

Integrating legacy components into a software system for storm sewer simulation

Shiu-Shin Lin^a, Shang-Hsien Hsieh^{a,*}, Jan-Tai Kuo^b,
Ying-Po Liao^a, Yen-Chang Chen^c

^a Department of Civil Engineering, National Taiwan University, Taipei 10617, Taiwan

^b Department of Civil Engineering and Hydrotech Research Institute, National Taiwan University, Taipei, Taiwan

^c Department of Civil Engineering, National Taipei University of Technology, Taipei, Taiwan

Received 4 February 2004; received in revised form 27 November 2004; accepted 13 May 2005

Available online 27 July 2005

Abstract

This paper presents an approach that integrates a legacy component into a software system for storm sewer simulation. The legacy component employed here is the Storm Water Management Model (SWMM). The Extended Transport (EXTRAN) block of the SWMM that applies the finite difference method (FDM) with explicit numerical schemes, solving the de Saint-Venant equations, is used to route the storm sewer flow. A storm sewer simulation system, named S4, that integrates SWMM-EXTRAN and implements a visualization model, has been developed to demonstrate the proposed approach. The approach makes use of the multi-thread technology to alternate the execution between SWMM-EXTRAN for flow simulation on one thread and the program controller that updates simulation state variables and displays the computed temporal water-stages at the junctions on the other thread at every time step of the FDM process. Two test examples are used to verify and demonstrate the feasibility of the proposed approach. The results show that the multi-thread technology is applied successfully for integrating legacy components, such as SWMM-EXTRAN, into a software system (in this case, S4). In addition, the proposed approach is generally applicable for integrating legacy models or components developed using FDM with explicit numerical schemes.

© 2005 Elsevier Ltd. All rights reserved.

Keywords: Storm sewer system; Storm Water Management Model (SWMM); Legacy components; Integrated modeling; Multi-thread; Inter-process communication; Language integration

1. Introduction

The capacity of storm sewers can be overtaxed occasionally and water rises in manholes, inundating the urban areas. Thus monitoring sewer flow for large cities becomes necessary and prevention or mitigation of damages due to inundation is always a concern. A storm sewer simulation system that receives the real-time monitoring rainfall data as system input to predict the

future states of the storm sewer system can help make decisions to reduce the damage during the storm events. This kind of system usually includes at least a program controller, a storm sewer simulator (or solver), and a visualization module. The program controller asks the storm sewer model to simulate the sewer flow based on the real-time rainfall data and passes the current states computed by the storm sewer model to the visualization model for immediate displays.

The flow conditions in storm sewers vary with time, which can be determined by proper hydrodynamics models. The de Saint-Venant equations, also referred to as the shallow water equations (Chaudhry, 1993),

* Corresponding author. Tel./fax: +886 2 2368 8213.

E-mail address: shhsieh@ntu.edu.tw (S.-H. Hsieh).

are the most popular hydrodynamics to describe the flows in channels and/or storm sewers. The continuity and momentum equations of the unsteady flow are depicted by the following equations, respectively:

$$\frac{\partial A}{\partial t} + \frac{\partial Q}{\partial x} = 0 \quad (1)$$

$$\frac{\partial Q}{\partial t} + \frac{\partial(Q^2/A)}{\partial x} + gA \frac{\partial H}{\partial x} + gAS_f = 0 \quad (2)$$

where Q , discharge; A , flow cross-sectional area; H , hydraulic head; x , distance along the pipe; t , time; g , gravitational constant; S_f , energy slope.

Many numerical models available today adopt numerical schemes to the solutions of full de Saint-Venant equations. For instance, models like SWMM-EXTRAN (Huber and Dickinson, 1988) and SUPERLINK (Ji, 1998) employ explicit and implicit schemes, respectively, to obtain the solutions.

The finite difference method (FDM) with explicit numerical schemes (Explicit-FDM) that require less memory and computation time is proper for dealing with storm sewer hydrodynamics. However, most legacy Explicit-FDM solvers, such as SWMM-EXTRAN, take a priori design storm and/or historical rainfall event as input data and then predicts the sewer flow at any time period of the known rainfall events. They cannot avoid re-starting the solver from the very beginning of the rainfall events whenever the rainfall data is given in real-time. It would be much better and saves much time if the program controller can suspend the execution of the solver at every time step and provide the solver with the real-time rainfall data for the computation of the next time step. Moreover, before the controller resumes the execution of the solver, it can obtain the computed results from the solver and ask the visualization module to display them. To achieve the integration of the legacy solver into a storm sewer simulation system capable of simulating the flow condition with real-time monitoring rainfall as described above, the interaction and message passing among the legacy solver, the program controller, and the visualization module need to be carefully addressed.

Concepts as well as technologies have been proposed to integrate the legacy models to accurately describe the storm sewer simulation or provide a friendly user interface that makes these models easier to use. The term “legacy component” or “legacy model” is used here to refer to software with older-technology (Lutz, 1995), typically in the form of Fortran subroutines (Ewer et al., 1995; Achee and Carver, 1997; Neil et al., 1999; Tolsma and Barton, 2000; Sang et al., 2002; Guo, 2003). The approaches for integrating legacy components into a software system can be classified into two groups with

different levels of difficulty. The approaches in the first group usually integrate legacy components through their input/output files and involve no modification on the legacy source code. They are therefore simpler and more straightforward. For instance, the well-known software package PCSWMM, which makes use of the executable SWMM as the simulation module, provides a friendly graphical user interface (GUI) to make SWMM easy to use. Hsu et al. (2000) integrated two legacy models, i.e. a storm sewer model and a two-dimensional inundation model, to simulate inundation problems. The two legacy models are executed in sequence and no modification on their source codes is required. The inundation model is always waiting for the output from the storm sewer simulation model. The integration approaches in the second group require some modifications on the legacy source code to achieve more seamless integration between the models. Message passing between the models is often the key issue in these approaches. The integrated modeling discussed by Rauch et al. (2002), which is the approach that simulates the interaction between two or more physical systems, belongs to this group. Many studies have focused on this group of approaches in the last decade. They include the studies on how to integrate the legacy models into an existing object-oriented framework (Neil et al., 1999; Riche et al., 2003) and into a flowsheet (Tolsma and Barton, 2000). The applications of integrated modeling in urban drainage or wastewater systems were also proliferated (Rauch and Harremoés, 1996, 1999; Schütze et al., 1996, 1999; Lin et al., 2003; Butler and Schütze, 2005). It has been shown that the task to integrate sub-models with different state variables into a single software tool can be quite complex (Rauch et al., 2002).

