

# REPORT

Cicerstr. 24  
D-10709 Berlin  
Germany  
Tel +49 (0)30 536 53 800  
Fax +49 (0)30 536 53 888  
[www.kompetenz-wasser.de](http://www.kompetenz-wasser.de)

## Applicability of OpenMI and API for coupling models within MIA-CSO Project acronym: SAM-CSO

by  
Hauke Sonnenberg  
Kompetenzzentrum Wasser Berlin gGmbH

for  
Kompetenzzentrum Wasser Berlin gGmbH

Preparation of this report was financed in part through funds provided by



Berlin, Germany

2009

## **Important Legal Notice**

**Disclaimer:** The information in this publication was considered technically sound by the consensus of persons engaged in the development and approval of the document at the time it was developed. KWB disclaims liability to the full extent for any personal injury, property, or other damages of any nature whatsoever, whether special, indirect, consequential, or compensatory, directly or indirectly resulting from the publication, use of application, or reliance on this document. KWB disclaims and makes no guaranty or warranty, expressed or implied, as to the accuracy or completeness of any information published herein. It is expressly pointed out that the information and results given in this publication may be out of date due to subsequent modifications. In addition, KWB disclaims and makes no warranty that the information in this document will fulfill any of your particular purposes or needs. The disclaimer on hand neither seeks to restrict nor to exclude KWB's liability against all relevant national statutory provisions.

## **Wichtiger rechtlicher Hinweis**

**Haftungsausschluss:** Die in dieser Publikation bereitgestellte Information wurde zum Zeitpunkt der Erstellung im Konsens mit den bei Entwicklung und Anfertigung des Dokumentes beteiligten Personen als technisch einwandfrei befunden. KWB schließt vollumfänglich die Haftung für jegliche Personen-, Sach- oder sonstige Schäden aus, ungeachtet ob diese speziell, indirekt, nachfolgend oder kompensatorisch, mittelbar oder unmittelbar sind oder direkt oder indirekt von dieser Publikation, einer Anwendung oder dem Vertrauen in dieses Dokument herrühren. KWB übernimmt keine Garantie und macht keine Zusicherungen ausdrücklicher oder stillschweigender Art bezüglich der Richtigkeit oder Vollständigkeit jeglicher Information hierin. Es wird ausdrücklich darauf hingewiesen, dass die in der Publikation gegebenen Informationen und Ergebnisse aufgrund nachfolgender Änderungen nicht mehr aktuell sein können. Weiterhin lehnt KWB die Haftung ab und übernimmt keine Garantie, dass die in diesem Dokument enthaltenen Informationen der Erfüllung Ihrer besonderen Zwecke oder Ansprüche dienlich sind. Mit der vorliegenden Haftungsausschlussklausel wird weder bezweckt, die Haftung der KWB entgegen den einschlägigen nationalen Rechtsvorschriften einzuschränken noch sie in Fällen auszuschließen, in denen ein Ausschluss nach diesen Rechtsvorschriften nicht möglich ist.

## **Colophon**

**Title**

Applicability of OpenMI and API for coupling models within MIA-CSO

**Authors**

Hauke Sonnenberg, Kompetenzzentrum Wasser Berlin gGmbH

**Quality Assurance**

Kai Schroeder, Kompetenzzentrum Wasser Berlin gGmbH

Benoît Béraud, Veolia Environnement Recherche & Innovation

**Publication / Dissemination approved by technical committee members:**

Christelle Pagotto, Veolia Eau

Cyrille Lemoine, Veolia Environnement Recherche & Innovation

Matthias Rehfeld-Klein, Senatsverwaltung für Gesundheit Umwelt und Verbraucherschutz Berlin

Dörthe von Seggern, Senatsverwaltung für Gesundheit Umwelt und Verbraucherschutz Berlin

Regina Gnirß, Berliner Wasserbetriebe

Kay Joswig, Berliner Wasserbetriebe

Erika Pawlowsky-Reusing, Berliner Wasserbetriebe

Nicolas Ramponoux, Veolia Environnement Recherche & Innovation

Emmanuel Soyeux, Veolia Environnement Recherche & Innovation

Yann Moreau-Le Golvan, Kompetenzzentrum Wasser Berlin gGmbH

**Deliverable number**

D 5

## **Abstract (English)**

The overall objective of MIA-CSO is to develop a model-based planning instrument for impact based CSO control. The objective of this study was to examine the potential and the drawbacks of different model coupling techniques that may be taken into account within the MIA-CSO project.

- The models InfoWorks CS (Wallingford Software Ltd.) and HYDRAX/QSim (German Federal Institute of Hydrology, BfG) that are intended to be used within MIA-CSO as well as two different techniques for model coupling (Open Modelling Interface – OpenMI and Application Programming Interfaces – API) are described.
- It is presented how impacts of combined sewer overflows (CSO) on receiving waters have been simulated in Berlin, so far.
- According to the objectives of MIA-CSO (statistical analysis, uncertainty and sensitivity analysis), this report describes in the form of two possible scenarios, how the intended integrated modelling system could be realized:
  - scenario one is based on the application of the OpenMI,
  - scenario two is based on the application of model-specific APIs.
- Practical tests are reported, in which simple scenarios, consisting of available OpenMI components, have been set-up and carried out by use of the OpenMI Configuration Editor. Potential and drawbacks of the OpenMI implementation in InfoWorks CS are described.
- A wrapping procedure is described that helps making existing models comply with the OpenMI standard. Assumptions are made about the effort involved.
- Experiences with the InfoWorks Type Library, which defines the InfoWorks API, are reported and the available methods and properties are documented. It is explained how specific tasks can be handled by means of API calls.

General Results and conclusions are:

- Concerning model coupling techniques, sequential linking, where models run one after the other can be distinguished from parallel linking where models advance simultaneously in time and where models may exchange data at each timestep. In the current modelling practice applied in Berlin, models run sequentially. So far, simulation, data export and import as well as data transformation are realized manually. By automating these processes, the modelling procedure can be improved in terms of efficiency, error-proneness and accuracy.

Results and conclusions concerning the OpenMI:

- The OpenMI is a generic interface specification that standardizes the parallel linking approach. It defines the manner in which models are run and data are exchanged between models. The process of making existing models compliant to the OpenMI standard (migration) can be seen as teaching the models to ‘speak the same language’.

- Migration of a model does not per se guarantee that the model can be used in any OpenMI scenario. The useability of OpenMI components strongly depends on
  - how models implement the OpenMI and
  - which exchange items and additional data operations they offer.
- Migration requires the model's source code to be available. The migration process is supported by software tools that are provided by the OpenMI Association. The effort needed to migrate an existing modelling software into a corresponding OpenMI-compliant component depends on the tasks to be performed by the OpenMI component, personal preconditions and technical preconditions. It strongly depends on the structure of the existing model engine.
- Of the model softwares used to date within MIA-CSO, only InfoWorks CS supports the OpenMI. The current OpenMI implementation in InfoWorks only allows for flows and water levels as input quantities.
- If OpenMI is implemented or not, will depend on the incentive for model developers and on their ability in doing the migration. At least within near future, it is improbable that the models Hydrax and QSim that are intended to be used within MIA-CSO will be migrated to the OpenMI standard.
- Further developments at KWB will therefore concentrate on additional tools for the MIA-CSO planning instrument, which allow for e.g. statistical analysis, automated calibration, uncertainty and sensitivity analysis. By doing so, the OpenMI approach can be followed, which will make own developments also attractive to others.

Results and conclusions concerning API:

- Through Application Programming Interfaces (API), a software allows a selection of its internal functions to be called from other softwares. An API is specific to the software that it is provided by. An API often allows for automated access to those functions of a software that normally are invoked by a Graphical User Interface.
- Compared to the OpenMI scenario, in the API scenario models are processed sequentially. However, model preparation, data import, start of simulation and data export are automated using the provided API functionality.
- The InfoWorks API documentation provided by Wallingford Software Ltd. is incomplete. This is problematic if InfoWorks shall be accessed through its API.
- This report describes step-by-step how InfoWorks CS can be accessed through its API from within the programming languages VBA and C#, respectively.
- The InfoWorks API allows for importing model data and input data from files. Changing of model parameters can be realized indirectly by a sequence of data export, data manipulation and data import. The same holds true for e.g. rainfall series data. Only complete simulations can be invoked. Being limited to properties of links (e.g. conduits, pumps, weirs), simulation results can be exported to files.
- The possibility to change internal model parameters through the InfoWorks API allows for applying this interface for e.g. statistical analysis, automated calibration, uncertainty and sensitivity analysis.

## Abstract (French)

L'objectif premier du projet MIA-CSO est d'élaborer un outil de planification basé sur un modèle afin de prédire l'impact des rejets des déversoirs d'orage par temps de pluie (CSO – Combined sewer overflows) sur la qualité des eaux de la Spree, la rivière berlinoise. Cette étude se propose d'évaluer les possibilités et les limites de différentes techniques de couplage de modèles susceptibles d'être employées dans le cadre de MIA-CSO.

- Le rapport décrit les modèles InfoWorks CS (Wallingford Software) et HYDRAX/QSim (Institut fédéral allemand d'hydrologie, BfG) dont la mise en œuvre est envisagée pour MIA-CSO, ainsi que deux techniques de couplage de modèles (OpenMI – Open Modelling Interface, interfaces de programmation API – Application Programming Interfaces).
- L'auteur présente les méthodes utilisées jusqu'ici par la ville de Berlin pour simuler l'impact sur les eaux réceptrices des rejets des CSO.
- Conformément aux autres objectifs du projet MIA-CSO (analyse statistique des résultats de simulation, analyse des incertitudes et de la sensibilité de la modélisation), ce rapport présente deux scénarios potentiels pour la mise au point du futur système de modélisation intégrée :
  - le premier scénario repose sur l'interface standard OpenMI,
  - le deuxième s'appuie sur l'utilisation d'interfaces de programmation (API) propres à chaque application de modélisation.
- Le rapport rend compte des tests pratiques : des scénarios simples basés sur des composants OpenMI disponibles ont été construits puis exécutés de manière intégrée à l'aide de l'éditeur de configuration OpenMI (OpenMI Configuration Editor). Les possibilités et les limites de l'implémentation d'OpenMI dans InfoWorks CS sont passées en revue.
- Une procédure d'empaquetage (wrapping), destinée à rendre les modèles existants compatibles avec le standard OpenMI, est présentée. L'effort que représente le wrapping est évalué.
- Le rapport décrit les expériences réalisées avec la bibliothèque de types InfoWorks, dans laquelle est définie l'interface de programmation (API) d'InfoWorks, et il documente les méthodes et propriétés mises à disposition par l'API. Il explique comment certaines tâches spécifiques peuvent être gérées par des appels d'API.

Résultats et conclusions d'ensemble :

- Techniques de couplage de modèles : on distingue globalement le couplage séquentiel, dans lequel les modèles s'exécutent l'un après l'autre, du couplage parallèle où les modèles progressent de manière simultanée et peuvent échanger des données à tout moment. L'approche actuellement en vigueur à Berlin s'appuie sur une modélisation séquentielle. Les opérations de simulation, d'exportation, d'importation ou de transformation des données sont réalisées manuellement. Leur automatisation permettrait d'améliorer l'efficience et la précision, et de réduire la probabilité d'erreur de la procédure de modélisation.

## Résultats et conclusions concernant OpenMI :

- OpenMI est une spécification d'interface générique qui standardise l'approche de couplage parallèle. Ce standard définit la façon dont les modèles s'exécutent et dont les données sont échangées entre modèles. Le processus de migration permet de rendre les modèles existants compatibles avec le standard OpenMI : on leur apprend ainsi à « parler le même langage ».
- La migration en soi ne garantit pas que le modèle en question puisse être utilisé dans n'importe quel scénario OpenMI. L'applicabilité des composants OpenMI dépend aussi sensiblement :
  - de la manière dont les modèles implémentent l'interface OpenMI, et
  - du type d'éléments d'échange (exchange items) qu'ils proposent – quelles variables du modèle (débit, hauteur d'eau par exemple) peuvent être échangées contre des éléments du modèle (regards, canaux, par exemple), et | (exemple : agrégation spatiale).
- La migration d'un logiciel de modélisation existant suppose que le code source soit disponible. Le processus de migration est soutenu par des outils logiciels fournis par l'Association OpenMI. L'effort nécessaire pour transformer un logiciel de modélisation existant en un composant compatible avec OpenMI dépend des tâches imparties à ce composant OpenMI, des difficultés rencontrées par le développeur et des pré-requis techniques. De plus, cet effort dépend largement de la structure du noyau de modélisation existant.
- Parmi les logiciels de modélisation utilisés jusqu'ici dans le cadre du projet MIA-CSO, InfoWorks CS est le seul à offrir OpenMI. Toutefois, l'implémentation actuelle d'InfoWorks ne propose comme variables d'entrée que les débits et les hauteurs d'eau.
- La décision de mettre en œuvre OpenMI ou d'y renoncer dépendra notamment de l'attrait de cette interface pour les développeurs des modèles, et de leur aptitude à mener à bien la migration. À brève échéance, il est peu probable que les modèles Hydrax et QSim retenus pour le projet MIA-CSO fassent l'objet d'une migration vers OpenMI.
- Ainsi, dans un proche avenir, le Centre de compétence sur l'eau de Berlin (KWB) concentrera ses efforts sur la mise au point de modules complémentaires pour l'outil de planification de MIA-CSO : analyse statistique, calage automatique, analyse de sensibilité et d'incertitudes, par exemple. Les efforts fournis afin d'appliquer l'approche OpenMI et l'expérience acquise pourraient être intéressants à plus long terme pour d'autres utilisateurs.

## Résultats et conclusions concernant les API :

- Par le biais d'interfaces de programmation (API), un logiciel permet à d'autres logiciels d'appeler certaines de ses fonctions internes. Une API est propre au logiciel avec lequel elle est fournie. Une API permet souvent d'accéder automatiquement aux fonctions d'un logiciel normalement appelées par une interface utilisateur graphique (GUI).
- Contrairement à ce qui se passe dans le scénario basé sur OpenMI, les modèles qui utilisent des API sont traités de façon séquentielle. Toutefois, la préparation du modèle, l'importation des données, le lancement de la simulation et

l'exportation des données s'effectuent automatiquement via des appels de fonctions API.

- La documentation sur l'application InfoWorks API fournie par Wallingford Software est incomplète. Cela pose problème pour accéder à InfoWorks via sa propre API.
- Ce rapport décrit étape par étape la manière d'accéder aux composantes du logiciel InfoWorks CS via son API en utilisant respectivement les langages de programmation VBA et C#.
- L'interface API d'InfoWorks permet d'importer les données décrivant un modèle et ses données d'entrée à partir de fichiers. Les paramètres du modèle peuvent être modifiés de manière indirecte par une séquence d'exportation des données, de manipulation des données exportées et d'importation de données. Ceci vaut également pour les données sur les séries pluviométriques stockées dans le modèle, par exemple. Seules des simulations complètes peuvent être appelées pour une période de simulation donnée via l'API. Les résultats des simulations permettent d'exporter dans des fichiers uniquement les propriétés au niveau des jonctions entre deux éléments du réseau tels que pompes, déversoirs; quant aux propriétés des nœuds (nodes), elles ne peuvent être exportées.

La possibilité de modifier les paramètres internes du modèle par le biais de l'interface API d'InfoWorks permet par exemple d'exploiter cette interface pour l'analyse statistique, le calage automatique, ainsi que l'analyse des incertitudes et de la sensibilité.

