might be accessed by models. Real models may be implemented as an external library or as other sources or remote services.

The concrete design example shown in Figure 10 can be drawn into one actor, SewerSimulator. It is designed with a 'Run()' abstract interface that EXPLICIT INTERFACE utilizes to trigger SWMM and give simulated sewer hydrodynamic status. Figure 10 depicts two examples implemented in RTS4 that are necessary for the storm sewer simulation to achieve its goals. A RealTimeSimulator is responsible for real-time simulation interaction with possible real-time operation. A TrivialSimulator, on the other hand, is responsible for typical sewer simulations without any real-time matters. The flexibility for switching models without perturbing other codes also can be provided by EXPLICIT INTERFACE, by means of simply implementing those models in accordance with concrete subclasses.

In contrast to implementing real-time simulations directly by incorporating SWMM, a better way is to apply a WRAPPER FAÇADE pattern to separate and wrap SWMM as an individual SWMMWrapper and to give necessary public interfaces for real-time simulation. Consequently, variations of SWMM are decoupled and encapsulated so that developers only focus on real-time control achievements, regardless of what really happens inside SWMM. In RTS4 pattern languages, we say that EXPLICIT INTERFACE is followed by WRAPPER FAÇADE. Therefore, WRAPPER FAÇADE can be followed by DYNAMIC INVOCATION INTERFACE, which allows the wrapped SWMM engine to be realized as different concrete entries, i.e. SWMM DLL (Lin *et al.* 2006) for Windows systems or remote SWMM Services.

To retrieve or update static status, VALUE OBJECT is designed to deliver efficiency. This design coerces all VALUE OBJECTS to be represented by public plain data (POD)

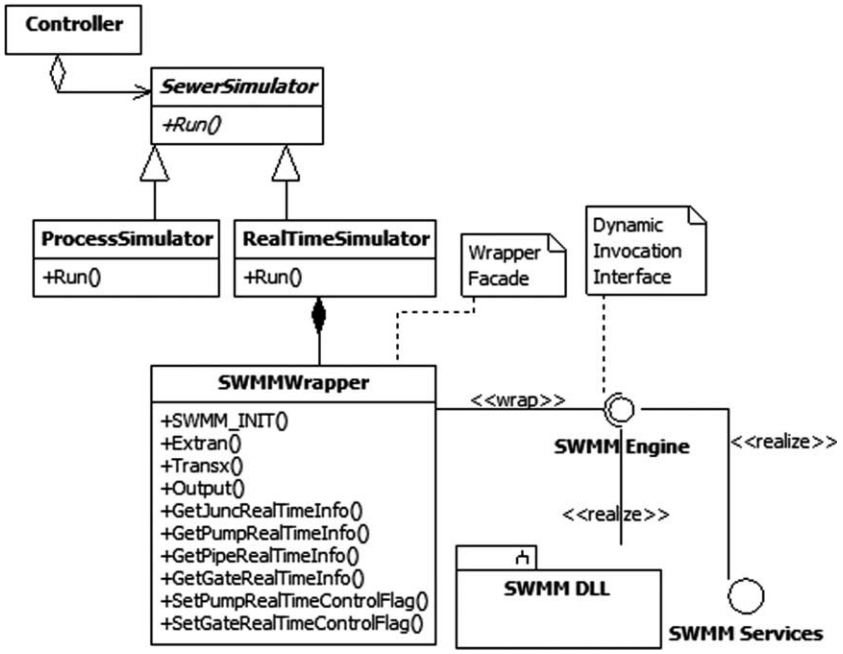

Figure 10 | Design structure—wrapper facade-dynamic invocation interface for sewersimulator.

fields. Figure 11, for example, demonstrates all types of VALUE OBJECT designed for RTS4. With the exception of several basic fields, the most important point in this design is the status records in time horizon. Solving unsteady hydrodynamics implies that logging profiles for each temporally varied status is very important for either simulation requesting or data demonstrations. Solving by VALUE OBJECT is much better than by a primitive array, since meaningful type information can be maintained without sacrificing efficiency.