This paper presents an integration approach that belongs to the second group described above. The approach takes advantage of the multi-thread technology to integrate legacy components. A storm sewer simulation system, named S4, that integrates the legacy SWMM-EXTRAN component and implements a visualization module, has been developed to demonstrate the proposed approach. With the multi-thread technology, S4 is able to suspend and resume the execution of SWMM-EXTRAN for flow simulation with real-time rainfall data on one thread during runtime and display the computed temporal water-stages at the junctions of the storm sewer system on the other thread at every time step of the Explicit-FDM process. One targeted application of the proposed approach is the real-time control simulation in urban drainage systems.

The remaining sections of this paper are organized as follows. Section 2 explains the background of the storm sewer model, i.e. SWMM, adopted and the design of the components in S4. Section 3 discusses the choices of programming language for the development of S4 and describes how message passing among S4 components

during runtime can be achieved through inter-process communication supported by the multi-thread technology. In Section 4, the discussions are on the implementation details of S4. Section 5 then uses two examples, an idealized drainage system and a real drainage system located in the Taipei City, to study and demonstrate the feasibility of the proposed approach. Finally, some conclusions are drawn in Section 6.

2. Design of the storm sewer simulation system

S4 is composed of four major components, i.e. the Simulator, Controller, Visualizer, and SWMM_Adaptor, as shown in Fig. 1. The Simulator is used to simulate the storm sewer flow. This work takes advantage of the legacy SWMM model to implement the Simulator. Further discussion on the Simulator is given later in the next paragraph. The Controller is the component to manage the flow of simulation. It suspends the execution of the Simulator at the end of every time step and receives information, such as the computed water-stages at the junctions in the storm sewer system, from the Simulator through the help of the SWMM Adaptor. It then calls the Visualizer to display the received information immediately. The SWMM Adaptor is designed as an object of a user-defined type (Stroustrup, 1997) that encapsulates the data (i.e. data member in the field of Object-Oriented Programming) used by both the Simulator and the Controller, and provides the methods (i.e. functions or subroutines) for both the Simulator and the Controller to operate on the data. During the simulation, the Simulator calls the methods of the SWMM Adaptor to set values into the data members of the SWMM Adaptor and then the Controller calls the methods to retrieve the data from the SWMM Adaptor.

The design of the Simulator of the S4 system takes advantage of the SWMM legacy model, which is developed by the U.S. EPA and has been widely applied in the fields of urban hydrology (Liong et al., 1995; Zaghoul, 1997; Park and Johnson, 1998; Campbell and

Sullivan, 2002). SWMM contains many blocks for calculating the quality and quantity processes of runoff in urban area. The RUNOFF block in SWMM is employed here to determine the relationship between rainfall and runoff. The EXTRAN block in SWMM is employed to route storm sewer flow by using an explicit numerical scheme to solve the one-dimensional Saint-Venant equations. Fig. 2(a) shows the main flow sequence of the application of SWMM for simulating storm sewer flow. SWMM starts to run in the main routine that calls the RUNOFF routine to calculate the

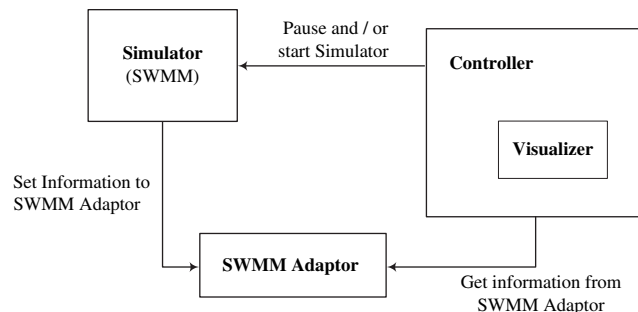
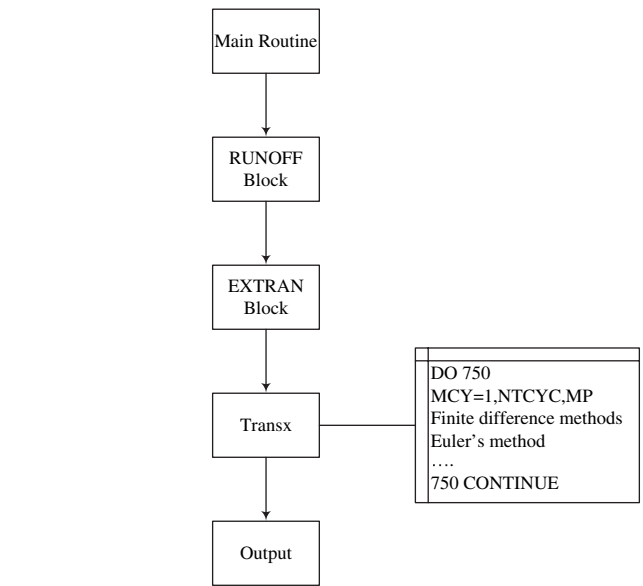
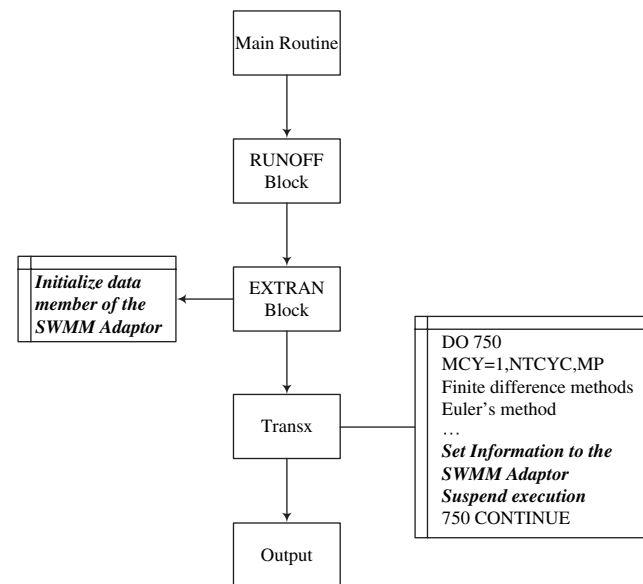


Fig. 1. System architecture.



(a) Original sequence of SWMM



(b) Modified sequence of SWMM

Fig. 2. Structure of SWMM: (a) original sequence of SWMM; (b) modified sequence of SWMM.

runoff quantity of a rainfall event. The EXTRAN block is the kernel for routing the storm sewer flow. The hydrograph of runoff, calculated by the RUNOFF block, is taken as part of the input file to run EXTRAN. Subroutine Transx, called by the subroutine EXTRAN, is the main procedure to solve the Saint-Venant equations with modified Euler's method. A large loop within the subroutine Transx performs the calculation of storm sewer flow and saves them into intermediate files. Finally, summary information is saved into an output file by calling the subroutine Output.