## **Abstract (German)**

Das übergeordnete Ziel des KWB-Projekts MIA-CSO ist die Entwicklung eines modellbasierten Planungsinstruments für die Vorhersage der Auswirkungen von Mischwasserüberläufen auf die Gewässerqualität in der Berliner Stadtspree. Das Ziel dieser Teilstudie war, verschiedene Techniken der Modellkopplung, die innerhalb des Projektes MIA-CSO zur Anwendung kommen können, auf ihre Möglichkeiten und Einschränkungen hin zu untersuchen.

- Die für den Einsatz in MIA-CSO vorgesehenen Modellanwendungen InfoWorks CS (Wallingford Software Ltd.) und HYDRAX/QSim (Bundesanstalt für Gewässerkunde, BfG) sowie zwei verschiedene Techniken der Modellkopplung (Open Modelling Interface, OpenMI und Application Programming Interface, API) werden beschrieben.
- Es wird dargestellt, wie die Einwirkungen von Mischwasserüberläufen auf die Gewässer in Berlin bisher simuliert wurden.
- Die zusätzlichen Ziele von MIA-CSO (statistische Auswertung der Simulationsergebnisse, Unsicherheits- und Sensitivitätsanalyse der Modellierungen) berücksichtigend beschreibt dieser Bericht anhand zweier möglicher Szenarien, wie das geplante integrierte Modellsystem realisiert werden könnte:
  - Szenario eins basiert auf der standardisierten Schnittstellentechnologie OpenMI,
  - Szenario zwei basiert auf der Verwendung von Programmierschnittstellen (API), die für die jeweiligen Modellanwendungen spezifisch sind.
- Es werden praktische Tests beschrieben, in denen verfügbare OpenMI-Modell-komponenten mit Hilfe des OpenMI Konfigurationseditors (OpenMI Configuration Editor) zu einfachen Szenarien zusammengesetzt und dann integriert ausgeführt wurden. Möglichkeiten und Einschränkungen der OpenMI-Implementierung in InfoWorks CS werden beschrieben.
- Der Vorgang des „Wrapping“ (dt. Einpacken), mit dessen Hilfe existierende Modellanwendungen in OpenMI-kompatible Modellkomponenten umgewandelt werden können, wird beschrieben. Es wird der Aufwand abgeschätzt, den das Wrapping erfordert.
- Erfahrungen mit der InfoWorks-Typbibliothek, in der die InfoWorks-Programmierschnittstelle (API) definiert ist, und die über die API verfügbar gemachten Methoden und Eigenschaften werden dokumentiert. Es wird beschrieben, wie spezielle Aufgaben der Modellierung und Simulation über entsprechende Aufrufe von API-Methoden erledigt werden können.

Allgemeine Ergebnisse und Schlussfolgerungen:

- Hinsichtlich verschiedener Modellkopplungstechniken kann allgemein zwischen sequentieller Kopplung und paralleler Kopplung unterschieden werden. Bei der sequentiellen Kopplung werden die Modelle nacheinander ausgeführt. Bei der parallelen Kopplung hingegen schreiten die Modelle gleichzeitig in der Zeit fort, und die Modelle können zu jedem Zeitpunkt Daten miteinander austauschen. In

der zurzeit für die Berliner Situation angewendeten Modellierungspraxis werden die Modelle sequentiell ausgeführt. Sowohl Simulationsstart, Datenexport und -import als auch erforderliche Datenumformungen werden manuell durchgeführt. Indem diese Prozesse automatisiert werden, kann die derzeitige Praxis in Bezug auf Effizienz, Fehleranfälligkeit und Genauigkeit zukünftig verbessert werden.

Ergebnisse und Schlussfolgerungen bezüglich OpenMI:

- OpenMI ist eine allgemeine Schnittstellenspezifikation, die den parallelen Modellkopplungsansatz standardisiert. Sie beschreibt die Art und Weise, in der Modelle ausgeführt und in der Daten zwischen ihnen ausgetauscht werden. Das als Migration bezeichnete Anpassen von Modellen an den OpenMI-Standard kann so verstanden werden, dass den Modellen beigebracht wird, dieselbe Sprache zu sprechen.
- Modellmigration an sich garantiert noch nicht, dass das Modell in jedem möglichen OpenMI-Szenario sinnvoll verwendet werden kann. Die Anwendbarkeit von OpenMI-Komponenten hängt in hohem Maße auch davon ab,
  - wie Modelle die OpenMI-Schnittstelle implementieren und
  - welche Austausch-Elemente (engl. exchange items) sie anbieten, d.h. welche Modellvariablen (z.B. Durchfluss, Wasserstand) für welche Modellelemente (z.B. Schächte, Haltungen eines Kanalnetzes) ausgetauscht werden können und welche Datenoperationen (z.B. räumliche Aggregation) von diesen Elementen angeboten werden.
- Die Migration einer existierenden Modellierungssoftware erfordert, dass der Quellcode der Software zur Verfügung steht. Der Migrationsprozess wird durch Software-Werkzeuge unterstützt, die von der OpenMI-Vereinigung (OpenMI Association) bereitgestellt werden. Der für die Migration einer existierenden Modellierungssoftware in eine entsprechende OpenMI-kompatible Komponente erforderliche Aufwand hängt neben den Aufgaben, die von der OpenMI-Komponente erfüllt werden sollen, von Erfahrungen des Softwareentwicklers und von technischen Voraussetzungen ab. Der Aufwand hängt darüber hinaus entscheidend von der Softwarestruktur des bestehenden Modellkerns ab.
- Von den Modellanwendungen, die bisher für die Verwendung in MIA-CSO vorgesehen sind, unterstützt nur die Software InfoWorks CS die OpenMI-Schnittstelle. Die aktuelle Implementierung von OpenMI in InfoWorks CS erlaubt jedoch nur Durchflüsse und Wasserstände als Eingangsgrößen in die InfoWorks OpenMI-Komponente.
- Ob OpenMI in existierende Modellanwendungen implementiert wird oder nicht, hängt von dem Anreiz für die Modellentwickler ab und von deren Fähigkeit, die Modellmigration durchzuführen. Zumindest in der nahen Zukunft ist es unwahrscheinlich, dass die Modelle HYDRAX und QSim, die für den Einsatz in MIA-CSO vorgesehen sind, mit der OpenMI-Schnittstelle ausgestattet werden.
- Die weiteren Entwicklungen am KWB werden sich deshalb auf die für das Planungsinstrument zusätzlich benötigten Werkzeuge konzentrieren, die z.B. statistische Auswertungen, automatische Kalibrierung, Unsicherheits- und Sensitivitätsanalyse ermöglichen. Dabei kann der OpenMI-Ansatz weiter verfolgt werden, was die eigenen Entwicklungen auch für Andere attraktiv machen kann.

## Ergebnisse und Schlussfolgerungen bezüglich API:

- Durch Anwendungs-Programmierschnittstellen (API) ermöglicht eine Software, eine Auswahl seiner internen Funktionen von einer anderen Software aus aufzurufen. Eine API ist spezifisch für die Software, die die API anbietet. Oft ermöglichen APIs den Zugriff auf diejenigen Funktionen einer Software, die von der Software selbst normalerweise über die graphische Benutzeroberfläche angestoßen werden.
- Im Vergleich zum OpenMI-Szenario werden die Modelle im API-Szenario sequentiell, also nacheinander, ausgeführt. Jedoch sind in diesem Szenario Modellvorbereitung, Datenimport, Simulationsstart und Datenexport durch den Aufruf entsprechender, über die API verfügbar gemachter, Funktionen automatisiert.
- Die von Wallingford Software Ltd. zur Verfügung gestellte Dokumentation der InfoWorks API ist unvollständig. Dies stellt ein Problem dar, wenn die InfoWorks API genutzt werden soll.
- Dieser Bericht beschreibt Schritt für Schritt, wie Teile der InfoWorks CS Software über deren API aus den Programmiersprachen VBA bzw. C# heraus verwendet werden können.
- Die InfoWorks API ermöglicht es, modellbeschreibende Daten und Modelleingangsdaten aus Dateien zu importieren. Modellparameter können indirekt durch eine Abfolge von Datenexport, Manipulation der exportierten Daten und Datenimport verändert werden. Auf gleiche Weise können z.B. im Modell hinterlegte Regenserien verändert werden. Über die API lassen sich nur komplett Simulationen über einen vorgegebenen Simulationszeitraum ausführen. Von den Simulationsergebnissen lassen sich über die API nur die Eigenschaften von Verbindungen (engl. links), wie Kanalhaltungen, Pumpen oder Wehre, in Dateien exportieren, nicht jedoch Eigenschaften von Knoten (engl. nodes).
- Die Möglichkeit, interne Modellparameter indirekt über die InfoWorks API zu ändern, erlaubt es, diese Schnittstelle z.B. für die statistische Auswertung, automatische Kalibrierung sowie Unsicherheits- und Sensitivitätsanalyse zu nutzen.

## Acknowledgements

This report was improved by conversations and discussions with several people. The author thanks all colleagues, who supported the work on this study. Particular thanks to Dr. Frank Schumacher (Dr. Schumacher Ingenieurbüro für Wasser und Umwelt, Berlin) for giving information about the programs HYDRAX, QSim and GERRIS and to Benoît Beraud (Veolia Water) for knowlegde transfer concerning the InfoWorks Application Programming Interface (API) and for the reviewing of this report. The author further thanks Frank Reußner (Technical University Darmstadt) for discussions on the OpenMI interface and the Wallingford Support team for providing information about the InfoWorks API as well as the InfoWorks implementation of the OpenMI.

## Table of Contents

Chapter 1 : Introduction.....	1
Chapter 2 : Materials and Methods.....	2
2.1 Involved Modelling Softwares .....	2
2.1.1 InfoWorks CS .....	2
2.1.2 HYDRAX .....	2
2.1.3 QSim.....	2
2.1.4 GERRIS .....	3
2.2 Model Coupling Techniques.....	3
2.2.1 OpenMI .....	4
2.2.2 Application Programming Interfaces (APIs) .....	5
Chapter 3 : Current Modelling Practice for CSO impact assessment in Berlin .....	6
3.1 Description of Current Modelling Practice .....	6
3.2 Disadvantages of Current Modelling Practice.....	8
Chapter 4 : Possible Model Coupling Scenarios.....	9
4.1 Fully OpenMI Integrated Model.....	9
4.2 Using Available APIs.....	12
Chapter 5 : Applicability of the OpenMI: Testing of existing components.....	16
5.1 Objectives and Strategy.....	16
5.2 Preconditions .....	16
5.2.1 Using the OpenMI Configuration Editor .....	16
5.2.2 Creating OpenMI components from InfoWorks CS models.....	17
5.3 Scenario A: Single InfoWorks CS component .....	19
5.4 Scenario B: Linking two different InfoWorks CS components.....	20
5.5 Scenario C: Linking InfoWorks CS to other OpenMI components .....	23
5.6 Results.....	25
5.6.1 Compatibility issues.....	26
5.6.2 Limitations of the OpenMI implementation in InfoWorks CS .....	27
Chapter 6 : Applicability of the OpenMI: Making existing models OpenMI-compliant.....	29
6.1 Model Migration .....	29
6.1.1 Requirements for an OpenMI-compliant component.....	29
6.1.2 Wrapping concept and wrapping pattern provided in the OpenMI SDK .....	30
6.1.3 Model migration in seven steps .....	32
6.2 Estimation of Effort.....	35
6.2.1 Tasks to be performed by the OpenMI component.....	35
6.2.2 Personal preconditions .....	35

6.2.3 Technical preconditions .....	36
6.2.4 Effort related to the different steps of model migration .....	36
6.2.5 Effort related to the models intended to be used in MIA-CSO .....	37
Chapter 7 : The InfoWorks Application Programming Interface (API) .....	38
7.1 Objectives.....	38
7.2 Materials and Strategy.....	39
7.2.1 InfoWorks API documentation.....	39
7.2.2 Chosen programming language and environment.....	40
7.2.3 Testing procedure .....	41
7.3 Results .....	41
7.3.1 Accessing the InfoWorks Type Library .....	41
7.3.2 Content of the InfoWorks Library.....	43
7.3.3 Calling individual API methods.....	44
7.3.4 Performing more complex tasks by calling API methods .....	49
Chapter 8 : Summary and Conclusions.....	52
Appendix A : Glossary of programming-specific terms .....	55
Appendix B : The IEngine interface .....	56
Appendix C : Step-by-step procedure to access the InfoWorks API from within VBA ....	59
Appendix D : Step-by-step procedure to access the InfoWorks API from within C#.....	64
Appendix E : Methods and properties of the InfoWorks library .....	69
Appendix F : Class Diagram of API test application .....	75
Appendix G : Evaluation of simulation results obtained by running an OpenMI-linked system .....	76
Appendix H : OpenMI Demo Models .....	79
Bibliography .....	80

## **Chapter 1: Introduction**

The overall objective of MIA-CSO is to develop a model-based planning instrument for impact based CSO control. This will be done by coupling InfoWorks CS (commercial sewer model by Wallingford Software Ltd.), QSim (water quality model developed by BfG, German Institute of Hydrology) and a module for statistical analysis of the simulation results. The planning instrument will be used for impact/recipient based scenario analysis of CSO management strategies. The impact assessment will be based on a minimum set of significant and operable parameters (dissolved oxygen and ammonia in the receiving water) under consideration of uncertainties. The water quality model QSim will be adapted to the Berlin situation. Monitored data (flows and water quality) will be available for model adaptation, calibration and validation.

In order to achieve these general objectives of MIA-CSO, methods are needed to facilitate the whole planning process including modelling, simulation and evaluation. Different approaches may be followed for model coupling and for the integration of the additionally needed tools (e.g. for model calibration, statistical analysis of results, uncertainty analysis, etc.).

This document describes the results of a practical study aiming at assessing some possible approaches. Particularly, OpenMI, the European standard for model linking, and the InfoWorks Application Programming Interface (API) are considered.

In order to provide tools helping to fulfill the requirements of the European Water Framework Directive, a consortium of European partners developed a standard for linking environmental models. This standard is called the Open Modelling Interface and Environment (OpenMI) and is already supported by some modelling softwares (e.g. the here used software InfoWorks).

Modelling softwares commonly provide a Graphical User Interface (GUI) which is the visible panel that enables the user to input or import data, setup a model, run a model, export results, etc. Additionally, some modelling softwares are shipped with a so called Application Programming Interface (API). An API gives access to a selection of program features of the software. So, it becomes possible to use parts of the software from within other softwares, without the need of user interaction through the GUI.

Objectives of this study are to:

- gain experiences with the Open Modelling Interface and Environment (OpenMI), especially with the OpenMI implementation of InfoWorks CS.
- estimate the effort needed to make models OpenMI-compliant.
- test the Application Programming Interface (API), provided for the modelling software InfoWorks.
- provide information on the applicability of OpenMI and API within MIA-CSO.

## **Chapter 2: Materials and Methods**

### **2.1 Involved Modelling Softwares**

#### **2.1.1 InfoWorks CS**

The software InfoWorks CS, produced by Wallingford Software Ltd in Wallingford, UK, is a hydrodynamic urban drainage modelling software which is able to simulate rainfall-runoff and the transport of sewage in the sewer system. Water quality is simulated by applying mass conservative approaches.

From its version 7, the InfoWorks program family supports the OpenMI interface. Furthermore, an Application Programming Interface (API) is provided.

#### **2.1.2 HYDRAX**

Hydrodynamics (flows and levels) of the CSO receiving river are simulated by the river transport software HYDRAX, developed by the German Federal Institute of Hydrology (BfG).