Flexible design for data handing and management is followed by Viewer design and contents update notification design. A main use in RTS4, shown in Figure 12, is to employ this pattern composition to handle an SWMM EXTRAN input file as well as RTS4 project management information. Because of the COMPOSITE pattern, a tree

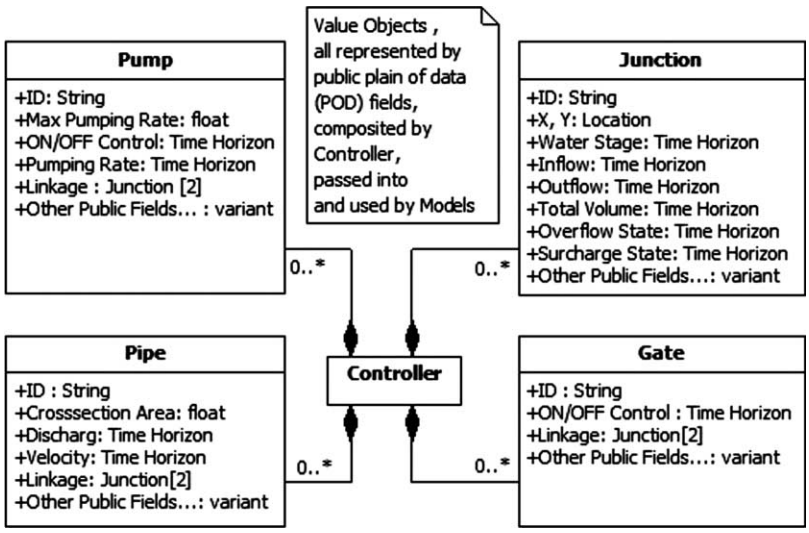

Figure 11 | Design structure—value object.

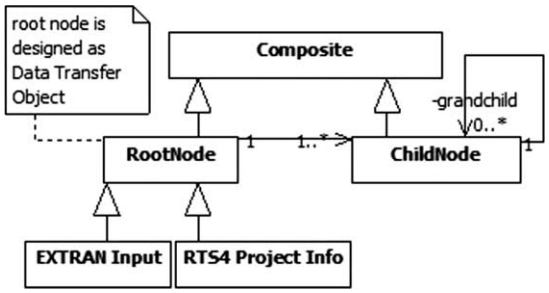

Figure 12 | Design structure—composite-data transfer object.

structure comprising a root node with several child nodes and grandchild nodes gives the flexibility to meet high variations of data structure. Only the root node is wrapped as a DATA TRANSFER OBJECT to assist the smooth passing of a single object for transfer and exchange.

Flexible design for Viewer and contents updating can be roughly categorized into two groups: two-way signalling or one-way signalling. As already described, their solutions are BINDER pattern or OBSERVER pattern, respectively. Regarding the BINDER pattern for two-way signal updating in RTS4, it is applied to Viewers developed by any widget toolkits, whose mutable contents should be bound to system data structures.

As shown in Figure 13, for example, input GUI auxiliaries are developed from existing forms of widgets. In turn, two additional interfaces given by BINDER are implemented for data binding details: BindFromBinding-Classes() and BindToBindingClasses() method. Data entries, implemented as a BINDER pattern, are wrapped as a DATA TRANSFER OBJECT to be carried by those two interfaces as one reference pointer for easy manipulations in input GUI auxiliaries. Before any modification by users, the implementations for the first interface bind the current data and fill them into mapped GUI fields. After several possible changes of those fields, the second interface is triggered in which implementations update those data entries that need to conform to the current UI fields. In turn, the pattern composition <DATA TRANSFER OBJECT, COMPOSITE> might be applied in this context. In pattern languages, we can therefore say that BINDER is followed by DATA TRANSFER OBJECT, which must be followed by COMPOSITE.