To allow the Controller to retrieve the computed results at the end of every time step of the loop of Transx, the original sequence of SWMM needs to be modified as shown in Fig. 2(b) with *Italic bold face*. The data member of the SWMM Adaptor is initialized in the EXTRAN block. The subroutine Transx passes the computed results to the SWMM Adaptor at the end of each looping step and suspends the execution to wait for the Controller to process the results (e.g. visualization of the results). More discussions are provided in Section 3 on the programming languages and corresponding development tools used to implement S4 and the multi-thread technology used for message passing between the Simulator and the Controller. Moreover, the implementation of S4 is discussed in Section 4.

3. Integration strategies

The successful integration of the legacy SWMM model into the S4 system requires careful consideration on the use of programming languages and corresponding development tools as well as the message passing mechanism between the Controller and the Simulator during runtime. This section discusses the consideration of this work and the strategies employed.

3.1. Programming languages and development tools used for integration

Several programming languages can be used to implement the S4 system. They are investigated and compared in this section. In this work, the popular Microsoft Windows, such as Windows NT, Windows 2000, or Windows XP, is the operating system considered and the version 4.4 of SWMM, which has the source distribution on Win32 platform, is employed.

Because SWMM has been programmed in the Fortran 77 programming language, it is natural to select Fortran as the programming language for development of S4. Moreover, Fortran 90/95 is chosen over Fortran 77 because the former has more advanced features. For instance, Fortran 90/95 provides many new features such as *module*, which is very suitable for extending the legacy SWMM and makes construction of user-defined

types easy. The module of Fortran 90/95 can also be used to replace the common block of Fortran 77 and bind both data and subroutines. Wrapping data and subroutines into modules is just like defining classes in object-oriented programming except that modules cannot be inherited. Compaq Visual Fortran (CVF) version 6.5 is a Fortran 90/95 compiler on Win32 platform that can compile SWMM version 4.4 without any problem. However, in the Win32 environment, CVF lacks libraries and tools to construct GUI software systems.

Another popular programming language that can be used to develop the non-SWMM part of S4 is C++. The C++ programming language was developed during 1970s with high performance in system programming and provides many programming paradigms, such as procedure-oriented, object-oriented, and generic programming (Stroustrup, 1997). Combining Fortran 90/95 with C++ is simple and has good runtime efficiency. The Microsoft Visual C++ 6.0 (MSVC6) and Borland C++ Builder 6.0 (BCB6) are the two well-known development tools of C++ on Windows platforms. MSVC6 and BCB6 provide the Microsoft Foundation Class (MFC), a general purpose C++ class library, and the Visual Component Library (VCL), which is a Rapid Application Development (RAD) tool, respectively, for developing Win32 applications. This work selects VCL over MFC because the learning curve of MFC is steeper and VCL is easier to use. This also means that the implementation task can be done using VCL in a shorter period of time and the maintenance and extension work in the future would be easier. However, the use of BCB6 faces a problem when it comes to integration with the SWMM part of the program compiled by CVF. The object file (.lib or .dll) built under CVF is in the Common Object File Format (COFF), while the object files (.exe) compiled and linked in BCB6 is in the Object Module Format (OMF). The COFF format is not compatible with the OMF! Fortunately, BCB6 provides a utility, called implib.exe, which can transform a COFF file into an OMF file and solve the incompatibility problem.

The other programming language for consideration is Python. Python is an object-oriented script language and is good for software integration (Lutz, 2001). It is designed to integrate with other Common Object Request Broker Architecture (CORBA) and Component Object Model (COM) components and modules, which have been programmed in C/C++, Fortran, and Java. The Pyfort module in Python can be used to integrate the legacy Fortran programs. Although Python's friendly features for software integration make it an attractive choice, its lack of runtime efficiency is a great disadvantage, especially for development of computationally intensive software.

Based on the above discussions and comparisons, it can be seen that the combination of C++ and Fortran

90/95 is the best choice for the program development of S4 and the corresponding development tools selected should be BCB 6.0 and CVF 6.5.

3.2. Inter-process communication strategy

Message passing between the Controller and the Simulator in the S4 system can be accomplished using Inter-Process Communication (IPC) technology. Two types of IPC technology are discussed here and one of them is adopted for implementing the message passing mechanism needed in S4. The first type of IPC technologies deals with message passing between different processes (called multi-process IPC here), while the other one deals with message passing between different threads in a single process (called multi-thread IPC here).

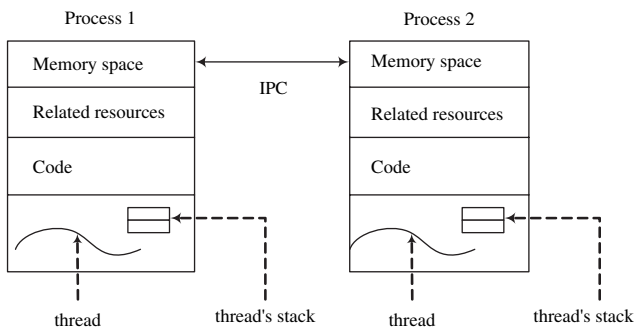
A process contains a private virtual address space (virtual memory), an executable program, data, a list of system resources (e.g. files), and at least one thread for execution (Solomon and Russinovich, 2000; Tanenbaum, 2001). Fig. 3 shows the structure of a process. It includes its own memory space, the variables stored, related

resources (e.g. open files and child process), and one or more threads to execute the code. A thread is the basic execution unit on an operating system, such as MS-Windows, and has its own stacks, program counter, registers, and state (Tanenbaum, 2001). The stack of a thread saves the thread's own variables. A thread uses its program counter to keep track of the execution sequence of the instructions. The registers of a thread record the current values of the working variables. A thread can be at the state of running, suspending, or resuming. Because the thread has the same properties of the process, it is also called lightweight process (Tanenbaum, 2001).

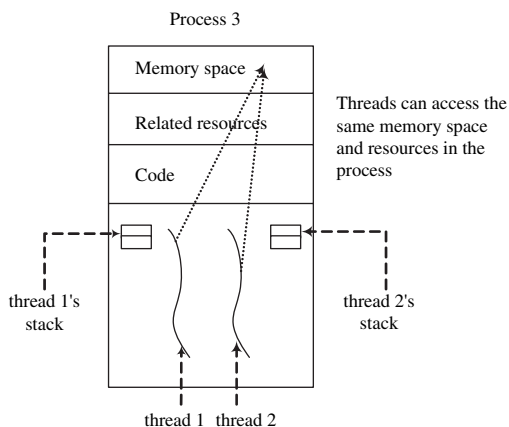
In multi-tasking operating system like Windows 2000 and Windows XP, many processes run concurrently in the sense of time slicing and sharing supported by the operating system. The system resources of processes are independent of each other, thus different processes cannot exchange and share data directly. For example, two processes, namely Process 1 and Process 2, as shown in Fig. 3(a), have their private memory spaces, related resources, and threads to execute their own codes. The variables stored in the memory space of Process 1 cannot be manipulated directly by Process 2, and vice versa. Many multi-process IPC technologies, including pipes, named pipes, mail slots, socket, remote procedure calls (RPC), and shared files, can be used to achieve data exchange and sharing between different processes (Tanenbaum, 2001). The pipes are used for the processes of the same machine but named pipes, sockets, and RPC are for processes being run on different machines.