In HYDRAX, a river course is defined by river stretches which themselves are determined by their ending points. The state variables of the model are water levels, flows and flow velocities. For each river stretch, boundary conditions need to be assigned.

An implicit difference scheme is used to transform the Saint Venant equations, which describe the dynamic behaviour of the water body, into non-linear difference equations. The quadratic elements are linearised by the Newton approximation.

The software HYDRAX is written in the programming language C. It is delivered as an executable program that does not provide a graphical user interface. It gets all input data in terms of input files and delivers all results in terms of output files. These and other program parameters are given to the program on the command line. HYDRAX does not implement the OpenMI interface (is not OpenMI-compliant). HYDRAX does not offer an API (personal communication with Dr. Schumacher).

#### **2.1.3 QSim**

The water quality model QSim (Quality Simulation) is an instrument for simulation and prediction of element budgets and plankton dynamics in rivers. Since 1979, it is developed and applied by the German Federal Institute of Hydrology (BfG). The model is mainly used to determine and to estimate the effects that hydro-engineering actions may have on water quality of the federal waterways. QSim is also applied in order to work on problems concerning water management and river basin management.

QSim mathematically describes the complex chemical and biological processes that occur in flowing waters. The model combines hydraulic and ecologic model components which are able to calculate the most important biological processes of oxygen and nutrient balances, the growth of algae and zooplankton as well as the processes occurring at the river bed.

QSim is mainly used to produce annual hydrographs of oxygen content and other water quality parameters as well as biological variables (e.g. algae biomass) along a river stretch.

The current version of QSim is 10.0. The continual development of QSim takes into account the experiences of numerous applications of QSim for different river systems in Germany. Also in future, QSim will be applied by the BfG.

As the calculation of water quality depends on flow conditions and water levels in the river, these hydraulic information are an important boundary condition for QSim. The hydraulic conditions are simulated first by the river transport model HYDRAX (see above) and then given as an input to QSim.

The software QSim is written in the programming language Fortran. As HYDRAX, QSim is delivered in the form of an executable file and does not offer a graphical user interface. The program accepts program parameters (e.g. paths to the required input files). QSim does not implement the OpenMI interface (is not OpenMI-compliant). QSim does not offer an API (personal communication with Dr. Schumacher).

#### **2.1.4 GERRIS**

GERRIS is a program developed by *Dr. Schumacher Ingenieurbüro für Wasser und Umwelt*, Berlin. By offering a user interface, the software facilitates the usage of HYDRAX and QSim. GERRIS can be seen as a mediator between input data, HYDRAX model, QSim model and output data.

Via the user interface, the GERRIS program allows the user to setup a river network interactively. Information describing the model, boundary and initial conditions are stored in an internal database. The program offers an 'import assistant' to read input data. It can handle Excel files containing comma separated values (CSV). These data are stored in the database and also transformed into input files that are required by the HYDRAX and QSim model softwares. GERRIS takes care of the different formats of files that are required and produced by HYDRAX and QSim. In two steps, the user can first start HYDRAX to calculate the hydraulic behaviour of the river and then run QSim for simulating water quality. Therefore, the results of the HYDRAX simulation are used as boundary and initial conditions for the QSim simulation.

The GERRIS application is a Visual Basic program that accesses a Microsoft Access database for data storage. GERRIS does not implement the OpenMI interface (is not OpenMI-compliant). GERRIS does not offer an API (personal communication with Dr. Schumacher).

### **2.2 Model Coupling Techniques**

When linking different models together, corresponding variables, which may be defined differently in the different models, need to be related to each other. For example, if a model, which uses COD concentrations as a measure for organic loads, is linked to a model, which is based on BOD concentrations, it is important to define, how the different variables can be transformed from one representation into the other. It has to be paid attention when different models use variables of the same name but with a different meaning. Often, conversion factors are defined, which are applied at the interface between two linked models. It is also important to consider the different spatial or temporal discretisations that are applied in the different models.

Beside these conceptional considerations it is important to choose an appropriate software technique that can be used in order to realize data transfer between two models in each case. Regarding software techniques, two different approaches can be distinguished.

Models can be linked sequentially. In this case, the models are run one after the other, each over the whole simulation period. The simulation results of the first model are fed into the second model, which, on his part, does the simulation before it feeds its own results to the next linked model etc. Sequential coupling of models does not consider feedback of one (data receiving) model back to the linked (data providing) model.

In contrast to sequential coupling, models can be linked in parallel (synchronously). In that case, at any timestep, all the incorporated models are evaluated (for that timestep). By doing so, a kind of feedback between models becomes possible since the output of each model at a given timestep can be used as input of any other model at the next timestep.

Which type of model linking to choose not only depends on the available software techniques but mainly on the problem to be investigated: If feedback between models (e.g. reverse flow) is not important to consider, sequential coupling may be sufficient. However, if feedback needs to be considered (e.g. if the sewer system is controlled due to a state variable of a wastewater treatment plant model), parallel model coupling becomes necessary.

In terms of software realization, parallel coupling requires extensive possibilities in accessing the individual model software programs to be coupled. This may be given if the model software codes are freely available or if the modelling software programs offer adequate software interfaces which allow to use (parts of) the models functionality from within self written software programs. The European standard interface OpenMI (see <http://www.openmi.org>) is an example of a parallel coupling approach.

## 2.2.1 OpenMI

The Open Modelling Interface and Environment (OpenMI) is the outcome of the European project *HarmonIT* (see <http://www.harmonit.org>). Main part of the OpenMI is a standard specification which pre-defines a format for communication and data exchange between software modules (components) in general and computer models in particular. There have been three releases of the OpenMI. The current release is 1.4 (after releases 1.0 and 1.2). A new OpenMI standard release is not expected until the release of OpenMI standard version 2 (planned in December 2009).

OpenMI defines both the interfaces between models (i.e. the set of functions, also called methods, that each model can expect from any other OpenMI-model to be exposed) as well as the structure of data that is being exchanged. The interface allows models to exchange data at run time, i.e. it represents the parallel approach (see above). This is realized by a so-called *request-reply* architecture (see Figure 1).

In that architecture, the linked components ('A' through 'D' in Figure 1) act as providers and acceptors of data. A model component that depends on data from another model, requests that data from the corresponding model component (by calling the *GetValues* method) and waits until the same component replied on that request. The data transfer between OpenMI components is handled within a single thread, i.e. a component handles only one request at a time before acting upon another request. The data

exchange in an OpenMI system is triggered by invoking the last component in the chain of linked components (component ‘A’ in Figure 1). From that moment on, computation, synchronization and data exchange between all components is handled autonomously. There is no supervising authority that controls the sequence of model calls. Rather, the order of model calculations results from the sequence of requests that are sent between the models.

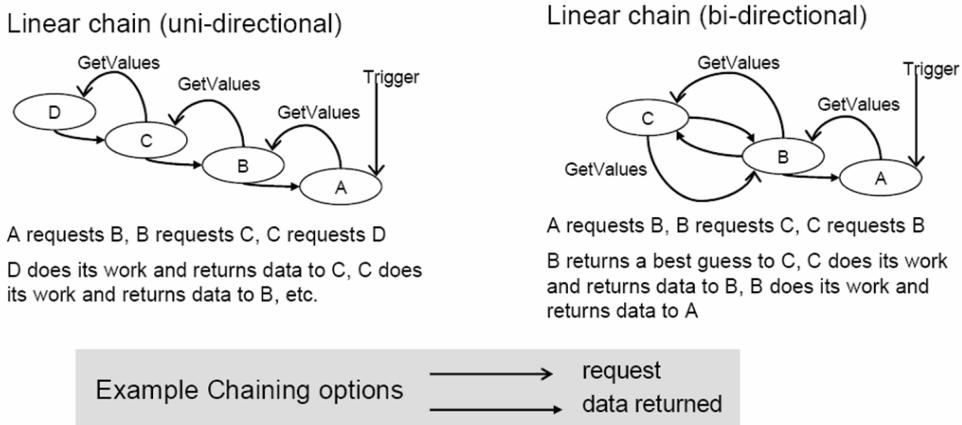


Figure 1: Linking of model components with OpenMI. Figure taken from (Gregersen et al., 2007).

The precondition for linking models using the OpenMI standard is that the models intended to be used support this standard. Making a model compliant with the standard is also referred to as *implementing* the standard. While the notation ‘component’ is generally used for software modules that are able to communicate with each other, the OpenMI terminology uses the term *Linkable Component* for components that implement the OpenMI specification. Models that are linkable components in this sense are also referred to as *OpenMI-compliant* models.

One major aim in the development of OpenMI was to make it easily feasible to render existing model software (legacy code) OpenMI-compliant. Therefore, the OpenMI dissemination contains tools for encapsulating legacy code into a so called wrapper that fulfills the OpenMI interface specification and that needs to be adequately connected to the code (cf. Chapter 6).

As the OpenMI specification is quite a new development, up to now there are only a few model software programs (mainly of big software companies) that are OpenMI-compliant (Sonnenberg, 2008).

## 2.2.2 Application Programming Interfaces (APIs)

An Application Programming Interface (API) is an interface, which a software system may provide in order to allow other software systems to use some of its functionality. The API gives access to a set of routines, data structures, object classes or other software elements. The API is a specification of the names and types of available objects, especially the routines (methods) and the parameters accepted by these routines. The software that provides the functionality described by an API is said to ‘implement’ the API. By providing an API, a software developer can give access to its software without revealing the underlying source code.

# Chapter 3: Current Modelling Practice for CSO impact assessment in Berlin

## 3.1 Description of Current Modelling Practice

This chapter describes the modelling practice which has been applied when the effects of CSO on the receiving river has been simulated in former studies (Schumacher et al., 2007).

A sequential coupling approach was applied, i.e. the models have been run one after the other with the results of one model being fed into the following model in each case.

Rainfall runoff and sewage transport was simulated by InfoWorks CS, resulting in the discharges and water quality parameters of CSO at the river outlets. The simulated discharges have been used as input data for the model software HYDRAX which simulated river hydraulics. The results of the HYDRAX simulation (water levels and flows along the considered river reach), together with the water quality parameters simulated beforehand by InfoWorks CS, have then been used as input data to the river quality model QSim.

Each of the applied modelling software operates according to the following principle, which is typical for modelling applications (Figure 2): The user supplies information in the form of input files (light blue boxes labelled ‘Input’). In these input files, the specific scenarios of the system to be modelled (e.g. sewer network, river network as well as initial and boundary conditions like specific rainfall events) are described. In creating the input files, the user might be supported by graphical user interfaces (GUI; grey boxes in Figure 2). The calculations of the modelled systems take place in the so called model engines (orange boxes). Usually, the engines are generic representations of the processes. A model engine reads the input files, performs the calculations and outputs the results again to files (light blue boxes labelled ‘Output’).

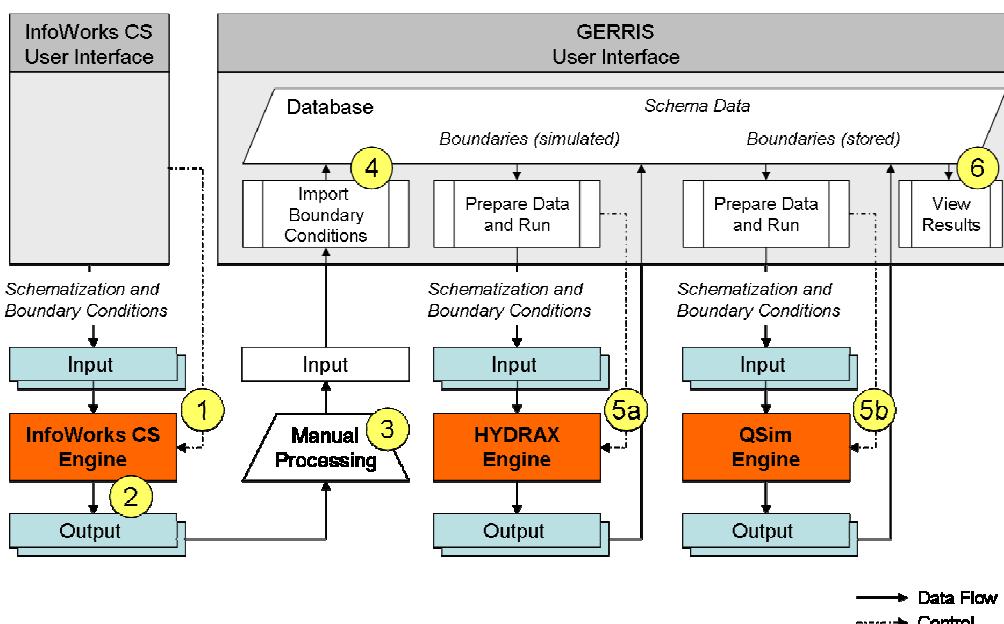


Figure 2: Interaction of models being applied for CSO impact analysis in Berlin. The numbers indicate the steps within an integrated model run, referenced in the main text.

As the involved programs do not share a common file format for input and output files, result files being produced by one model cannot just be taken as an input to the next model. Data transformations have to be performed on result files in order to produce input files which fulfill the semantic and formal requirements of the next model in each case.

Concerning the river models HYDRAX and QSim, the necessary transformations between HYDRAX output files and QSim input files are performed within the software GERRIS, which manages all data that is required by any of the two models, over a common graphical user interface.

Main part of these data are the CSO characteristics (quantity and quality), simulated by InfoWorks CS. The simulated overflow events are imported into the GERRIS database by use of the so called ‘import assistant’, a sub-module within the GERRIS software which offers a dialog window and so interactively supports the import process.

The data format exported from InfoWorks CS does not correspond to the format that is expected by the import assistant of GERRIS. Thus, the result files produced by InfoWorks (in CSV-format; CSV = *comma separated values*) need to be adapted (manually or automated) to the file structure that is understood by the import assistant.

The preliminary actions to be performed in order to prepare simulations are:

- Related to the sewer system and InfoWorks CS:
  - o Setup of the sewer network model. *Berliner Wasserbetriebe* hold and maintain InfoWorks CS networks of different Berlin catchments.
  - o Preparation of input data (e.g. rainfall series) and import to InfoWorks.
  - o Definition of real time controls (RTC) in InfoWorks CS. RTC are used to describe the controlling of the pumping stations and further actuators.
  - o Definition of a specific simulation run including start time, duration of the simulation and other simulation parameters like simulation timestep, gauging timestep, applied RTC scenario, etc.
- Related to the river system and HYDRAX/QSim/GERRIS:
  - o Setup of the river model. The GERRIS user interface allows to interactively define river sections, cross sections, etc.
  - o Preparation of input data that is not related to CSO, and thus will not be delivered by the InfoWorks CS simulations. Related to water quality simulations, these data are for example weather conditions or chemical or biological parameters like concentration of algae. Of these parameters average values are assumed due to lack of measured data.

When the models are prepared, the steps to be performed for an overall simulation of the coupled sewer/river system including data transfer are (see Figure 2; the numbers in yellow circles correspond to the step numbers mentioned here):

Step 1: Simulation of the sewer system in InfoWorks CS. The InfoWorks CS engine is started from within the InfoWorks GUI.

Step 2: Export of simulated overflow hydrographs from InfoWorks CS into CSV files. For each quantity (e.g. flows, levels, concentrations of COD, BOD) a separate file is produced by InfoWorks.

Step 3: Manual preparation of exported CSV files for import to HYDRAX/QSim.

Step 4: Import of prepared data into the internal database of GERRIS via the ‘Import Assistant’ of GERRIS.