As for a typical one-way signalling update, an OBSERVER pattern is the most commonly used solution. Despite being widely used, its design for details is subject to variations and the proposed pattern differs slightly from other typical patterns. The major consideration comes from different

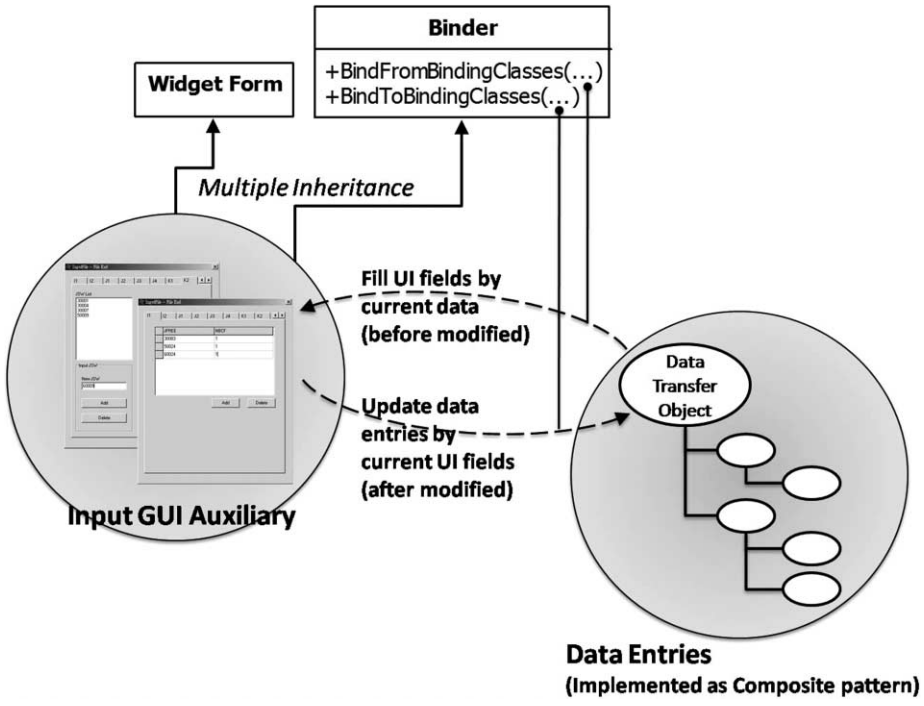

Figure 13 | Design structure—binder-data transfer object-composite.

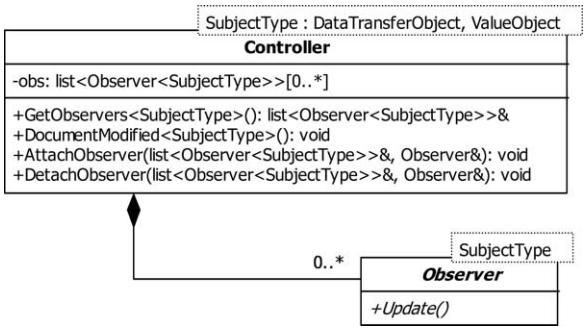

Figure 14 | Design structure—observer.

subject types announcing notifications for different updates: Data Transfer Object or Value Object. Any changes from either subject would inform their own observers only. In order to describe the purpose, a subject type-binding approach is taken combined with an Observer pattern (as shown in Figure 14). Invokers provide subject type-bound standard interfaces for the Observer pattern, in which the DocumentModified() function would notify only observers that are related to this pre-bound subject type.

Observer updating for Data Transfer Object contents is mainly about concrete data entries exporting or rebuilding. The Builder pattern subsequently steps in to collaborate with the Observer pattern. As adopted by the RTS4 design, the structure shown in Figure 15 is the most plausible. By giving a set of general interfaces, the Builder pattern