Having multiple threads of a process to execute different parts of a computational work in parallel is very similar to having multiple processes in one computer to execute the same work in parallel. For example, as shown in Fig. 3(b), Process 3 has two threads running at the same time. Both thread 1 and thread 2 have their own stacks. They can access the same variables resources in Process 3. The two threads can carry out different parts of work in parallel. Similar to the multi-process IPC, the multi-thread IPC deals with data exchange and sharing between different threads of the same process, instead of different processes. However, communication latency of multi-thread IPC is usually less than that of multi-process IPC.

This work employs the multi-thread IPC technology to implement the message passing mechanism in S4 because it consumes less system resources by sharing resources among threads of the same process and has lower communication latency. Two threads are designed in S4 to execute the Simulator in one thread and the Controller in another. Both the Controller and the Simulator can access variables in the same memory space of the process. The Controller can suspend and resume the Simulator during the simulation. More detailed discussions on the implementation of S4,



(a) Communication between different processes



(b) Communication between different threads in a single process

Fig. 3. Inter-process communication: (a) communication between different processes; (b) communication between different threads in a single process.

especially on the multi-thread IPC implementation, are provided in the next section.

4. System implementation

As discussed in the previous section, Fortran 90/95 is employed to implement the Simulator and the SWMM Adaptor of S4 and C++ is used to implement the Controller and the Visualizer of S4. The corresponding development tools employed for Fortran 90/95 and C++ are CVF 6.5 and BCB 6, respectively.

One of the Fortran 90/95 features, called *modules*, is used to accomplish the implementation of the SWMM Adaptor and achieve encapsulation of both data and methods. Fig. 4 shows the Unified Modeling Language (UML) notation of the SWMM Adaptor. The SWMM Adaptor has only one object, called LXX_JUN, which uses the Junction_info object to manage many attributes representing the state variables in the SWMMs EXTRAN block, and some methods (e.g., Init()) to operate on the data. For example, TIME and Y(:) are attributes representing the simulation time and the water depth of a junction at a given computation step, respectively. Init() and Final() are subroutines that dynamically allocate the memory spaces for data arrays (such as Y(:), Z(:), etc.) when the program starts and release the memory when the program stops, respectively. SetJuncDynInfo() and GetJuncDynInfo() are subroutines for the Simulator to store the computed results into the SWMM Adaptor and for the Controller to retrieve the results from the SWMM Adaptor, respectively. The modified source codes of SWMM and the SWMM Adaptor are compiled and linked by CVF into a single library which is then used and invoked by the Controller.

Fig. 5 schematically depicts the implementation details of the S4 system. The system includes a dynamic-link library (DLL) that binds together the modified SWMM code, the SWMM Adaptor, and the Controller. A DLL is a set of procedures that are compiled, linked, and stored separately from the calling processes (Solomon and Russinovich, 2000). Unlike a static-link library (.lib file), the object code of the procedures is not part of the object

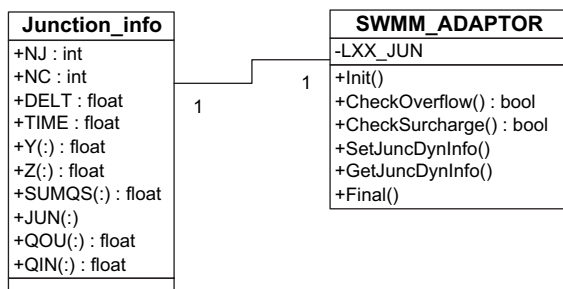


Fig. 4. Design of the SWMM Adaptor.

code of the calling processes. It is dynamically invoked by the calling processes and loaded into the processes' memory space during runtime. The DLL and the process that invokes the procedures of the DLL share the same virtual memory space and all threads in the process can call these DLL procedures. However, the procedures of a DLL are not visible by the procedures outside the DLL. They must be exported from the DLL and imported into the calling procedures before using them. In CVF, subroutines in a DLL can be exported by using the compiler directives, !DEC\$ ATTRIBUTES DLLEXPORT (Etzel and Dickinson, 1999), for example:

```

Subroutine TRANSX(time)
!DEC$ ATTRIBUTES DLLEXPORT :: TRANSX
  
```

The directive statement is used to force the CVF compiler to export the subroutine TRANSX as the symbol TRANSX to a corresponding import library, which is like a static library but contains no executive code and only includes the symbols exported from the corresponding DLL (Petzold, 1999; Richter, 1999).

Because the Controller is coded in C++ but the SWMM Adaptor is coded in Fortran 90/95, calling the subroutines of the SWMM Adaptor from the Controller involves mixed-language programming. The subroutines of the SWMM Adaptor, e.g. subroutine GetJuncDynInfo(), must be declared in the code of the Controller using the extern "C" statement as follows:

```

extern "C" void _stdcall
GETJUNCODYNINFO(float *time, float *y, float
*sumqs, int *n);
  
```

The extern "C" linkage specifier indicates that the subroutine is not implemented in C++ but conforms to the convention of C implementation. The __stdcall keyword asks the C++ compiler to employ the calling convention of Fortran for argument passing when the subroutine is called (Etzel and Dickinson, 1999). All subroutines invoked by the Controller are declared in swmm.h file. The Controller invokes the subroutines in the DLL through the DLLs corresponding import library. However, the import library generated by CVF is in the COFF format and cannot be used in BCB because it requires the import library to be in the OMF format. Fortunately, BCB provides the utility, called implib.exe, to generate an import library in the OMF format from the source DLL in the COFF format with the following command:

```
implib target-import-lib source-dll
```

Fig. 6 illustrates the application of the multi-thread IPC technology in S4. There are three components (i.e. the Controller, the Simulator, and the SWMM Adaptor)