Step 5: Simulation of river system by first running HYDRAX (5a) and then running QSim (5b). The simulations are invoked from within the GERRIS GUI.

Step 6: Results from the river simulations can be viewed within GERRIS or exported to files (export is not considered in Figure 2).

### **3.2 Disadvantages of Current Modelling Practice**

The modelling procedure described in the last section can be improved in terms of efficiency, error-proneness and accuracy.

Using different non-compatible modelling software programs comes with the drawback of several steps in the modelling and simulation practice that have to be performed manually (see above section). The manual processing is time-consuming and requires the presence of the modeller to start the next process in each case. In InfoWorks CS, for example, starting the simulation and exporting the simulation results (which includes loading of the results, defining the elements and variables to export and running the export process) are manual processes that are invoked via the InfoWorks GUI. The same holds true for the import of simulated CSO into the GERRIS database and for the invoking of the HYDRAX and QSim simulations via the GERRIS GUI.

It would be an improvement if the different steps to be performed could be defined in terms of a ‘batch file’ and if the corresponding processes could be run automatically. As long as the number of simulations to be carried out is small, the advantage of automated processing does not carry weight compared to a manual procedure. However, for future applications, which may take into account many repeated simulation runs, an automatic approach could become indispensable.

Until now, at the interface between InfoWorks CS and HYDRAX/QSim/GERRIS, a manual processing of the result files produced by InfoWorks is necessary (see last section, Step 3). The CSV files exported from InfoWorks CS have been joined into Excel files. The Excel files were modified (e.g. applying thresholds, reducing temporal resolution by averaging rows, addition and renaming of columns, etc.) before they were imported by the GERRIS Import Assistant. That kind of manual creation of input data is error-prone. The correctness of produced data highly depends on the person who edits the data. Automating the transformation of data to be transferred would not only speed up the exchange process but also avoid accidental typing errors.

Sequential coupling of models does not take into account potential feedbacks of the data receiving model (here: river model) back to the data providing model (here: sewer model). In the real system, such a feedback could be given for example if water levels in the river, which are elevated due to CSO, cause backwater in the sewer and thus have an effect on the overflow behaviour. However, for the Berlin system (Stadtspree) there is no relevant physical feedback that needs to be taken into account. Another type of feedback could become necessary for the investigation of more elaborated control strategies, which, for example, control pumps or other elements in the sewer system according to water quality parameters in the river. For the type of simulations that have been carried out so far, the sequential, uni-directional approach is not assumed to have a negative effect on the quality of simulation results.

## **Chapter 4: Possible Model Coupling Scenarios**

In the last chapter, the current modelling practice being applied in Berlin, has been introduced. Drawbacks of that practice have been discussed. Here, two different theoretic modelling scenarios, in which some of these drawbacks are overcome and which may be applied in future, are presented. First, in section 4.1, a solution is discussed, in which all involved models and tools communicate and exchange data via the OpenMI (see 2.2.1). That scenario assumes that all required models and evaluation tools are available in the form of OpenMI-compliant components. Second, in section 4.2, a scenario is presented, in which the models are not accessed through the standardized OpenMI interface but through specific Application Programming Interfaces (APIs, see 2.2.2). That solution requires the existence of such APIs for the models and additional tools involved.

Of the model software used to date, only InfoWorks CS supports the OpenMI. Also, that software is the only one of the incorporated programs which offers an API. This means that none of the two presented scenarios is feasible at present. However, they could become feasible if the desired interfaces were implemented in the corresponding software.

Assuming that the software provided will be used unchanged in future and thus, none of the presented scenarios will become practicable, there are other alternatives that may improve efficiency and quality of the modelling practice. The non-existence of appropriate interfaces may prevent the automation of data exchange between models. However, even if data import, simulation and data export need to be invoked manually, the manual data exchange may be supported by the use of appropriate converter programs. These programs may be applied in order to transform simulated data that has been exported from one model program into the format that is expected by the model to which that data is going to be imported. With regards to the current modelling practice (see 3.1), such a tool may substitute the manual creation of Excel input files as it is needed for import to the GERRIS program (Step 3 in 3.1). The converter tool could be realized in the form of a configurable command line program. Being started from the Windows command prompt and given the path to the input files, that program may perform data transformations according to the given specification and produce an output file that can be imported into GERRIS. Such a utility is currently being developed at KWB.

### **4.1 Fully OpenMI Integrated Model**

Figure 3 demonstrates the modelling scenario in which all models and tools are linked via the OpenMI. That scenario assumes that all models and additional tools (e.g. Database, Calibration Controller, Statistical Analysis, Uncertainty and Sensitivity Analysis) are available in the form of corresponding OpenMI-compliant components (see 2.2.1). These components, which in Figure 3 are symbolised by blue ellipses, may have been received by ‘wrapping’ existing model softwares (white ellipses; see also Chapter 6). To keep more general, the components in Figure 3 are labelled with their type and not with concrete model names.

The integrated system of OpenMI components could be configured either in a ‘hard-coded’ way or by use of a Configuration Editor as it is for example developed and

distributed by the OpenMI Association Technical Committee (OATC). For a hard-coded system, the modeller needs to have access to a programming environment in which software can be written in a programming language that is able to access the OpenMI components. The modeller is then in the role of a programmer. Loading components and establishing links between the components are performed by means of language-specific commands and by calling the OpenMI-specific methods on the components. By contrast, using an application like the OATC Configuration Editor (see also 5.2.1) may facilitate the configuration of the integrated model system. The editor offers a graphical user interface (GUI) that allows to interactively define the model components to be used and the links to be established between them. Running a hard-coded system is done by compiling and running the corresponding self-written piece of software. Using a configuration editor, however, lets you start the simulation of the integrated system from within the GUI.

The GUI approach allows non-programmers to modify the linking of models while the hard-coded way prevents non-programmers to modify the model linking or even to include new components.

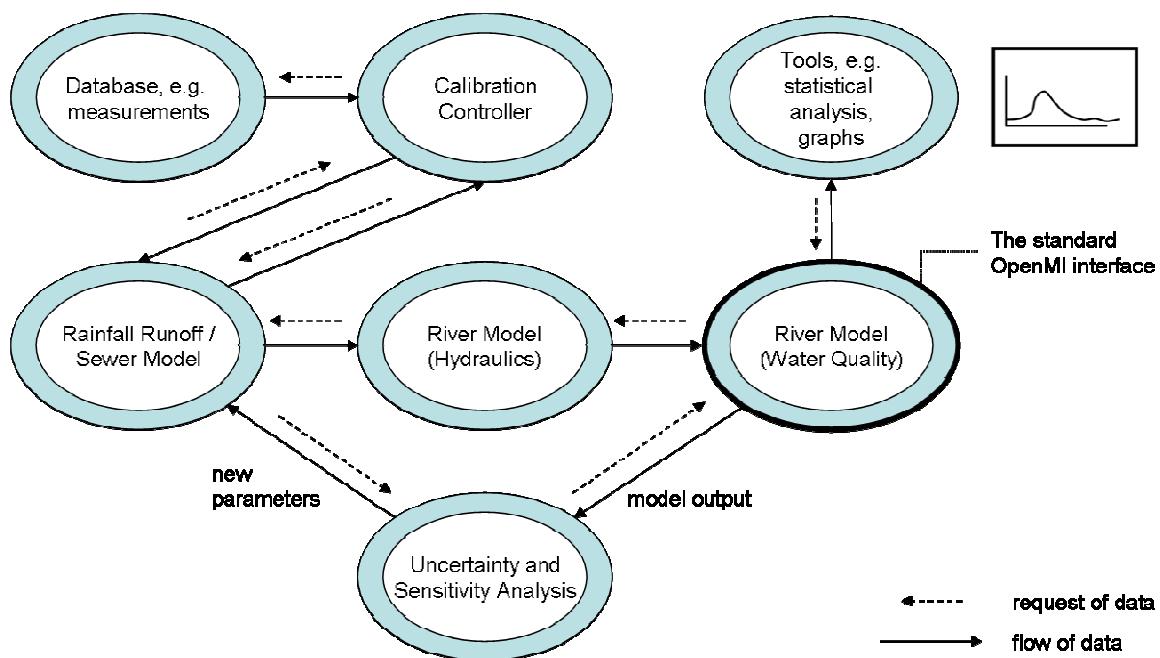


Figure 3: The OpenMI approach for linking models and tools within MIA-CSO. Figure adapted from (Gijsbers et al., 2007b).

In the integrated sewer/river system, the three models (rainfall runoff and sewer, river hydraulics and river quality) are linked in a chain.

In OpenMI terminology, a *link* defines *what* (which quantity, e.g. flow, level, mass concentration of BOD) is exchanged *where* (at which locations) and between which models (data requesting target model and data providing source model).

In the existing case, there are links for the quantity ‘flow’ between the sewer and the river hydraulics model and links for flow and for water level between the river hydraulics and the river quality model. Furthermore, there need to be direct links between the sewer and the river quality model which transfer the water quality data (mass concentrations of suspended solids, BOD, COD, etc.). For purposes of clarity, the latter mentioned links

are not indicated in Figure 3. Assuming that there are neither feedbacks between river quality and river hydraulics nor feedbacks between river hydraulics and the sewer system, there are no links directed ‘backwards’. However, the OpenMI would enable these feedback links.

Concerning the geographical locations that are linked, for example each link between sewer and river model may connect one (outlet) node of the sewer model to exactly one corresponding node of the river model. Such a direct one-to-one relationship between locations (sewer nodes to river nodes) may not be possible due to different spatial representations (e.g. points versus lines versus grids, one dimensional versus two dimensional, etc.) being applied in the models to be linked. The OpenMI specification is designated to account for these differences; however, the needed conversion functions (e.g. aggregation, averaging, interpolation, translation between different reference systems) need to be envisaged and implemented in the concerned model components.

For each quantity represented by a link, the dimension of the quantity (e.g. length, time, mass, velocity = length per time) and a conversion factor to SI units (SI = Système International d'unités) must be given. Thus, the OpenMI can account for different units of the same dimension (e.g. feet to metres, both being units of the dimension ‘length’). However, the OpenMI does not release the modeller from conscientiously checking the meaning of the different quantities in the different models. Further transformations between quantities may become necessary; it is the responsibility of the model developer to offer these types of transformations in terms of so called *data operations* which the model component may provide for its different quantities.

Considering the three model components, an overall simulation would be performed either by a simple trigger component or by a superior controlling component (like the component for ‘Uncertainty and Sensitivity Analysis’ in Figure 3). The last model in the chain (here the river quality model) would be requested (dotted arrows in Figure 3) to deliver the water quality for the last timestep of the intended simulation period. The water quality model would start its calculations at the beginning of its simulation time horizon. Each time the model needs a value of a quantity that is linked to another model, it requests that value for the actual timestep from the corresponding model and waits until that model delivers the value (solid arrows in Figure 3). So the river hydraulics model would be requested to deliver water levels and flows that are valid for the first point in time. The river hydraulics model on his part would start its calculations at the beginning of its own time horizon until it reaches the requested point in time. On that way it needs the information about possible CSO discharging into the river. It requests the CSO flows from the sewer model which on his part would start its calculation until the requested point in time is reached. The sewer model would then stop and wait for the next request.

A strength of the OpenMI is that it is able to link models which advance in time with different timesteps. Therefore, the OpenMI interface requires the models to be always able to deliver a value for a requested quantity at any requested point in time. The model is responsible to implement the interpolation or extrapolation of available values in order to return a ‘best guess’ for the value at the requested point in time.

Summarized, the request and reply mechanism of OpenMI allows for data exchange on a timestep basis and thus realizes the parallel model coupling approach. All model components run quasi-simultaneously, i.e. only one model runs at a time but altogether, they follow the same time bar. The OpenMI is designated for temporal and spatial mapping. Default implementations of the algorithms needed to perform these mappings

are provided by the OpenMI Association. However, it is the responsibility of the model developer to implement these algorithms or more sophisticated mapping procedures as needed for the specific model software.

In theory, the additional tasks being planned within MIA-CSO (aid in model calibration, statistical analysis of model results, uncertainty and sensitivity analysis) can also be performed by use of OpenMI. For model calibration, an OpenMI component ('Calibration Controller' in Figure 3) could be provided that on the one hand reads observed data from a database and on the other hand requests simulation results from the model to be calibrated (for example 'Rainfall Runoff/Sewer Model' in Figure 3). Following the OpenMI concept, the model to be calibrated would on his part ask the calibration controller for a new parameter set to apply. The controller would deliver a parameter that, according to already tested parameter sets, it guesses to give a better match between simulated and observed values. Based on the given parameters, the model to be calibrated would run a simulation and return the simulation results that are requested by the controller. The controller would judge the fitness of the parameter set and prepare a new parameter set to be tested next.

In a similar way, uncertainty and sensitivity analysis could be performed by linking a corresponding OpenMI component on the one hand to the model to which changes in parameter or input data shall be applied and, on the other hand, to the last model in the chain of which the final simulation results are requested.

Comparatively easy should it be to develop an OpenMI component for statistical analysis of the simulated water quality conditions in the river. Such a component would regularly request the variables to be analysed from the river quality model and update the statistics which could be saved internally or even exposed to other components via the OpenMI.

An OpenMI configuration as described above (sewer modelling, river hydraulics and river quality plus additional data processing tools) is highly complex. As far as the author knows, such complex systems are not yet reported in the scientific literature. At least the example of a calibration controller is explicitly mentioned in the OpenMI documentation series (Gijsbers et al., 2007a).

Furthermore, the feasibility of an OpenMI scenario as described here strongly depends on the availability of OpenMI components as well as the willingness of model developers to improve their actual OpenMI implementations.

For InfoWorks, which out of the three models intended for use in MIA-CSO is currently the only one being OpenMI-compliant, the actual possibilities and restrictions of the OpenMI implementation are highlighted in Chapter 5.

## 4.2 Using Available APIs

As already anticipated in the previous section, a fully OpenMI integrated modelling system does not seem to be feasible within near future. Indeed, yet few modelling software programs do support the OpenMI, being a standardized way for model coupling. However, there are more programs available that can be accessed through (software-specific) Application Programming Interfaces (APIs; see 2.2.2) that also may be used to automate model runs and data exchange between models. Moreover, it is easier for software developers to give access to parts of an existing software in the form of an API than to offer a standardized interface like the OpenMI. While the former only

needs to build a generically accessible library out of parts of the software, the latter may require profound structural changes of the software involved (cf. Chapter 6). A drawback of model linking by use of API is that the links built between two models via their API are specific to these two models. If the second model has to be replaced by another one, specific new developments will be required.

This section describes a general approach that uses APIs for model and tool integration within MIA-CSO. API is not a standard but just a technology to make parts of an existing software accessible to other software without the need for the existing software to reveal its source code. Thus, the possible options for model and tool integration depend on whether the used software programs provide access through APIs and, if so, which functions (methods) of the software are made available.

As mentioned in 2.1, of the modelling software intended for use in MIA-CSO, only InfoWorks CS is equipped with an API. The results of a detailed investigation of that API are described in Chapter 7. Here, some of the gathered information are anticipated, while assumptions are made about possible APIs that the other programs (which so far do not offer access through an API) could possibly be able to provide in future.

Often, an API gives access to those functions of a software that are normally initiated by pressing a button or choosing a menu item in the Graphical User Interface (GUI) of the software. This holds true for the InfoWorks API: among others, the API offers methods for importing different kinds of data (like networks, rainfall data, etc.), for setting up and starting a simulation run and for exporting the results of completed simulations. However, the API does neither give direct access to most of the internal variables like the properties of the network nor does it allow to calculate intermediate model states; only complete runs of pre-defined simulations can be triggered through the API.