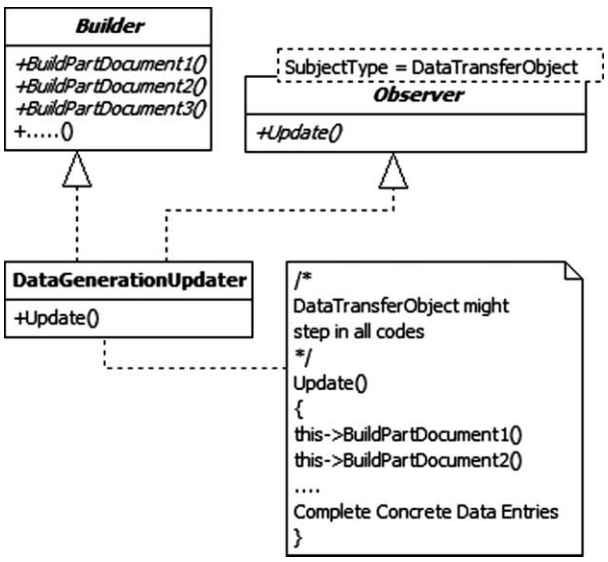

Figure 15 | Design structure—builder-observer.

generates concrete data entries for different exporting products such as XML documents, databases or EXTRAN input text files. The DataGenerationUpdater takes the changes from the Data Transfer Object and forwards it to Builder interfaces for data entry rebuilding. These eventually complete newly refreshed concrete data entries, such as newly refreshed EXTRAN input text files. The pattern languages are consequently matched.

IMPLEMENTATIONS

Conceptually, the proposed pattern-oriented design for real-time storm sewer simulation should not be bound to any specific tools, environments, programming languages, or operation systems. However, to give an example that will demonstrate feasibility, our product RTS4 is implemented with reference to real engineering problems. The development of RTS4 uses C++ programming language and selects Borland C++ Builder 6.0 (BCB6) as the development environment, as well as now only working for the Windows platform. BCB6 provides a Visual Component Library (VCL) framework for simple GUI development (Holling *et al.* 2003), regarded as a pre-adopting 'widget toolkit' and later adapted for RTS4 by Binder pattern. As for the models, the SWMM engine is wrapped as an external DLL and adapted for RTS4 by Dynamic Invocation Interface and wrapper façade. Legacy model linking details follow the approach by Lin *et al.* (2006).

APPLICATIONS

In order to demonstrate the applicability of RTS4, one example is selected with respect to the main use case: real-time operation simulation. A practical engineering problem selected here is the Zhung-Gung drainage system, located in the south of Taipei City. The design storm takes place over a 5-year return period and has a 90-minute duration. Before sewer hydrodynamic simulation, the rainfall hyetograph is first provided for the SWMM-RUNOFF module to generate a runoff hydrograph through kinematic wave routing. It is then taken as an inflow hydrograph for inputs of this drainage system.

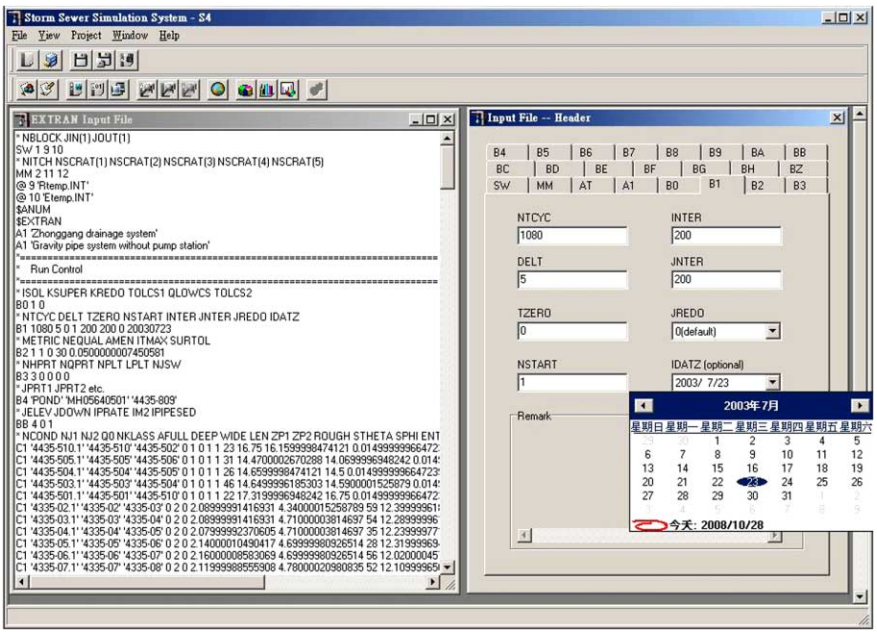

Figure 16 | Input file text editor vs input GUI auxiliary.