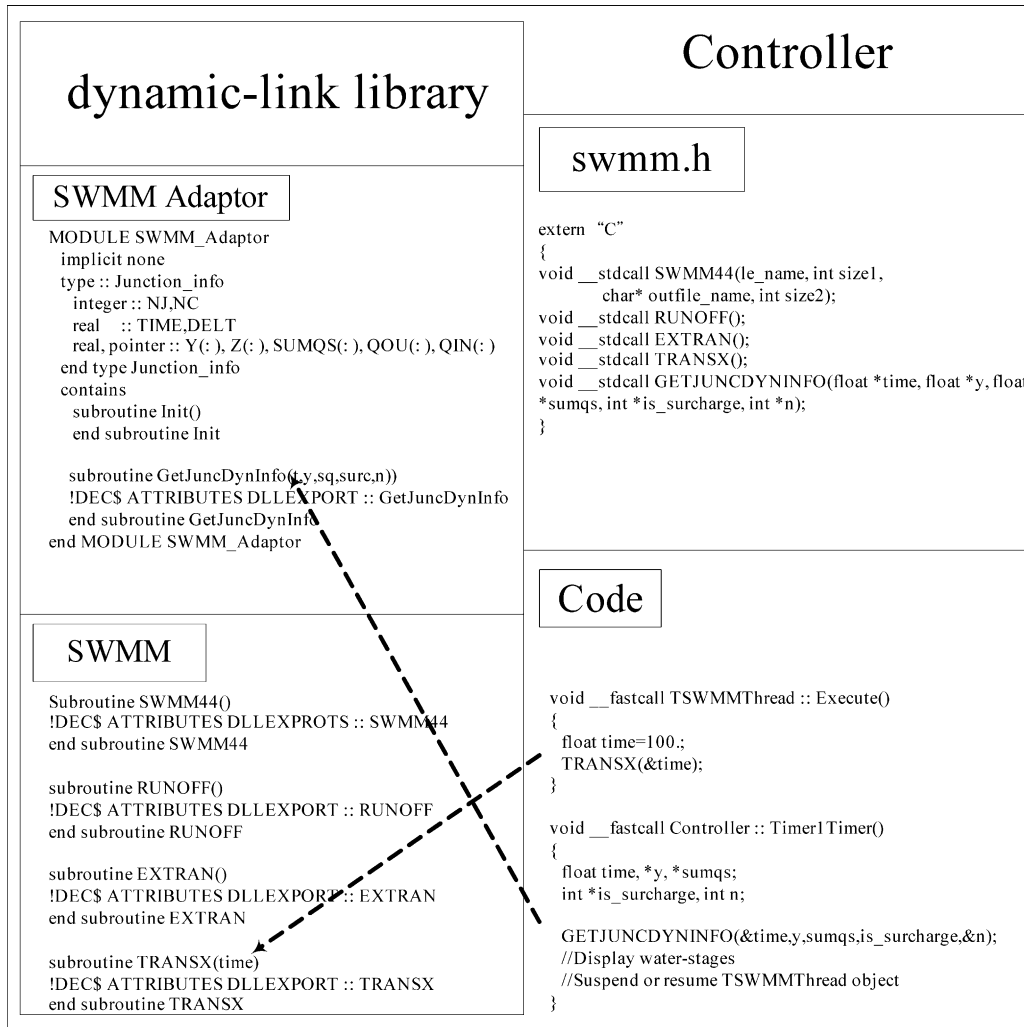


Fig. 5. Implementation of the S4 system.

running on two threads (i.e. the Controller thread and the Simulator thread). The Controller thread is generated automatically by the operating system (OS) when S4 starts to run. When the OS receives the user's request, it sends a WM_TIME message with the request to the Controller. The Controller then takes the actions in response to the user's request. For example, after receiving the user's request to simulate the storm sewer flow, the Controller constructs a new thread to run the Simulator. It is convenient to use the TThread class (Holling et al., 2003) provided by BCB to implement the TSWMMThread class, which is used to instantiate the thread object to execute the Simulator. The major methods of the TSWMMThread class is shown below:

```

class TSWMMThread: public TThread
{
private:
protected:
void __fastcall Execute();
                
```

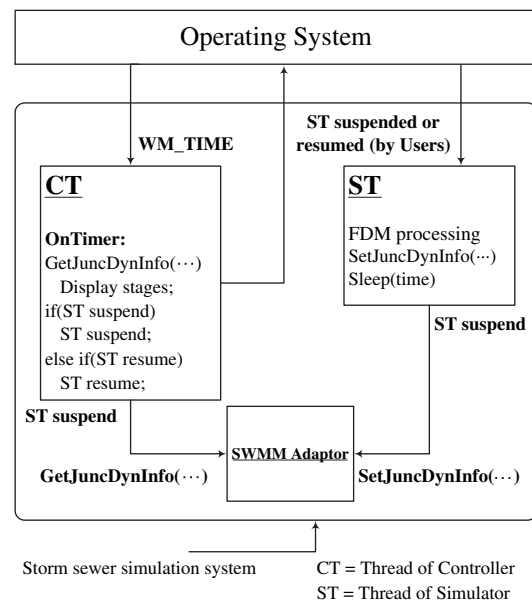


Fig. 6. Thread communication concept in the S4 system.

```
public:
    __fastcall TSWMMThread(bool CreateSuspended);
};
```

The TSWMMThread class is inherited from the TThread class. The Execute() method is called when the thread is running. Fig. 7 exhibits the implementation details of the thread communication and subroutine calls to the DLL in S4. Once the user clicks on the start button, the Controller generates a new TSWMMThread object and the Execute() method of the object is called automatically. The Transx() subroutine in the DLL is then called by the Execute() method to start the execution of the Simulator. At this point, both the Controller thread and the Simulator thread are running.

When the Simulator finishes the FDM processing for a computation step, it sets the computed results into the SWMM Adaptor and suspends the execution of its own thread (i.e. the Simulator thread) for a specified time by calling the Sleep(time) subroutine. The Controller thread then takes the control of the execution of S4 and waits for the user's request.

5. Application examples

Two examples are used to verify and demonstrate the application of the S4 system developed using the proposed integration approach. The first example uses a real but small drainage system, named the Liu-Guan