Considering these restrictions, which are assumed to be valid also for other modelling programs, the following procedure may be applied for running an integrated model:

1. Prepare all models (as far as invariant data like sewer network, river network or parameters that are assumed to be constant during simulation are concerned) in the 'normal' way by use of the particular application's GUI.
2. Perform the sewer simulation by calling the appropriate API methods (offered by the sewer model) for:
  - a. loading the sewer simulation as prepared in 1.,
  - b. running the prepared simulation and
  - c. exporting the simulation results to files.
3. Programmatically read the results exported in 2.c. and transform them into the format that is requested by the next model in the chain (river hydraulics).
4. Perform the river hydraulics simulation by calling the appropriate API methods (assumed to be offered by the river hydraulics model) for:
  - a. loading the river hydraulics simulation as prepared in 1.,
  - b. importing the boundary conditions regarding CSO discharges by reading the files produced in 3.,
  - c. running the river hydraulics model and
  - d. exporting the results to files containing flows and levels along the river.

5. Programmatically read the CSO quality parameters exported in 2.c. and the hydraulics results exported in 4.d. and transform them into the format that is requested by the next model in the chain (river quality).
6. Perform the river quality simulation by calling the appropriate API methods (assumed to be offered by the river quality model) for:
  - a. loading the river quality simulation as prepared in 1.,
  - b. importing the boundary conditions regarding river hydraulics (flows, levels) and CSO quality data by reading the files produced in 5. and
  - c. running the river quality model.

The theoretical procedure described above is kept very general. It treats the integrated problem modularly and distinguishes the three models for sewer system, river hydraulics and river quality. In case of the specific river modelling software HYDRAX and QSim (see 2.1.4), however, the link between these two models has already been realized in the form of the software GERRIS (see 2.1.4). Assuming that the program GERRIS is made accessible via an API, step 3. of the above procedure could be changed in such a manner that the results from 2.c. are transformed programmatically into the format that is expected by the ‘import assistant’ of the GERRIS software. Steps 4. through 6. could then be reduced to:

- 4\*. Perform the river simulation (consisting of both, river hydraulics and river quality) by calling the appropriate API methods (assumed to be offered by GERRIS) for:
  - a. loading the river simulation as prepared in 1.,
  - b. importing the boundary conditions regarding CSO discharges (flows, levels) by calling the ‘import assistant’ on the file produced in 3.,
  - c. running the HYDRAX simulation,
  - d. defining the HYDRAX results as hydraulic boundary conditions for QSim,
  - e. importing the boundary conditions regarding CSO discharges (quality parameters) by calling the ‘import assistant’ on the file produced in 3. and
  - f. running the QSim simulation.

Finally, the results of the river quality simulation could be viewed and judged within the GERRIS user interface.

Compared to the OpenMI scenario (see 4.1), in this API scenario the models are not processed in parallel, but sequentially. However, model preparation, data import, start of simulation and data export are automated using the provided API functionality. A superior software needs to be developed in order to implement the above procedure in terms of the corresponding API calls. That controlling software would need to keep track of all the particularities of the used model software and their APIs. The developer of that software would need to know about the data formats that are both, expected by and produced by the different software. The software product would be specific for the actual case and for the models (and their APIs) involved.

The additional tasks (calibration, uncertainty and sensitivity analysis, statistical analysis) could also be performed by means of API access. The first two of these tasks rely on the ability to change specific model parameters. As it is the case for InfoWorks CS, the API may not give direct access to these parameters. In that case, and if the API allowed for

import and export of model parameters, the modification of parameters could be performed indirectly by

- exporting the model parameters to a file,
- changing the model parameters in the file and
- reimporting the (changed) model parameters into the model software again.

For uncertainty and sensitivity analysis, the software SimLab, which is freely available for non-commercial use (see <http://simlab.jrc.ec.europa.eu>) could be applied. That software is distributed in terms of libraries that can be accessed through an API.

# **Chapter 5: Applicability of the OpenMI: Testing of existing components**

## **5.1 Objectives and Strategy**

In order to get first experiences with OpenMI, some practical tests with simple scenarios have been carried out. For all tests, the OpenMI Configuration Editor (available from the OpenMI distribution) has been used for the setup of the model.

The different considered scenarios are:

- Scenario A: Running an InfoWorks CS component alone, only linked to a trigger.
- Scenario B: Linking two different InfoWorks CS components and running the linked system.
- Scenario C: Linking an InfoWorks CS component to another OpenMI-compliant component and running the linked system.

In the following, some preconditions needed to perform the tests are explained in section 5.2. Then, the above scenarios are discussed in the sections 5.3 through 5.5. Finally, section 5.6 summarizes the results obtained.

## **5.2 Preconditions**

Section 5.2.1 briefly describes the OpenMI Configuration Editor, which is applied for all test scenarios. Then, section 5.2.2 explains how the InfoWorks OpenMI components, that are needed for the tests, are generated.

### **5.2.1 Using the OpenMI Configuration Editor**

The OATC Configuration Editor is a program provided by the Technical Committee of the OpenMI Association (OATC). Equipped with a graphical user interface, the program allows to create, load, save and run OpenMI configurations. An OpenMI configuration is the description of the OpenMI-linked model system. It contains information about which model components take part in the system and about how they are linked. The Configuration Editor visualizes the configured system by means of rectangles (representing the OpenMI components) and arrows (representing the links between the components). Figure 4 shows how the InfoWorks CS component is loaded into the OpenMI Configuration Editor and how the component is represented graphically.

If ‘Composition → Add Model’ is selected (left in Figure 4), the user is asked to choose a file with the extension ‘.omi’. That file contains a description of the OpenMI component to be loaded (see the second requirement for OpenMI-compliance in 6.1.1). The next section describes how the required .omi-file can be obtained for an InfoWorks CS model that is planned to be placed within the OpenMI configuration (right in Figure 4).

In fact, an OpenMI component, described by an .omi-file, does not only represent the model itself (e.g. specific InfoWorks CS sewer network) but it also defines start time and duration of the model run as well as the specific boundary conditions (e.g. waste water conditions, rainfall data) to be applied. In InfoWorks, this model setup is referred to as the ‘Hydraulic Run’ (cf. 5.2.2). The same model component (identified by a unique model ID in the .omi-file) cannot be loaded twice into the Configuration editor. Otherwise, the

exception ‘Composition already contains model with ModelID ...’ occurs. However, it is possible to load for example two Infoworks components that represent different ‘Hydraulic Runs’.

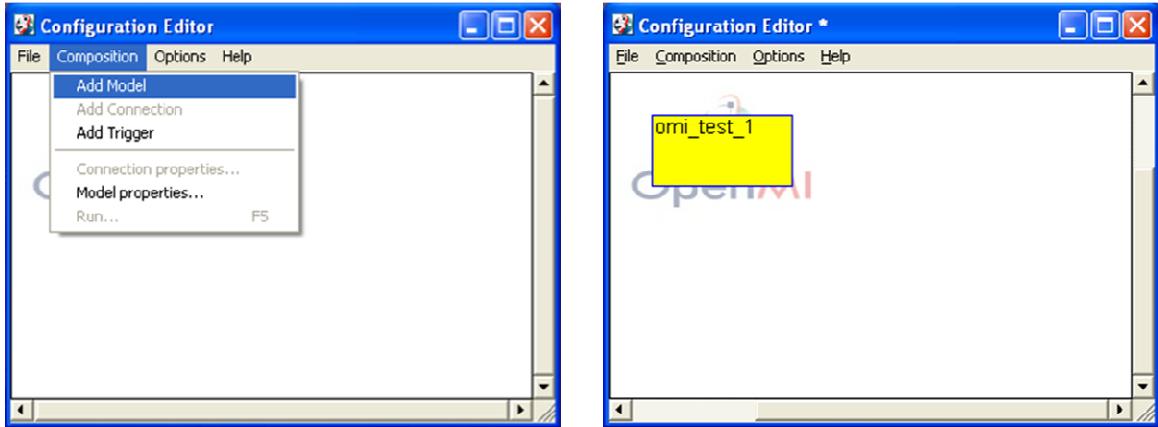


Figure 4. Loading an OpenMI component into the OpenMI Configuration Editor. After ‘Add Model’ has been chosen (left) and an .omi-file has been selected, the component is shown in the form of a yellow rectangle (right).

All reported tests were undertaken with InfoWorks 9.5 and OATC Configuration Editor 1.4.0.0. These versions are compatible (see 5.6.1).

### 5.2.2 Creating OpenMI components from InfoWorks CS models

An OpenMI-compliant InfoWorks CS component is created by an export process. In the InfoWorks CS application, the option ‘Export to OpenMI’ needs to be chosen when a Hydraulic Run is scheduled (see item one in Figure 6). Then, pressing the ‘Run Simulations’ button does not start the simulation but exports an OpenMI component. The exported component represents the specified simulation run. In fact, the OpenMI export process produces an .omi-file that describes which model engine needs to be populated with which kind of data in order to represent the currently specified Run.

When exporting to OpenMI, an input dialog appears that allows to specify the export (see Figure 5).

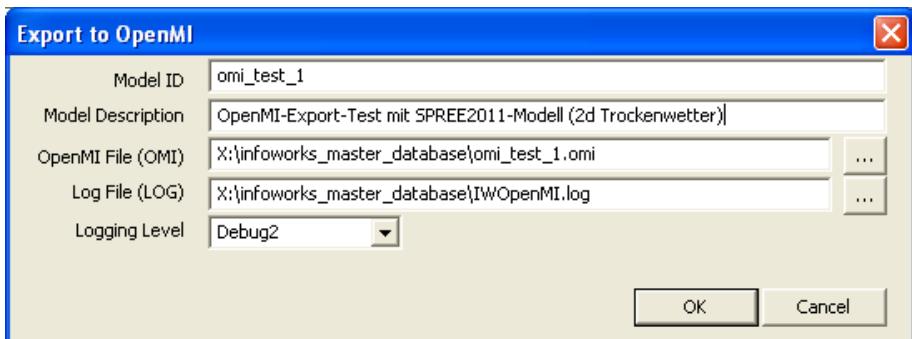


Figure 5: OpenMI export dialog. Model ID, model description and the locations of the files to be created (.omi-file and logfile) can be specified.

The specification in the ‘Schedule Hydraulic Run’ also determines, which input exchange items the exported OpenMI component will offer. It has been found out that the exported OpenMI component will only then offer input exchange items for

- flows, if the Inflow property has been set with an appropriate Inflow object, and for
- levels, if the Level property has been set with an appropriate Level object (see Figure 6).

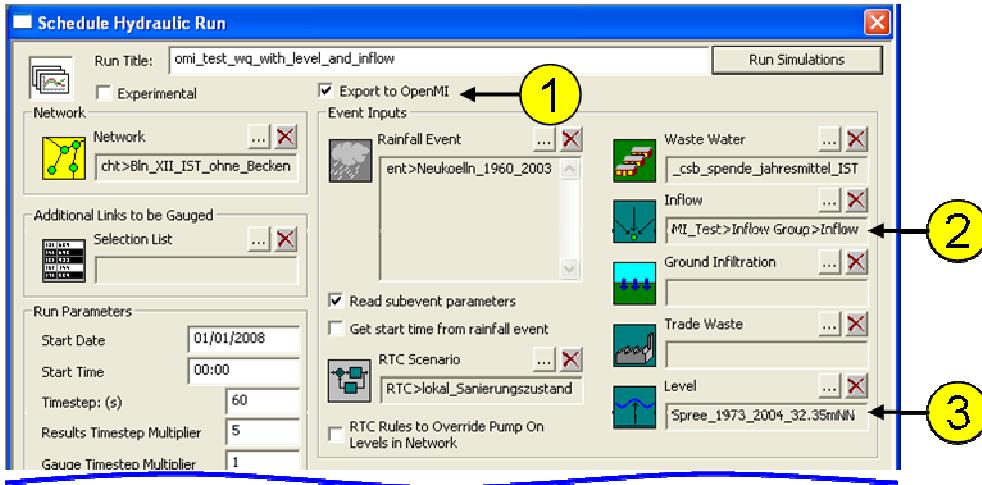


Figure 6. ‘Schedule Hydraulic Run’ dialog in InfoWorks CS. The checkbox ‘Export to OpenMI’ needs to be ticked if an OpenMI component shall be exported (1). Inflows need to be specified (2), if the OpenMI component shall contain input exchange items for flow. Levels need to be specified (3), if the OpenMI component shall contain input exchange items for levels.

In InfoWorks, Inflow and Level boundaries are given in terms of tables (Figure 7, left). The columns of the tables represent the network nodes, for which inflows or levels, respectively, are defined. Only for these nodes, corresponding input exchange items for Inflow and Level, respectively, will be available in the exported OpenMI component (Figure 7, right).

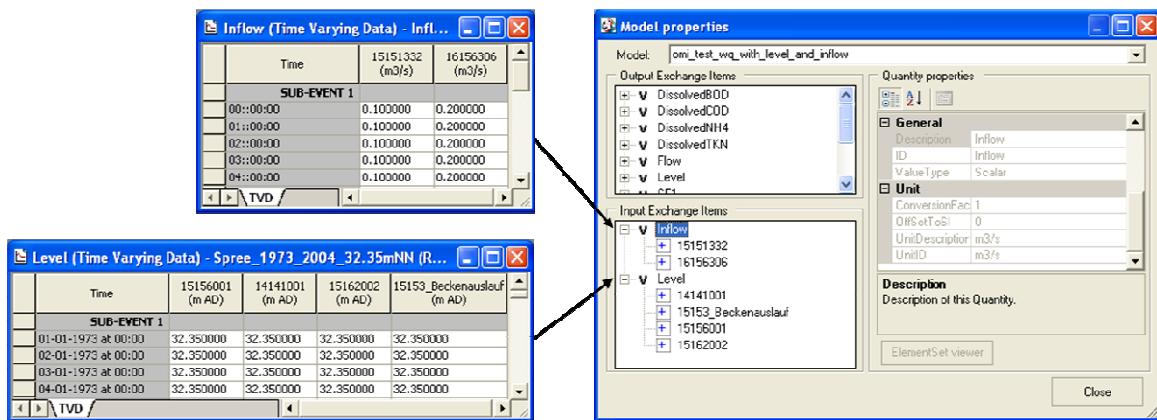


Figure 7. Representation of Inflow and Level data in InfoWorks (tables on the left) and the corresponding input exchange items as they are shown in the ‘Model properties’ dialog of the OpenMI Configuration Editor (right).

It could be expected that specifying other ‘Event Inputs’ in the ‘Schedule Hydraulic Run’ dialog would accordingly produce input exchange items for the corresponding types (e.g. Waste Water, Trade Waste, etc.). However, this is not the case. On request, Wallingford’s support team confirmed that the current OpenMI implementation in InfoWorks only allows for flows and water levels as the possible input quantities.

### 5.3 Scenario A: Single InfoWorks CS component

Running a single InfoWorks CS component, being only connected to a Trigger component (configuration as shown in Figure 8 and Figure 9) worked as expected.