Users can first edit the input file through two different ways demonstrated in Figure 16. An advanced user who is familiar with the EXTRAN input file format can directly edit the plain text file on an editor (shown on the left) whereas a user who is not so familiar can easily edit input fields using a set of input auxiliary windows. Modifications from either one will be further synchronized with each other, whose pattern-oriented

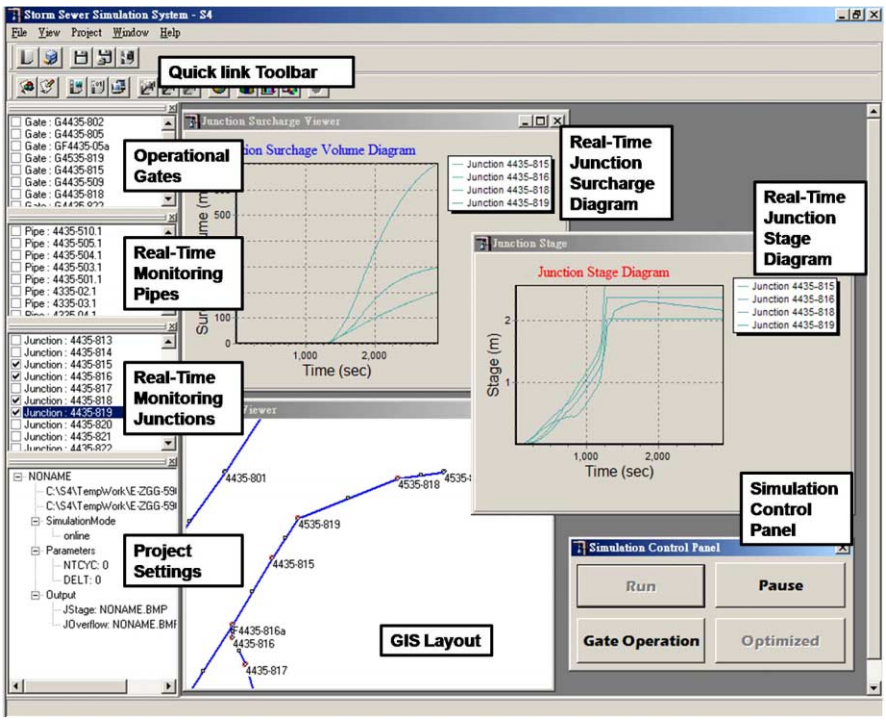

Figure 17 | RTS4 screenshot of real-time storm sewer simulation.

designed notification-update mechanisms work very well for this purpose.

Figure 17 shows the main screenshot as RTS4 is performing a real-time storm sewer simulation. The top is a quick link toolbar. Forms arrayed in the left part are the selectable operational gate list, the selectable real-time monitoring pipe list, the junction list and project settings. Several real-time visualizers, such as a real-time junction surcharge diagram, real-time junction stage diagram or GIS layout viewer, can be opened in the main window. The simulation control panel consists of four options: 'Run' starts or resumes simulation, 'Pause' suspends simulation and 'Gate Operation' performs real-time operations, and 'Optimized' executes automatically optimal real-time simulation.

We initially select junctions 4435-815, 4435-816, 4435-818 and 4435-819 (Figure 17) for real-time monitoring, and select 'Run' to trigger the simulation. Real-time information responses including junction stage and surcharge volume, as well as red-circle warning icons in the GIS layout viewer, appear while that surcharge of junction is taking place. At any time during the simulation, we can select what operational gate will be working when the 'Gate Operation' button is selected, i.e. in this case G4535-819, G4435-815 and G4435-818. As shown in Figure 18, after the launch of real-time operation launch, all monitoring stages drop significantly. As long as the operating gates are released, that is, real-time control has been turned off, junction stages then accordingly go bounceback. Water stages suddenly arise from low level and continue to keep high values.