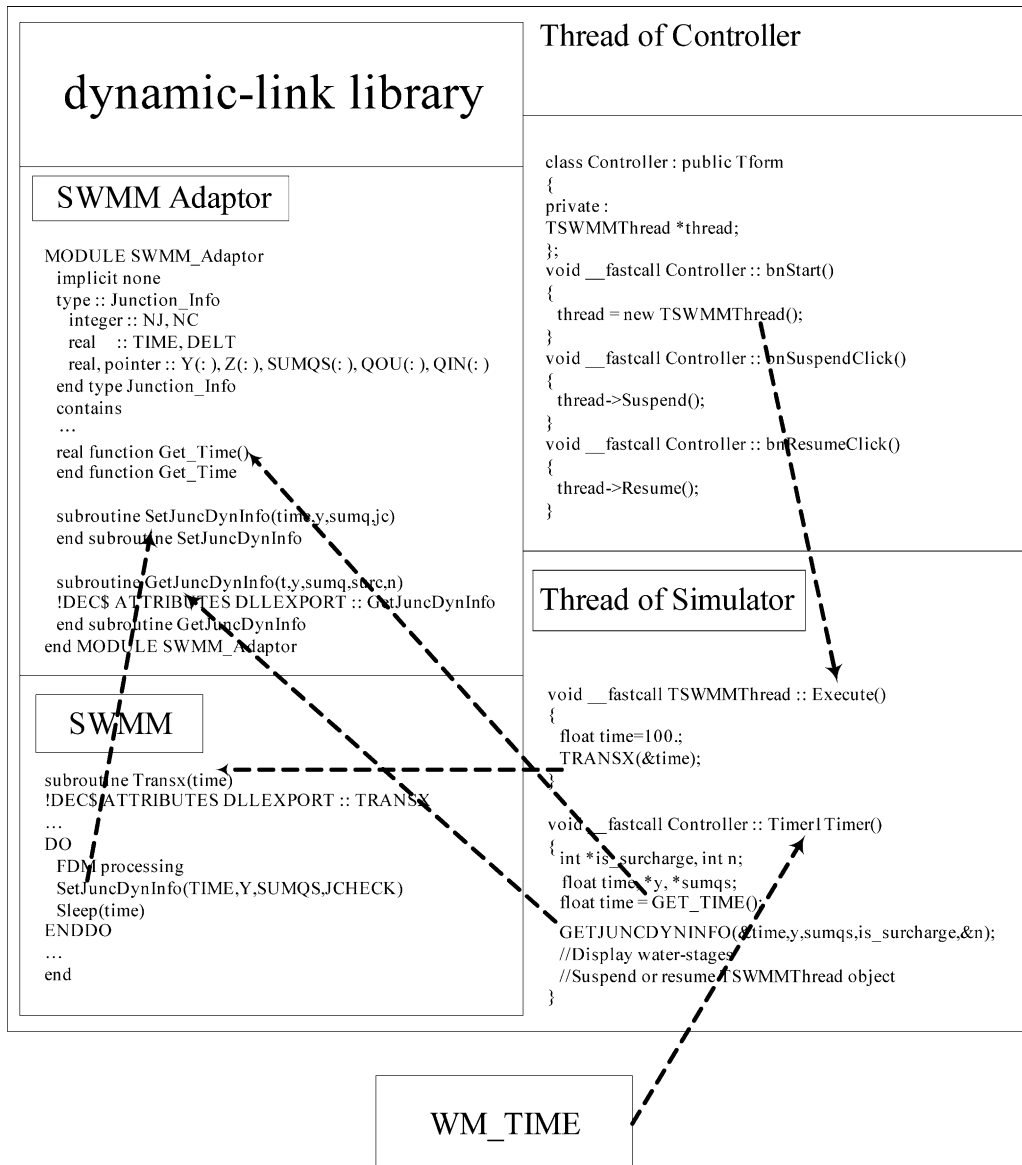


Fig. 7. Implementation of thread communication in the S4 system.

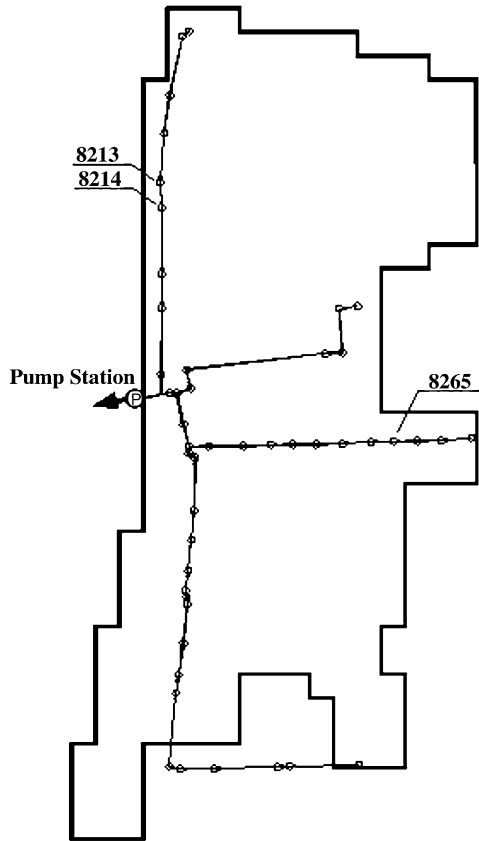


Fig. 8. The layout of the Liu-Guan drainage system.

system, which is located in the eastern part of the Taipei City. The capability of the S4 system to suspend/resume the execution of simulation and allow user interactions is demonstrated in this example. The second example uses an idealized drainage system to demonstrate the

capability of S4 to handle the real-time input of the runoff information.

5.1. The Liu-Guan drainage system

The layout of the Liu-Guan drainage system is shown in Fig. 8. The system contains 45 manholes (or junctions) and one pumping station with a maximum pumping rate at 20.0 cms. An 8-h design rainfall event is calculated first by SWMM-RUNOFF to produce the runoff hydrograph, which is then taken as the inflow hydrograph into the manholes for the sewer flow simulation performed by SWMM-EXTRAN in S4. The water-stages hydrograph of junctions computed by SWMM-EXTRAN can be displayed in real-time during the simulation in S4. The user is also allowed to start, suspend, and resume the storm sewer simulation simply by clicking on the Start, Suspend, and Resume buttons, respectively, provided in the graphical user interface of S4. Fig. 9 shows the water-stage hydrograph at junctions 8213 and 8214 when the simulation has been carried out for 23, 900 s. There are check boxes on the right hand side of the system for the user to toggle on and off the display of the water-stage hydrograph at each junction in the drainage system.

5.2. An idealized drainage system

Fig. 10 shows an idealized drainage system. Each conduit in the system is a circular pipe of 1.5 m in diameter and the depth at each junction (or manhole) is 6 m. The junction 82 309 separates the system into two slopes, 1/500 from junction 80 408 to 82 309 and 1/750 from junction 82 309 to 16 109). Two simulations are

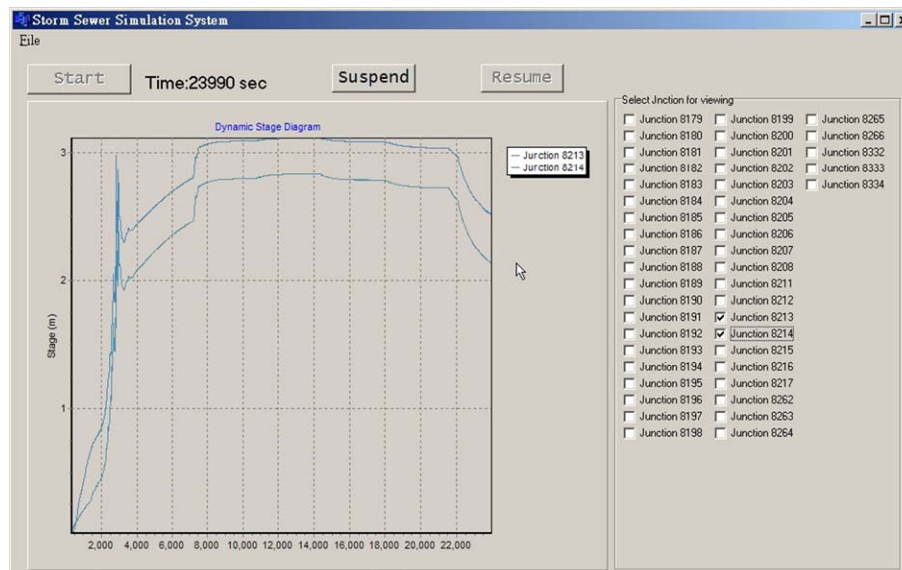


Fig. 9. Water-stage hydrograph at junction 8213 and 8214 when the simulation time is at 23, 990 s.