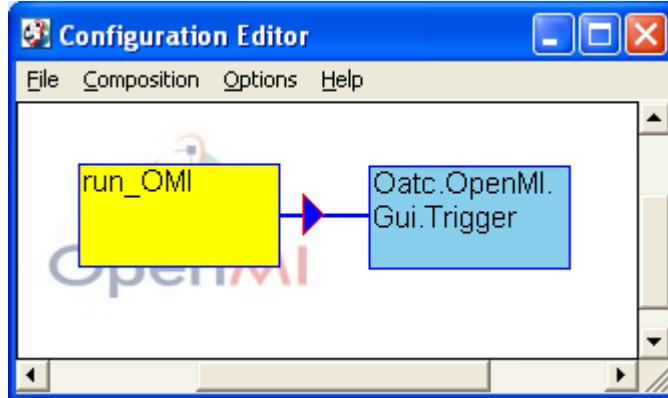


Figure 8: InfoWorks CS component 'run\_OMI', only linked to the OpenMI standard Trigger component in the OpenMI Configuration Editor.

The simulation results produced by the InfoWorks component ('run\_OMI' in Figure 8) were exactly the same as the results which were produced when the scheduled Run, from which the OpenMI component was exported (see 5.2.2), was invoked in the usual manner from within the InfoWorks GUI.

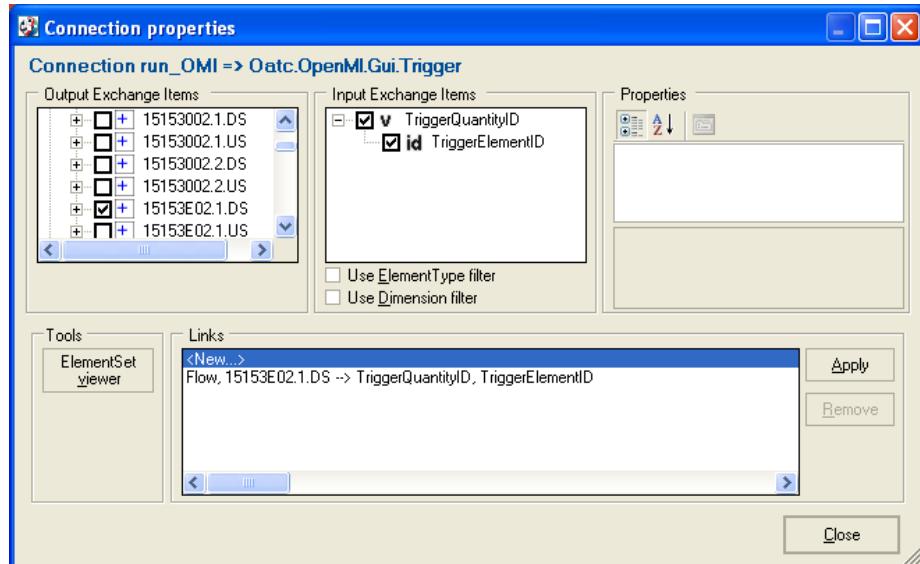


Figure 9: Configuration of the link between the InfoWorks CS component and the Trigger component. The trigger requests the flow at the downstream (DS) end of link '15153E02.1'.

If the InfoWorks CS model is run as an OpenMI component, the simulation does not necessarily perform until the end of the component's time horizon. It runs just as long as the time, at which the trigger is invoked, is not reached. Triggering the component at earlier points in time leads to earlier endings of the simulation.

## 5.4 Scenario B: Linking two different InfoWorks CS components

In the second scenario, two different OpenMI components of the same model type (InfoWorks CS) were coupled. Therefore, an existing InfoWorks CS model network was divided into a northern and a southern part (see Figure 10).

In the original model, both systems were connected by exactly one conduit (conduit '16154312.1'). The downstream node of that conduit (node '16156306') remained in both sub-models. In the northern network, that node was transformed into an outfall. In the southern network, the same node was specified in such a way that it accepted a direct Inflow. A corresponding Inflow object was created. For a given list of nodes, the Inflow object specifies the corresponding timeseries of direct inflows. In this case, the timeseries were filled with zero values (see right sub-window in Figure 10). The Inflow object is needed for the OpenMI export procedure. That procedure reads the node identifiers from the Inflow object in order to know, for which nodes the OpenMI component shall provide corresponding input exchange items (cf. 5.2.2).

Out of both, the northern and the southern network, OpenMI components were created by means of the OpenMI export procedure (see 5.2.2). Both Run specifications, on which the OpenMI exports are based, were configured equally except for the network to be simulated: The Runs were configured such that they simulate the same rainfall event and the same waste water conditions over the same time horizon.

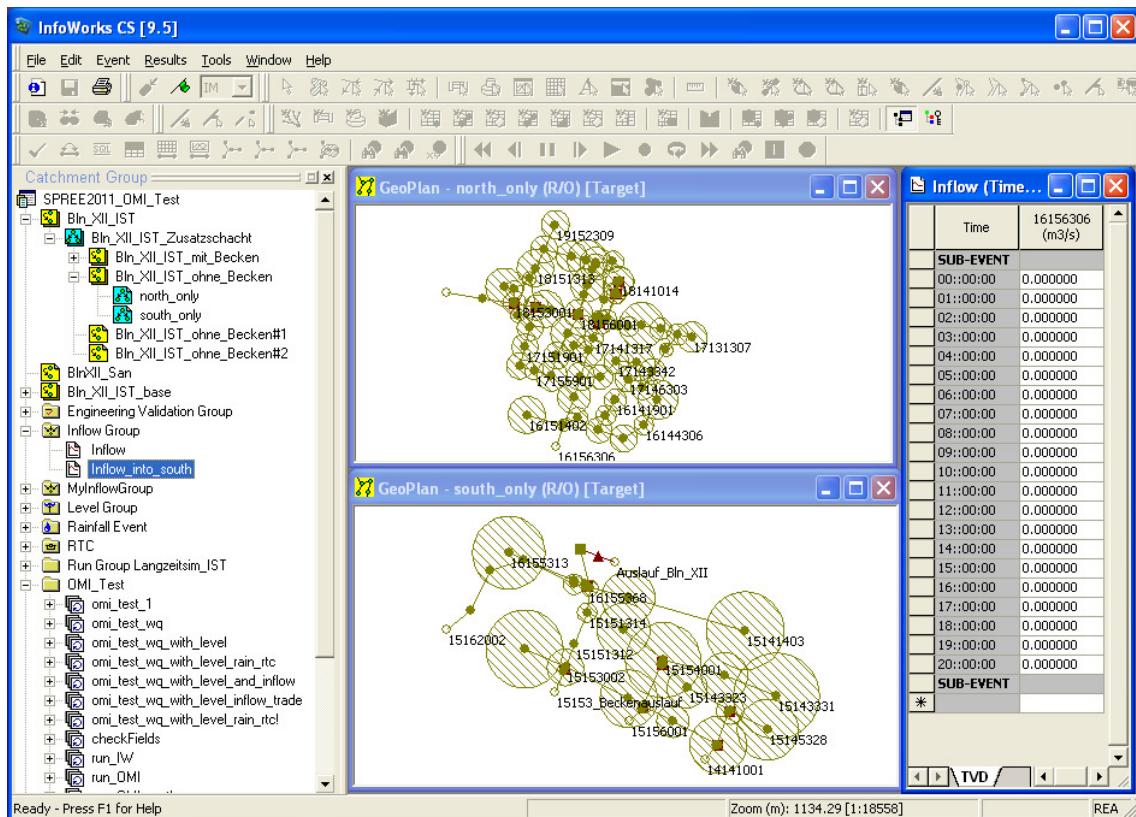


Figure 10: InfoWorks CS model network representing a Berlin catchment, divided into a northern (upper network window) and a southern sub-network (lower network window).

In the Configuration Editor, the exported OpenMI components representing the northern and southern InfoWorks CS sub-models, respectively, were linked as shown in Figure 11. The connection between the two components has been configured in the

desired way (see Figure 12). A Trigger component was connected to the south component.

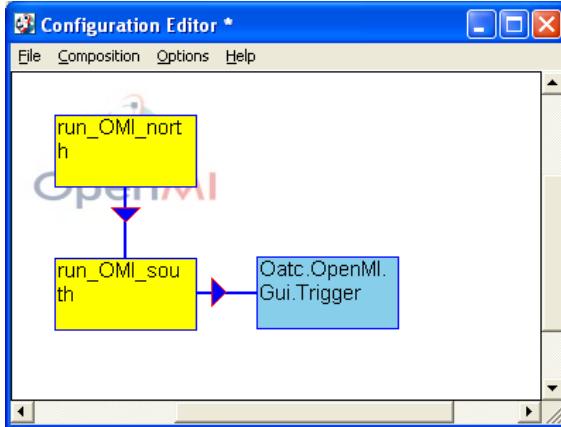


Figure 11: OpenMI-linked system of two OpenMI components of the same type (InfoWorks CS) in the OATC Configuration Editor. The component ‘run\_OMI\_north’ provides inflow into the component ‘run\_OMI\_south’. The latter component is triggered.

The configured OpenMI system was run with the trigger being invoked at the end of the simulation time horizon, being the same for both models (see above). The simulation performed successfully. The simulation results have been analyzed within the InfoWorks CS GUI. For the associated figures, see Appendix G. First, the outflows leaving the northern sub-network have been compared to the inflows that, through OpenMI, were transferred to the southern sub-network (see Figure 27 in Appendix G).

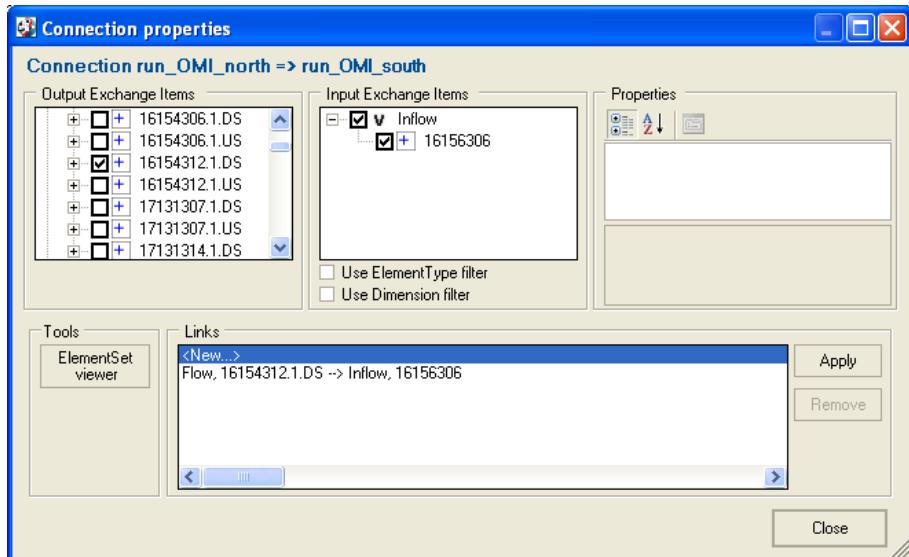


Figure 12: Dialog box within the OATC Configuration Editor showing the properties of the connection between the components ‘run\_OMI\_north’ and ‘run\_OMI\_south’. Flow in the downstream (DS) end of conduit ‘16154312.1’ in the northern sub-network is used as inflow to node ‘16156306’ in the southern sub-network (cf. Figure 11).

As expected, it was observed that the hydrographs of both, outflows from the north and inflows to the south, matched perfectly. This means that data transfer through the OpenMI connection between the northern and the southern sub-model was carried out correctly.

Next, it was investigated if the simulation results obtained by the OpenMI-linked system (case ‘OPENMI’) were the same as the results that were calculated for the original, entire InfoWorks CS network (case ‘ORIG’). Therefore, the overflows that are released into the receiving river at one outlet of the southern sub-network have been compared for the two cases (see Figure 13).

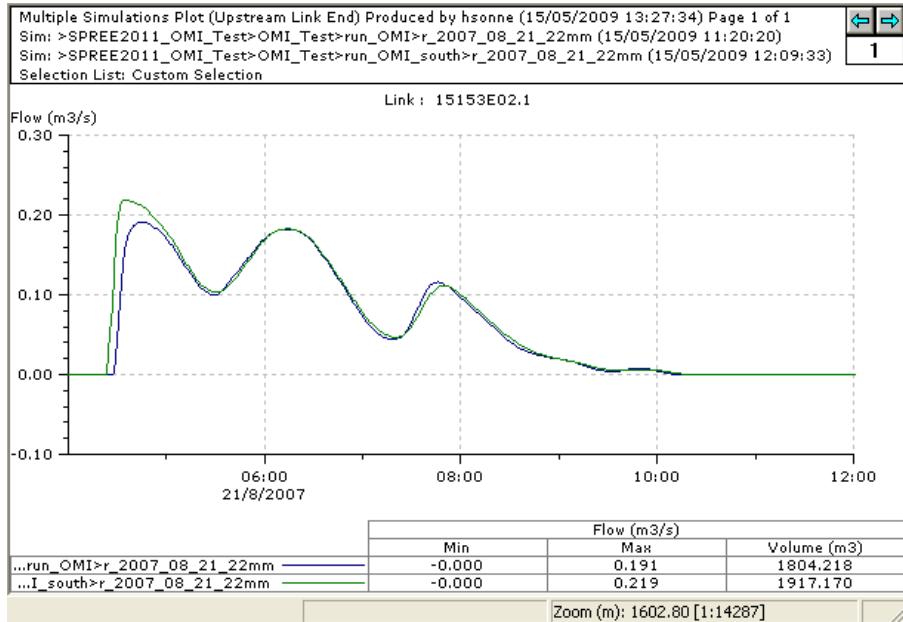


Figure 13: Overflow hydrographs simulated in InfoWorks for the entire network (blue line) and simulated in the OPENMI scenario without feedback between southern and northern sub-network (green line).

At the outlet to the river and also in the last sewer sections upstream, deviations between the overflow hydrographs could be observed (see Appendix G, Figure 28). Tracking the sewer system further upstream (see Appendix G, Figure 29) shows that the deviation at the end of the sewer system results from a superposition of even slighter deviations in the upstream sewer sections. The flow deviations seem to accumulate along the sewer.

The differences could result from backwater effects at the connection between the two sub-networks. These are not taken into account in the OPENMI scenario as there, only a unidirectional link between the northern and the southern model is established. As the OpenMI concept allows for bi-directional links, a backward link from the southern to the northern model was added to the OPENMI scenario. Via that link, the level at the entry node of the southern model is fed back into the outlet node of the northern model. In order to make the required input exchange item for levels available in the OpenMI component of the northern model, it was necessary to export a new OpenMI component from an accordingly configured hydraulic run in InfoWorks (cf. 5.2.2). Running the modified OPENMI scenario resulted in a different hydrograph at the outlet of the southern sub-network, which, again, was compared to the ORIG hydrograph produced by InfoWorks for the entire network (Figure 14).