This fact demonstrates effective reductions of water stages for specific junctions due to real-time gate operations. Due to a well-designed software system, delicate interworking between many real-time status changes and SWMM-EXTRAN numerical response can be seamlessly integrated. Engineers can arrange a set of 'what-if' control scenarios to see how the surcharge flooding can be mitigated by performing real-time control of hydrological facilities. Due to very limited observation data available in Taipei city, it is difficult to verify numerical results. However, instead of predicting precise values using RTS4 and a set of control scenarios, a 'what-if' analysis can be applied for practical cases. The aim is to understand the

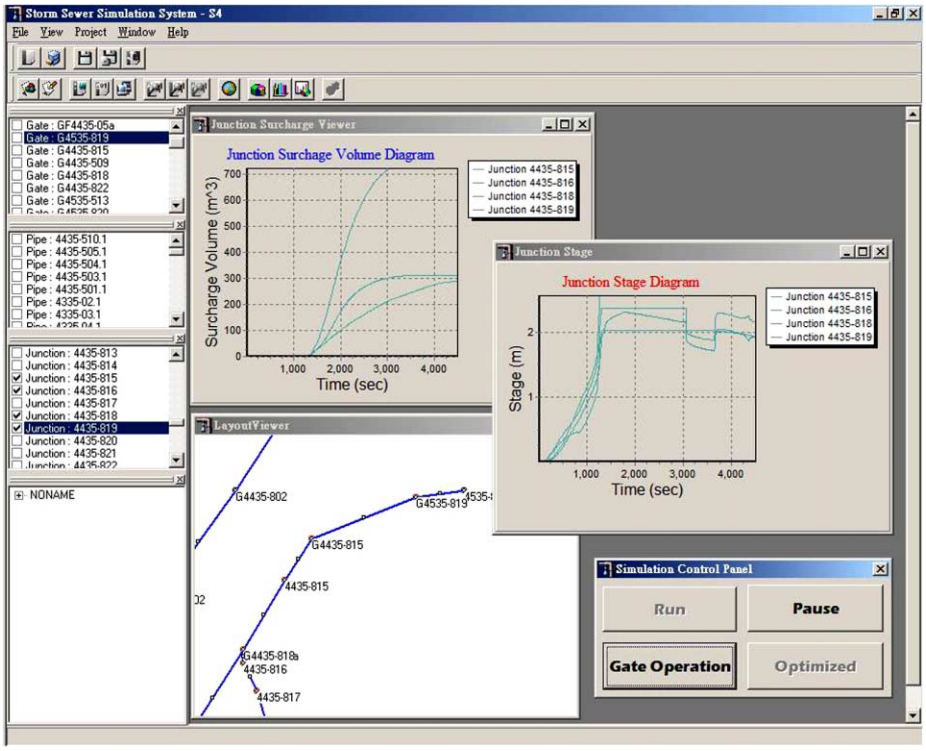

Figure 18 | The results of real-time operation.

differences between responses due to different control scenarios, in order to determine the best scenario and achieve maximal mitigation for surcharges, rather than to predict precise physical quantities.

CONCLUSIONS

This paper has proposed a pattern-oriented approach to software design for a real-time storm sewer simulation system. By a top-down strategy, the process commenced with system requirement analysis, followed by conceptual model design, pattern-languages, pattern-oriented design and finally implementations and applications.

Three main conclusions can be drawn: (1) a top-down standard strategy employed in object-oriented software analysis and design is successful in the development of a hydrodynamic-related simulation system; (2) this study marks the first ever proposal of pattern-languages and pattern-oriented design for software design for a real-time storm sewer simulation; and (3) our software system (RTS4) successfully followed the proposed pattern-oriented design, and the results demonstrate its potential applicability to real-time operation simulations.