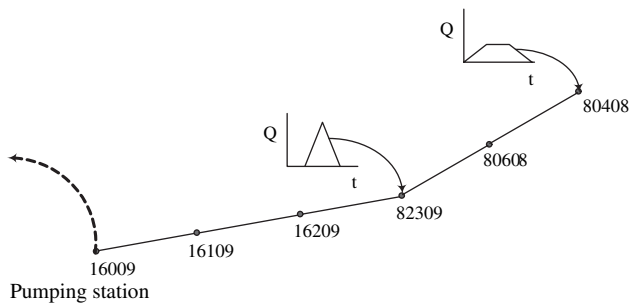


Fig. 10. The idealized drainage system.

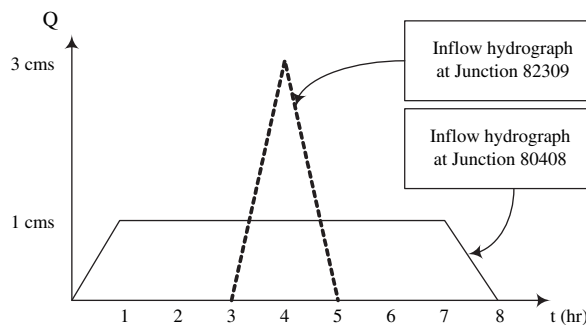


Fig. 11. Inflow hydrographs at junctions 80408 and 82309.

conducted by S4 for comparison. Both of the simulations use the input file containing a given inflow hydrograph at junction 80408, as indicated by the solid line in Fig. 11. The second one assumes that a storage tank located at junction 82309 is suddenly broken at the time of 3 h and the inflow hydrograph at junction 82309, as indicated by the dash line in Fig. 11, is used as a real-time input. For the second simulation, the S4 system suspends the simulation at the time of 3 h and reads the inflow information at junction 82309 before it resumes the simulation. The computed water-stage hydrographs at junctions 80608, 82309, and 16209 without and with the real-time input at junction 82309 are shown in Figs. 12 and 13, respectively. It can be seen that S4 is capable of taking real-time inflow data for sewer flow simulation.

6. Conclusions

This paper has presented an integration approach that employs the multi-thread technology to integrate the

legacy SWMM model into a storm sewer simulation system, named S4, developed in this work. The S4 system takes advantage of the SWMM-EXTRAN block to carry out the sewer flow simulation and allows for display of the computed water-stage hydrograph at the junctions immediately after the results are computed. It is also capable of suspending and resuming the simulation at the end of any time step of the explicit-FDM process to account for real-time inflow information for sewer flow simulation.

The proposed multi-thread approach for system integration should be generally applicable to integrated modeling for water resources systems consisting of legacy models as long as code modification on the legacy models is permitted. This approach gives better efficiency of the integrated system when compared with the approach that integrates legacy components through their input/output files. When compared with the multi-process approach, this approach consumes less system resources and has lower IPC latency. In addition, the proposed approach can be easily extended to address the

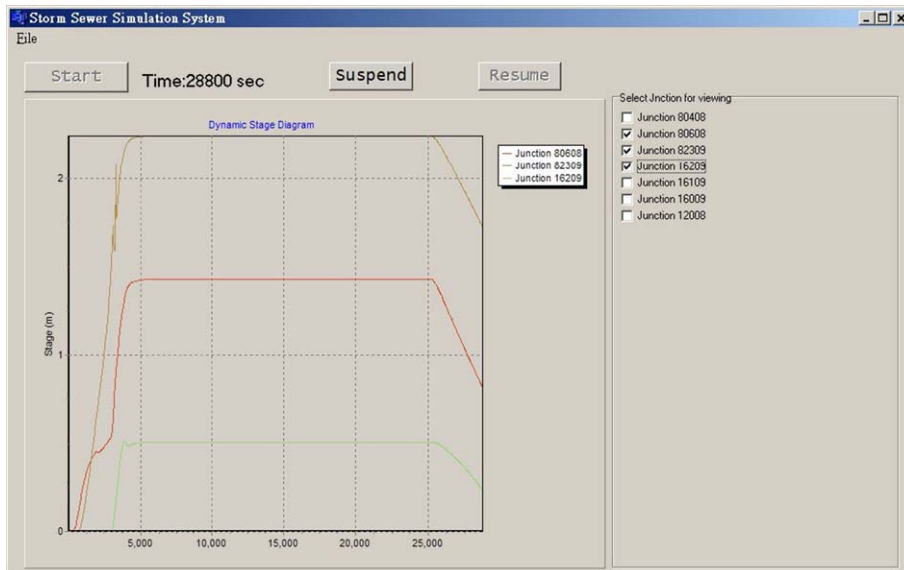


Fig. 12. Water-stages at junctions without real-time inflow.

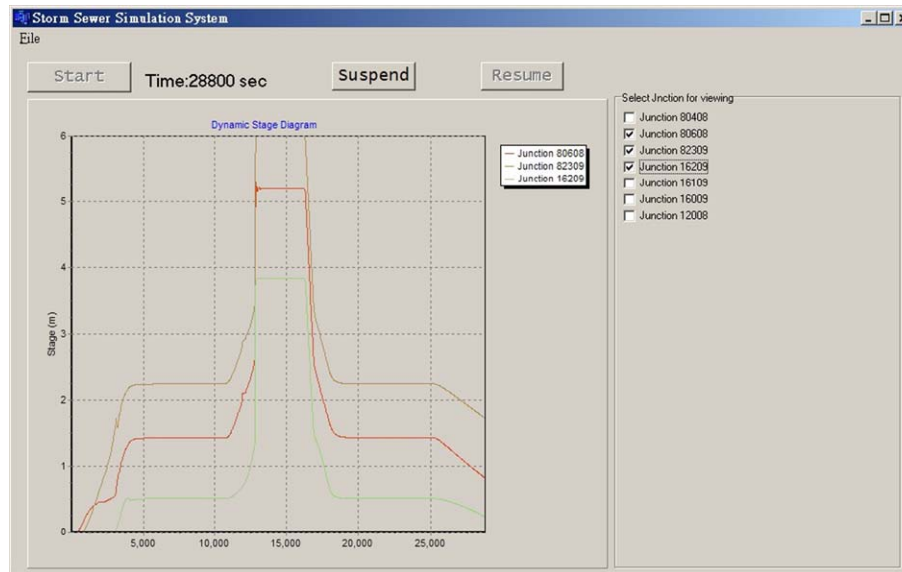


Fig. 13. Water-stages at junctions with real-time inflow.

real-time and/or optimal control simulation problems in storm sewer systems (Pleau et al., 2005) or wastewater systems (Beck, 2005; Butler and Schütze, 2005; Vanrolleghem et al., 2005). For example, one more thread can be added into S4 to calculate the optimal operation strategy using some optimization software. Periodically the real-time simulation in S4 can be suspended for computation of the optimal operation strategy before it is resumed with the updated operation strategy.