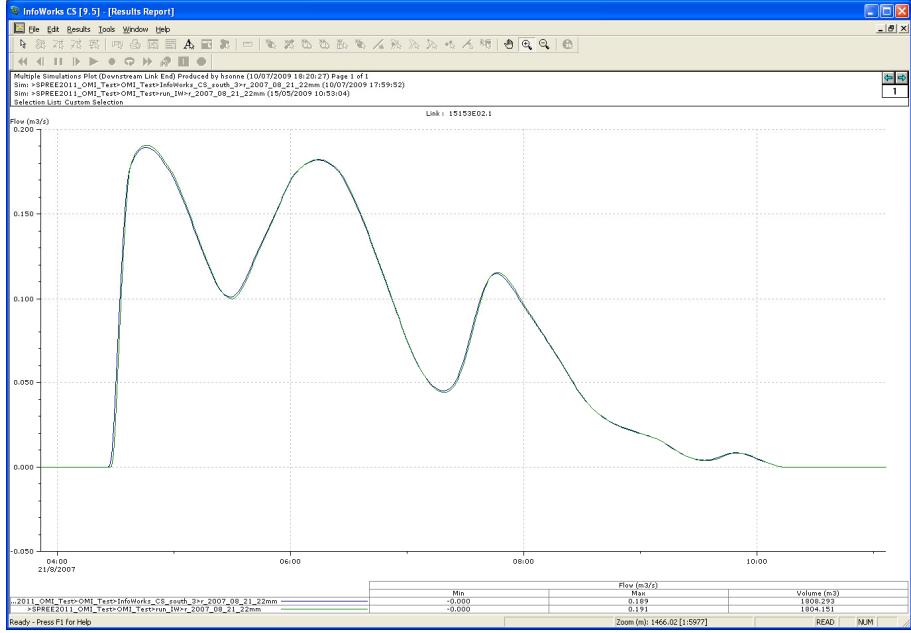


Figure 14: Overflow hydrographs simulated in InfoWorks for the entire network (case ‘ORIG’) and simulated in the ‘OPENMI’ case with feedback of water levels between southern and northern sub-network. The hydrographs match almost perfectly.

The remaining very small differences of calculated flows (hardly detectable in Figure 14) could result from different timesteps that are calculated within the ORIG simulation compared to the OPENMI simulations, in which the simulated networks are smaller. The more complex a network is, and the more elements (nodes, links, etc.) it comprises, the more probable it is that somewhere in the network the approximation procedure that is applied in order to solve the St. Venant equations does not deliver satisfying results. In that case, the timestep is decreased in order to achieve a better approximation. It is assumed that in the more complex network, more intermediate timesteps are calculated, which finally leads to a higher accuracy in comparison to a less complex network.

## 5.5 Scenario C: Linking InfoWorks CS to other OpenMI components

Only few OpenMI-compliant components were found that are freely available on the internet. One of these components is the CUAHSI-HIS component (see <http://his.cuahsi.org/hydrolink.html>), provided by the Consortium of Universities for the Advancement of Hydrologic Science (CUAHSI; <http://www.cuahsi.org>).

An OpenMI system was tested in which that component is linked to an InfoWorks CS component.

The CUAHSI-HIS component does not represent a model in the classical sense. It acts as a database, giving access to hydrologic time series, which are generated by the WaterOneFlow web services, hosted by the CUAHSI. The time series are stored in a specific format (WaterML format), which is a type of XML (Extensible Markup Language) format. The CUAHSI-HIS component complies to the OpenMI 1.4 standard (cf. 5.6.1).

In the test scenario, an example WaterML file was used (downloaded from <http://code.google.com/p/cuahsi-his-openmi-component/downloads/list>). Based on this file, the CUAHSI-HIS component offers exactly one exchange item for flow (‘flow\_exchange\_item’, see Figure 15). According to Figure 16 and Figure 17, the CUAHSI-HIS component was linked to the InfoWorks CS component which was

already used in Scenario B (see 5.4), where it represented the southern sub-system of a sewer network. The InfoWorks CS component was triggered. In that way, the OpenMI configuration should have performed as follows: The trigger requests the flow through any conduit in the InfoWorks component, which, in turn, requests the direct inflows into its boundary node from the CUAHSI-HIS component. The CUAHSI-HIS component reads the requested flow data from the WaterML file.

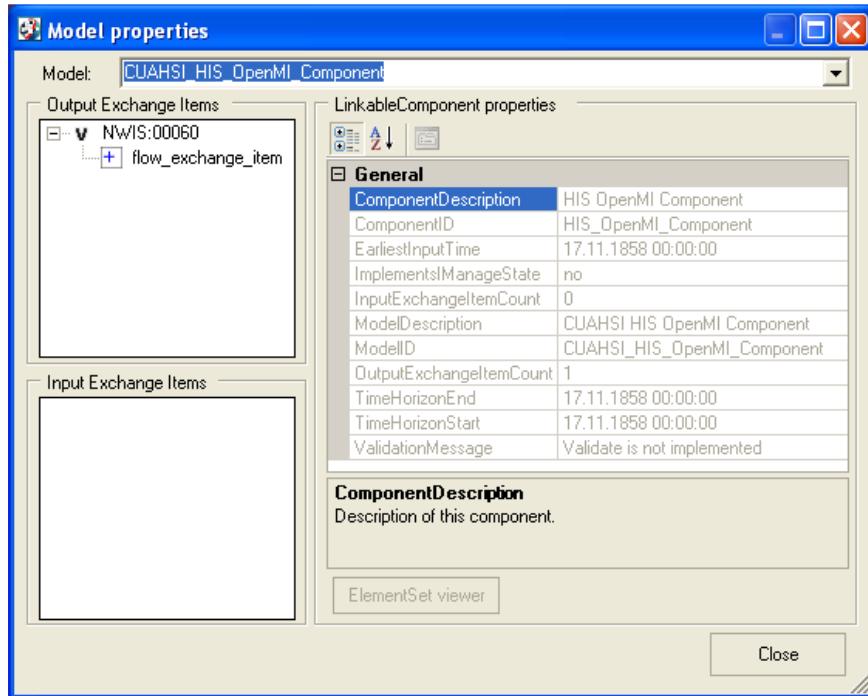


Figure 15: Model properties of the CUAHSI-HIS component, shown in the OATC Configuration Editor. The component offers the ‘flow\_exchange\_item’ being the only output exchange item.

When trying to run the OpenMI system as configured, the warning ‘Model time horizons don’t overlap’ is issued. The reason for this is that the CUAHSI-HIS component does not expose the correct time horizon that is defined in the actual WaterML file. Instead, the component claims the date ‘17/11/1958’ to be both, the start and the end of its time horizon (see Figure 15). In OpenMI, dates are represented as floating point numbers, where zero represents the above date (‘17/11/1958’). Although the OpenMI system could be started in spite of the warning about non-overlapping time horizons, the simulation failed. It is possible that the failure was due to the component’s inability to interpret dates correctly. Possibly, the reason for this lies in different regional settings (e.g. English versus German) that different programs may account for or not (cf. 5.6.1).

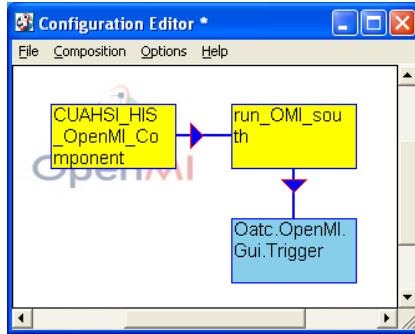


Figure 16: OpenMI configuration with a CUAHSI-HIS component providing inflow to an InfoWorks CS component in the OATC Configuration Editor.

The main conclusion that can be drawn from the scenario considered here is that, to make OpenMI work, it is not sufficient for the model components to comply to the OpenMI standard. Rather, it is essential that the methods, that the OpenMI components are required to offer, are implemented by the model components in the correct manner.

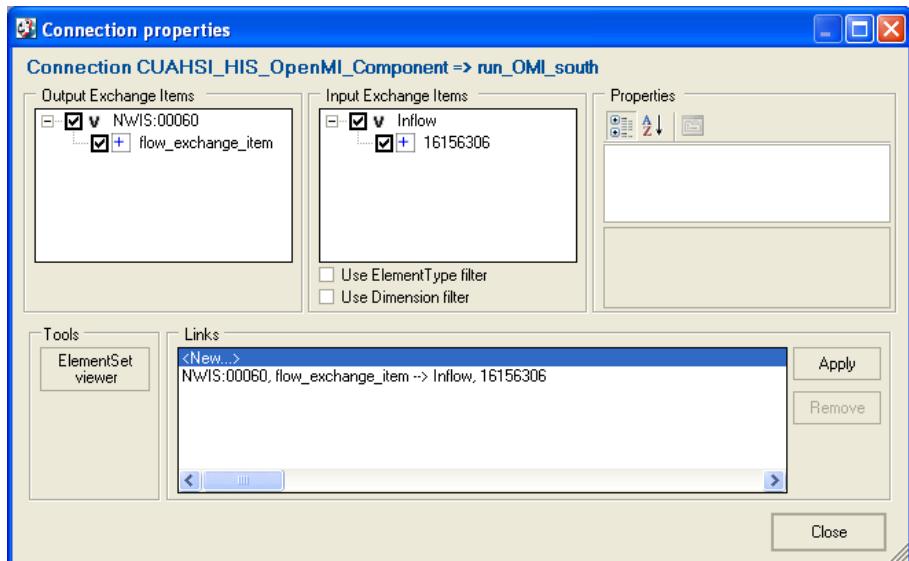


Figure 17: Properties of the connection between CUAHSI-HIS and InfoWorks CS component in the OpenMI configuration shown in Figure 16.

The OpenMI Association Technical Committee (OATC) provides simple demo models which may be used in order to create further simple test scenarios. Appendix H contains an overview of these models which represent different environmental domains (e.g. river, groundwater).

Concerning the coupling between InfoWorks CS and InfoWorks RS (the river modelling module of InfoWorks), already many use cases are reported. An extensive overview is given in the KWB report “Literature Review on the OpenMI” (Sonnenberg, 2008).

## 5.6 Results

Within this section, errors due to compatibility reasons, which have been occurred during the tests, are reported and appropriate solutions are given (see 5.6.1). The limitations of the InfoWorks’ OpenMI implementation, which have been encountered during the tests or which are reported in literature, are summarized in 5.6.2.

### 5.6.1 Compatibility issues

Generally, when working with OpenMI components, it is important to take care of possibly different supported versions of the OpenMI standard. The loading of a model component (the corresponding .omi-file) into the OpenMI Configuration Editor fails if model and editor support different versions of the OpenMI standard. In that case, the Configuration Editor will throw an exception or get blocked and not respond any more.

The current version of the OpenMI Configuration Editor (1.4.0.0) is based on the OpenMI specification in version 1.4. InfoWorks 8.5, which was first used, supports the OpenMI standard in version 1.0 (see Table 1). There were two possibilities to solve the compatibility problem:

1. Use a former version of the Configuration Editor (OmiEd 1.0).
2. Use a newer version of InfoWorks (at least 9.5) for creating OpenMI components that comply with the OATC Configuration Editor 1.4.

Within this study, the second solution was chosen. From testing different program versions of OpenMI-compliant models with different versions of the OpenMI Configuration Editor, the compatibilities given in Table 1 were found.

Table 1: Versions of the OpenMI standard that are supported/required by some tested OpenMI-compliant model engines.

OpenMI Model Engine	Supported/Required OpenMI version
InfoWorks CS 8.5	1.0
InfoWorks CS 9.5	1.4
CUAHSI-HIS	1.4

There seem to be compatibility problems concerning different regional cultures (English versus German). As shown in Figure 18, the ‘Model properties’ dialog of the loaded InfoWorks component displayed the Dimensions property in the ‘Quantity properties’ section incorrectly (dimension of length = -3000000, dimension of mass = 100000). Changing Windows’ regional settings from German to English (Great Britain), led to correct dimensions (dimension of length = -3, dimension of mass = 1).

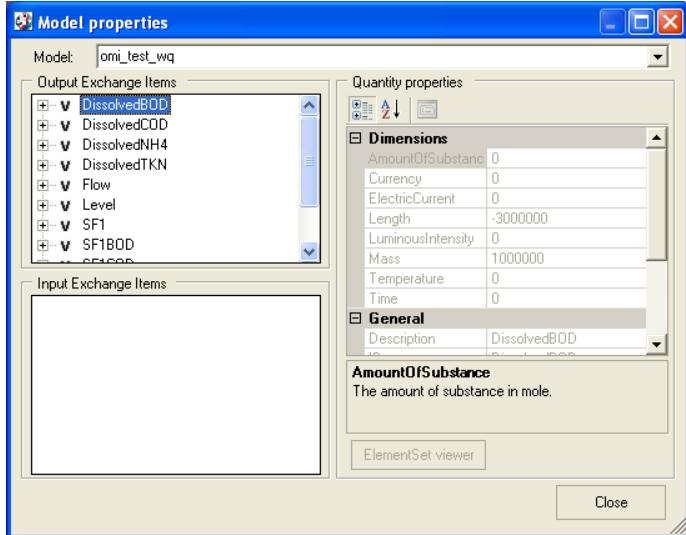


Figure 18: Model properties of the InfoWorks CS component as they are shown in the OpenMI Configuration Editor. Dimensions are incorrect due to incompatibilities of regional cultures. Input Exchange Items are not available as no Inflows and no Levels were configured in the corresponding scheduled run (see 5.2.2).

### 5.6.2 Limitations of the OpenMI implementation in InfoWorks CS

It depends on the program version of InfoWorks which version of the OpenMI standard is supported by the corresponding OpenMI-compliant InfoWorks components. InfoWorks 8.5 supports OpenMI 1.0, whereas since InfoWorks 9.5 the OpenMI standard is supported in its version 1.4 (current version in June 2009).

The InfoWorks OpenMI implementation does not give full access to the InfoWorks model. Only the variables flow and water level are possible input items of the model component (see 5.2.2). For example, it is neither intended to provide rainfall data nor e.g. Waste Water or Trade Waste specifications via the OpenMI to an InfoWorks CS component.

In particular, the InfoWorks implementation of the OpenMI standard does not give access to parameters of the rainfall runoff and sewer model (e.g. runoff coefficient, runoff routing factor, etc.). The modification of model parameters would be needed for model calibration purposes or sensitivity and uncertainty analysis. In a personal contact, the Wallingford support team recommends the use of the InfoWorks API (see Chapter 7) as long as the current OpenMI implementation of InfoWorks does not support access to model parameters via the OpenMI. Wallingford is open for a discussion on future improvements of the OpenMI implementation of InfoWorks (personal communication with Wallingford Software Ltd.).

The following further OpenMI-related concerns about InfoWorks (CS and RS) are reported in literature (openmi-life.org, 2007a; b):

- Increasing the timestep to speed up simulations of long dry weather periods (up to 32 times the normal timestep) is at present not yet supported by the OpenMI implementation of InfoWorks (openmi-life.org, 2007a).
- For the flood exchange between InfoWorks CS and InfoWorks RS (flood volume at CS nodes as RS Q,t boundary at RS boundary node), it may be necessary to make a few adaptations to both CS or RS to allow all the necessary transfers.

- Special attention must be given to the exchange of flood parameters, as this might not be all possible in the current version of InfoWorks.
- Another possible gap is the possibility of having both open and predefined boundaries in one and the same simulation. According to Wallingford Software this should work.
- The OpenMI Configuration Editor does not always accept the .omi-files produced by InfoWorks RS.

# Chapter 6: Applicability of the OpenMI: Making existing models OpenMI-compliant

## 6.1 Model Migration

The process of turning a model engine into an OpenMI-compliant component is also referred to as ‘migration’. In order to support model developers in migrating their model engines, the OpenMI Association disseminates a *Software Development Kit* (SDK). The SDK provides software utilities that can be used to facilitate the migration process.

Referring to the structure of Book 4 in the OpenMI documentation (Gijsbers et al., 2007a), the following sections describe the requirements for an OpenMI component (6.1.1), introduce the recommended software design (6.1.2) and give step-by-step instructions for the migration process (6.1.3).