REFERENCES

Alexander, C. 1979 *The Timeless Way of Building*. Oxford University Press, New York.

Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I. & Angel, S. 1977 *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, New York.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. & Stal, M. 1996 *Pattern-Oriented Software Architecture. A System of Patterns*, Vol. 1. John Wiley & Sons, Chichester, UK.

Buschmann, F., Henney, K. & Schmidt, D. C. 2007 *Pattern-Oriented Software Architecture. On Patterns and Pattern Languages*, Vol. 5. John Wiley & Sons, Chichester, UK.

Chaudhry, M. H. 1993 *Open-Channel Flow*. Prentice-Hall, Englewood Cliffs, NJ.

Coplien, J. O. 2004 A Pattern Definition, http://hillside.net/patterns/definition.html

Fowler, M. 1996 *Analysis Patterns: Reusable Object Models*. Addison-Wesley, Reading, Massachusetts.

Fowler, M. 2003 *Patterns of Enterprise Application Architecture*. Addison-Wesley, Reading, Massachusetts.

Gamma, E., Helm, R., Johnson, R. & Vlissides, J. 1995 *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, Massachusetts.

Hoare, C. A. R. 1978 Communicating sequential processes. *Commun. Assoc. Comput. Mach.* **21** (8), 666–677.

Hoare, C. A. R. & Jifeng, H. 1999 A trace model for pointers and objects. *Proceedings of ECOOP 1999*, Springer, Berlin.

Holling, J., Swart, B., Cashman, M. & Gustavson, P. 2003 *Borland C++ Builder 6 Developer's Guide*. Sams, Indiana.

Huber, W. C. & Dickinson, R. E. 1988 *Storm Water Management Model. User's Manual* Ver. IV. U.S. Environmental Protection Agency.

Larman, C. 2004 *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

Lin, S. S., Hsieh, S. H., Kuo, J. T., Chen, Y. C. & Liao, Y. P. 2004 Integrating SWMM with optimization module for feasibility of real-time control in urban drainage systems. *Proceedings of the 6th International Conference on Hydroinformatics* 2, pp. 1899–1906, Singapore.

Lin, S. S., Wang, L. P., Hsieh, S. H., Kuo, J. T. & Chen, Y. C. 2005 An approach for modeling gate operations under surcharge in urban drainage systems. *Proceedings of the International Conference on Monitoring, Prediction and Mitigation of Water-Related Disasters*, pp. 203–208, Kyoto, Japan.

Lin, S. S., Hsieh, S. H., Kuo, J. T., Liao, Y. P. & Chen, Y. C. 2006 Integrating legacy components into a software system for storm sewer simulation. *Environ. Model. Softw.* **21** (8), 1129–1140.

Massingill, B. L., Mattson, T. G. & Sanders, B. A. 2000 A pattern language for parallel application programs. *Lect. Notes Comput. Sci.* **1900/2000**, 678–681.

Object Management Group 2005 UML 2.0 Infrastructure Specification, http://www.omg.org/spec/UML/2.0/

Park, H. & Johnson, T. J. 1998 Hydrodynamic modeling in solving combined sewer problems: a case study. *Water Res.* **32** (6), 1948–1956.

Rising, L. 1999 Design patterns in communications software. *Commun. Magazine* **37** (4), 32–33.

Tsihrintzis, V. A., Sylaios, G. K., Sidiropoulou, M. & Koutrakis, E. T. 2007 Hydrodynamic modeling and management alternatives in a Mediterranean, fishery exploited, coastal lagoon. *Aquacult. Eng.* **36** (3), 310–324.

Zaghloul, N. A. 1998 Flow simulation in circular pipes with variable roughness using SWMM-EXTRAN model. *J. Hydraul. Eng.* **124** (1), 73–76.

Zhao, L., Macaulay, L., Adams, J. & Verschueren, P. 2008 A pattern language for designing e-business architecture. *J. Syst. Softw.* **81** (8), 1272–1287.

First received 5 April 2009; accepted in revised form 4 July 2009. Available online 31 March 2010