Acknowledgements

The authors would like to acknowledge the financial support from the Construction and Planning Agency in Minister of Interior Affairs, and the National Science Council (under Grant No. NSC 92-2211-E-002-044).

References

- Achee, B.L., Carver, D.L., 1997. Creating object-oriented designs from legacy Fortran code. *Journal of System Software* 39, 179–194.
- Beck, M.B., 2005. Vulnerability of water quality in intensively developing urban watersheds. *Environmental Modelling and Software* 20, 381–400.
- Butler, D., Schütze, M., 2005. Integrating simulation models with a view to optimal control of urban wastewater systems. *Environmental Modelling and Software* 20, 415–426.
- Campbell, C.W., Sullivan, S.M., 2002. Simulating time-varying cave flow and water levels using the storm water management model. *Engineering Geology* 65, 133–139.
- Chaudhry, M.H., 1993. *Open-Channel Flow*. Prentice-Hall, Englewood Cliffs, NJ.
- Etzel, M., Dickinson, K., 1999. *Digital Visual Fortran Programmer's Guide*. Digital Press, MA.
- Ewer, J., Knight, B., Cowell, D., 1995. Case study: an incremental approach to re-engineering a legacy Fortran computational fluid dynamics code in C++. *Advances in Engineering Software* 22, 153–168.
- Guo, J., 2003. Software reuse through re-engineering the legacy systems. *Information and Software Technology* 45, 597–609.
- Holling, J., Swart, B., Cashman, M., Gustavson, P., 2003. *Borland C++ Builder 6 Developer's Guide*. Sams, Indiana.
- Hsu, M.H., Chen, S.H., Chang, T.J., 2000. Inundation simulation for urban drainage basin with storm sewer system. *Journal of Hydrology* 234 (1–2), 21–37.
- Huber, W.C., Dickinson, R.E., 1988. *Storm Water Management Model. User's Manual Ver. IV*. U.S. Environmental Protection Agency.
- Ji, Z., 1998. General hydrodynamic model for sewer/channel network systems. *Journal of Hydraulic Engineering* 123 (3), 307–315.
- Lin, S.S., Chang, H.K., Hsieh, S.H., Kuo, J.T., Lai, J.S., 2003. An integrated approach for inundation simulation in an urban area. In: *International Conference of GIS and Remote Sensing in Hydrology, Water Resources and Environment*. Three Gorges Dam Site, China, September 16–19, p. 143 (7 pages, CD-ROM).
- Liong, S.Y., Chan, W.T., ShreeRam, J., 1995. Peak-flow forecasting with genetic algorithm and SWMM. *Journal of Hydraulic Engineering* 121 (8), 613–617.
- Lutz, M., 2001. *Programming Python 2/e*. O'Reilly, CA.
- Lutz, R.R., 1995. Distributed vertical model integration. *Johns Hopkins APL Technical Digest* 16 (2), 187–196.
- Neil, P.G., Sherlock, R.A., Bright, K.P., 1999. Integration of legacy sub-system components into an object-oriented simulation model of a complete pastoral dairy farm. *Environmental Modelling and Software* 14, 495–502.
- Park, H., Johnson, T.J., 1998. Hydrodynamic modeling in solving combined sewer problems: a case study. *Water Research* 32 (6), 1948–1956.
- Petzold, C., 1999. *Programming Windows 5/e*. Microsoft Press, Washington.
- Pleau, M., Colas, H., Lavallée, P., Pelletier, G., Bonin, R., 2005. Global optimal real-time control of the Quebec urban drainage system. *Environmental Modelling and Software* 20, 401–413.
- Rauch, W., Harremoës, P., 1996. Minimizing acute river pollution from urban drainage systems by means of integrated real time control. In: *First International Conference on New/Emerging Concepts for Rivers*. Chicago, September 1996.

- Rauch, W., Harremoës, P., 1999. Genetic algorithms in real time control applied to minimize transient pollution from urban wastewater systems. *Water Research* 33 (5), 1265–1277.
- Rauch, W., Bertrand-Krajewski, J.-L., Krebs, P., Mark, O., Schilling, W., Schütze, M., Vanrolleghem, P.A., 2002. Deterministic modeling of integrated urban drainage systems. *Water Science and Technology* 45 (3), 81–94.
- Riche, R.L., Gaudin, J., Besson, J., 2003. An object-oriented simulation–optimization interface. *Computers and Structures* 81, 1689–1701.
- Richter, J., 1999. *Programming Applications for Microsoft Windows 4/e*. Microsoft Press, RW.
- Sang, J., Follen, G., Kim, C., Lopez, I., 2002. Development of CORBA-based engineering application from legacy Fortran programs. *Information and Software Technology* 44, 175–184.
- Schütze, M., Butler, D., Beck, B., 1996. Development of a framework for the optimization of runoff, treatment and receiving waters. In: *Seventh International Conference of Urban Storm Drainage*. Hannover, 9–13 September 1996.
- Schütze, M., Butler, D., Beck, B., 1999. Optimization of control strategies for the urban wastewater system – an integrated approach. *Water Science and Technology* 39 (9), 209–216.
- Solomon, D.A., Russinovich, M.E., 2000. *Inside Microsoft Windows 2000 3/e*. Microsoft Press, Redmond, Washington.
- Stroustrup, B., 1997. *The C++ Programming Language 3/e*. Addison-Wesley, Massachusetts.
- Tanenbaum, A.S., 2001. *Modern Operating System 2/e*. Prentice-Hall, NJ.
- Tolsma, J., Barton, P.I., 2000. DAEPACK: an open modeling environment for legacy models. *Industrial and Engineering Chemistry Research* 39, 1826–1839.
- Vanrolleghem, P.A., Benedetti, L., Meirlaen, J., 2005. Modelling and real-time control of the integrated urban wastewater system. *Environmental Modelling and Software* 20, 427–442.
- Zaghloul, N.A., 1997. Unsteady gradually varied flow in circular pipes with variable roughness. *Advances in Engineering Software* 28, 115–131.