### 6.1.1 Requirements for an OpenMI-compliant component

There are three basic requirements that an OpenMI-compliant component has to meet (Gijsbers et al., 2007a):

1. The component must implement the *ILinkableComponent* interface, forcing the component to provide
  - attributes that allow to identify the component (id and description),
  - attributes and methods that give access to the input and output exchange items,
  - methods that allow to add or remove links to or from the component,
  - methods that allow to perform specific model actions before and after a model run (e.g. validation, preparation, finishing, disposal),
  - a method *GetValues* that returns values for a given time at a given location,
  - methods that enable the component to register ‘event-listeners’, to which specific events will be sent. An example for an event is the *DataChanged* event, which is ‘thrown’ by a component after having answered a *GetValues* call from another linked component. Events allow ‘event listeners’ to monitor the progress of the linked system when running.
2. The finally developed component must be provided together with an XML-file (XML = *Extensible Markup Language*) that, in a pre-defined manner, contains information about where to find the component on the computer and about how to load it. That file should have the extension ‘.omi’ and is also referred to as ‘.omi-file’.
3. The component must be able to invoke the methods mentioned first in this list in the following sequence of specific phases:
  - initialization,
  - inspection and configuration,
  - preparation,
  - computation/execution,
  - completion,

- disclosure.

Before migrating a model to an OpenMI-compliant component, it is important to have a precise idea of how the model will be used as an OpenMI component. Any situation, in which the component will be linked to other components (such as other models but also data providers, optimization or calibration tools), should be kept in mind.

It is recommended to define use cases, which describe the functionality, that the component must provide, in detail. The use cases shall give a step-by-step description of how the models will be used. Most important when defining use cases is that after completion of software development, it must be unambiguously possible to determine whether the use case is covered or not.

Based on the use cases, input and output exchange items that the model shall expose, need to be defined. An exchange item defines *what* (which quantity, e.g. flow) can be exchanged *where* (at which elements, e.g. nodes, river branches).

### 6.1.2 Wrapping concept and wrapping pattern provided in the OpenMI SDK

The OpenMI standard and the SDK are provided in the form of source code written in the programming language C# (read: ‘C sharp’), which is based on the Windows .NET (read: ‘Dot Net’) framework (<http://msdn.microsoft.com/en-gb/netframework>). Thus, model migration is best supported when the same environment is used for the development of model components. However, most existing model engines are written in other programming languages, such as Fortran, Pascal, C or C++ (read: ‘C plus plus’). In order to bridge the gap between the different technologies and to minimize the number of changes needed to be applied to the engine core, a *wrapping* procedure is recommended.

*Wrapping* basically means to create a C# class that implements the *ILinkableComponent* interface (see 6.1.1). In Figure 3 and in the following figures, the implementation of this interface is indicated by a blue ellipsis. The wrapper class can be seen as a ‘black box’, in which internally the model engine is accessed but which prevents the user from seeing how that happens. All communication with the OpenMI component will take place through the interface, that is, by accessing the specific attributes and methods. Internally, the wrapper class will communicate with the engine core. If the engine is written in another programming language, the wrapper class needs to know how to call the software code in that language and it is responsible for doing these calls. Within the wrapping process the developer needs to ‘wire’ the OpenMI interface to the engine core. This is mainly done by translating method calls that come through the interface into adequate calls of the specific functions provided by the model engine.

An advantage of the wrapping concept is that OpenMI-specific code can be implemented in the wrapper class without the need to change the model core. So, the same model core can still be used in its former, non-OpenMI context, e.g. within a stand-alone application programmed in its own language.

A default implementation of a wrapper class is provided in the OpenMI SDK. Moreover, the OpenMI guidelines describe a software *pattern* that is recommended to be used in order to realize the wrapping (Gijsbers et al., 2007a). Software patterns generally describe the structure of some part of software, including the participating software modules (classes), their responsibilities, dependencies and interactions. The wrapping pattern applied in the OpenMI SDK consists of different C# classes (see Figure 19).

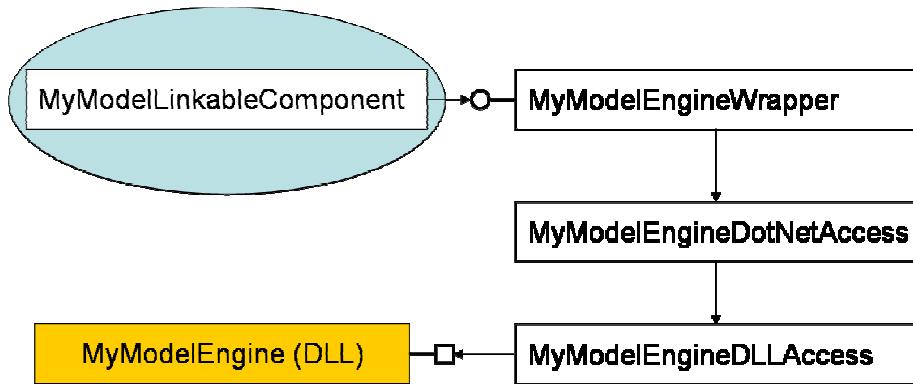


Figure 19: Participating elements of the wrapping pattern (orange rectangle: dynamic link library; white rectangles: C# classes; blue ellipse: *ILinkableComponent* interface; round ‘lollipop’ symbol: *IEngine* interface).

It is assumed that an existing model engine *MyModelEngine*, written in Fortran, is available in the form of a Dynamic Link Library (DLL). In order to connect the engine to the OpenMI wrapper class *MyModelLinkableComponent*, which implements the *ILinkableComponent* interface (blue ellipsis) and thus represents the final OpenMI-component, the design envisions three intermediate C# classes. The classes may be seen as a set of stacked layers, which in each case use methods of the underlying bottom layer and provide methods on a higher level of abstraction to the top layer:

- The first class *MyModelEngineDLLAccess* ‘knows’ how to access the functions in the DLL via a Windows-specific interface (square ‘lollipop’ symbol in Figure 19) and makes these functions available in the form of C# class methods.
- The second class *MyModelEngineDotNetAccess* uses the methods of the first class. It performs an adaptation between calling conventions and exception handling, which are different in Fortran and in C#. This class allows to use the model engine functions indirectly by calling C# methods in a C# manner.
- The first two classes are specific to the model engine, i.e. they are not expected to offer any particular, standardized method. This, however, is done by the third layer class *MyModelEngineWrapper*, which at the bottom uses the methods of the second class and on top (or rather left in Figure 19) provides a standardized interface (round ‘lollipop’ symbol), namely the *IEngine* interface. The actual wrapper class *MyModelLinkableComponent* finally accesses the model engine through that last mentioned interface.

The OpenMI standard puts a lot of responsibilities on OpenMI components. An OpenMI component must always be able to deliver values if its *GetValues* method is called. The call may refer to a point in time which is out of the current scope of the component’s simulation progress. It may also refer to a location which does not correspond to the own inner spatial representation. Therefore, the wrapper class needs to offer the following features, of which basic implementations exist in the base wrapper class provided in the OpenMI SDK (Gijsbers et al., 2007a):

- Buffering,
- Temporal interpolation and extrapolation,
- Spatial operations,

- Book-keeping of links from and to the component,
- Event-handling.

### 6.1.3 Model migration in seven steps

The OpenMI documentation series describes the model migration process as a seven-step procedure, which, in brief, is given here. As Figure 20 illustrates, each step (number in yellow circle) corresponds to one or more elements of the wrapping pattern described in the last section. Exemplarily, the documentation contains a description of how the step-by-step procedure was applied to a simple model, the ‘Simple River’ model (Gijsbers et al., 2007a).

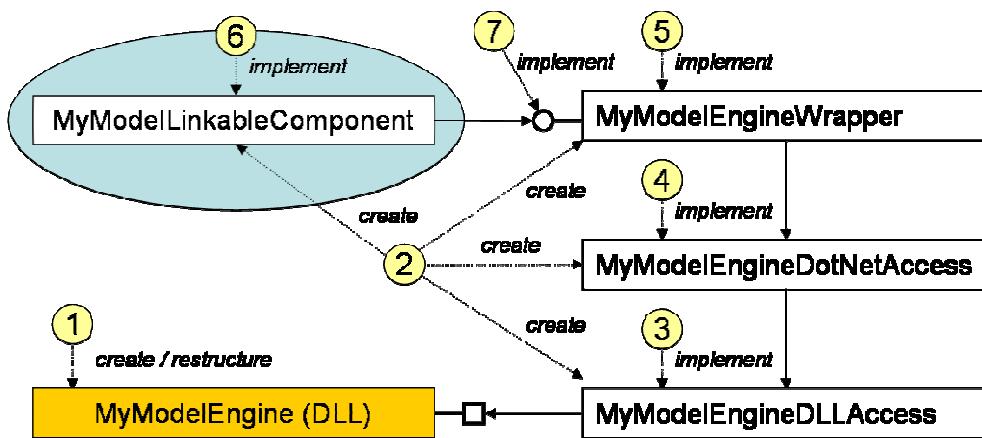


Figure 20: Overview of steps that are performed in order to apply the wrapping pattern.

#### Step 1: Changing the engine core

In a stand-alone model application, the model engine often exists in the form of an executable file (EXE). The internals of an EXE cannot be accessed from other software components as it is needed for OpenMI. For maintenance reasons it should be the aim that one and the same engine file can be used by both, the OpenMI component and the stand-alone model application. Therefore it is recommended to compile the engine into a Dynamic Link Library (DLL) instead of an EXE. The DLL allows for external access. The aforementioned aim can then be achieved by providing a DLL-function *RunSimulation* that performs the whole simulation process that beforehand was realized within the EXE. A new executable application can then be built that does no more than calling the *RunSimulation* function of the DLL. Figure 21 illustrates the situation described.

In order to make the SDK helper classes applicable, the DLL engine needs to be structured in such a way that it offers the following functions:

- Initialize (Open files and populate the engine with initial data),
- PerformTimeStep,
- Finish (Close files),
- Dispose (Free memory).

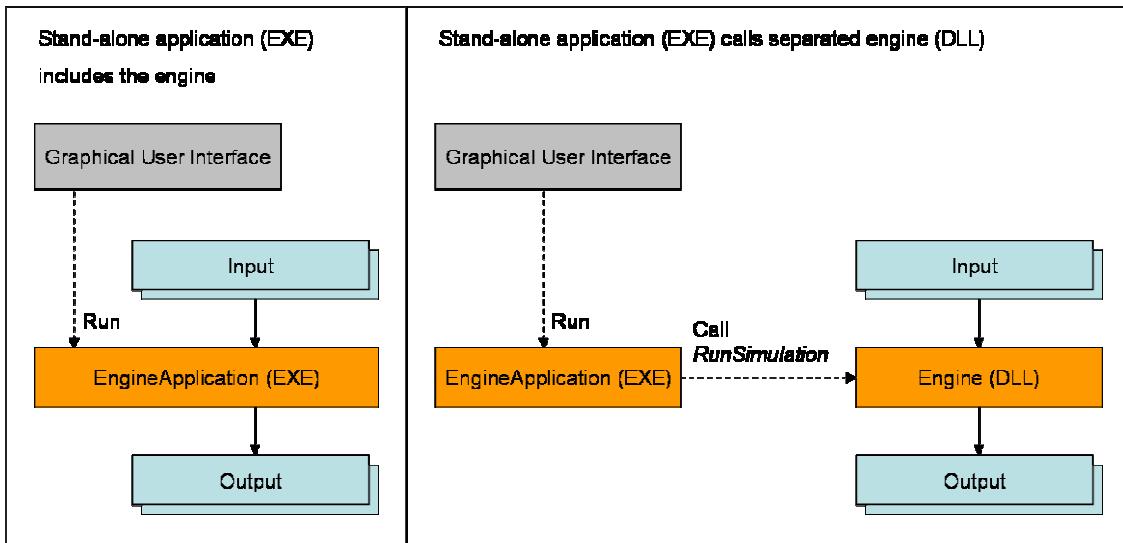


Figure 21: Separation of model engine (DLL) from executable stand-alone-application (EXE).

The RunSimulation function shall perform a whole simulation by calling the Initialize function, then repeatedly calling the PerformTimeStep function until the simulation has completed, and finally calling the Finish and Dispose functions. If the engine does not already follow this structure, it needs to be re-structured accordingly.

### Step 2: Creating the .NET assemblies

Having restructured the engine and having made the engine functions available as a DLL, the C# wrapper classes (see 6.1.2) can be created. The .NET environment and the OpenMI SDK need to be loaded (both freely available on the internet). Let the actual model to be migrated be 'MyModel'. Following the recommended naming convention, the classes to be created are (cf. Figure 19):

- MyModelEngineDLLAccess (to be implemented in Step 3),
- MyModelEngineDotNetAccess (to be implemented in Step 4),
- MyModelEngineWrapper (to be implemented in Step 5),
- MyModelLinkableComponent (to be implemented in Step 6).

The created classes may be seen as skeletons, which still need to be implemented, i.e. filled with functionality (see the following steps). The wrapper class MyModelLinkableComponent, which finally represents the OpenMI component, shall *inherit* from the predefined *LinkableEngine* class, which is part of the wrapper package in the SDK. By doing so, the wrapper gets a default implementation. In object oriented programming, inheritance refers to making properties and methods of a *parent* class available to an inheriting *child* class.

All created classes shall be compiled into one .NET assembly. An Assembly is the .NET-specific kind of a software library. It is recommended to create corresponding test classes in a separate assembly.

### Step 3: Accessing the functions in the engine core

Within the third step, the MyModelDLLAccess class (created in step 2) is implemented. This class will make a one-to-one conversion of all exported functions in the engine core

to public .NET methods. The specific implementation depends on the programming language and the compiler that has been used to produce the DLL. The OpenMI document series gives an example for accessing a Fortran DLL (Gijsbers et al., 2007a).

#### *Step 4: Implementing MyModelEngineDotNetAccess*

Within the fourth step, the MyModelEngineDotNetAccess class (created in step 2) is implemented. This class has two purposes: to change the calling conventions to C# conventions and to change error messages into .NET exceptions:

- The different treatment of parameters and return values of functions (call-by-value versus call-by-reference) in C# and other languages like Fortran are adapted in this class. Other language-specific differences, e.g. those in array indexing, are handled here, too.
- In order to indicate if an error occurred, Fortran functions may return error codes. In this class, these codes may be transformed into corresponding .NET exceptions. An exception can be seen as an object that describes an error. The exception can be ‘thrown’ through an application until it reaches a part of the software which ‘feels responsible’ for that kind of error, ‘catches’ and handles it.

#### *Step 5: Implementing the MyModelEngineWrapper class*

Within the fifth step, the MyModelEngineWrapper class (created in step 2) is implemented. This class needs to implement the *IEngine* interface (see 6.1.2). It may be started by letting the C# programming environment automatically generate stubs of the needed interface methods. The method bodies have then to be filled with code (to be completed in step 7). The first step is to implement the *Initialize* and the *Finish* method. At initialization, an instance of the MyModelEngineDotNetAccess class has to be created. Requests coming through the *IEngine* interface (e.g. the request of performing one timestep) can then be redirected to the engine by calling the corresponding methods of that instance of the MyModelEngineDotNetAccess class.

#### *Step 6: Implementing MyModelLinkableComponent*

Within the sixth step, the MyModelLinkableComponent class (created in step 2) is implemented. An object of this class represents the actual OpenMI-compliant component that is going to be accessed by other models (cf. 6.1.2). The implementation of this class is simple. The class creates an instance of the MyModelEngineWrapper class and stores a reference to it. Then, the default implementation of the MyModelLinkableComponent class, which is inherited from the SDK class *LinkableEngine*, becomes operational. That implementation uses the stored reference in order to access the engine wrapper (instance of MyModelEngineWrapper) through the *IEngine* interface.

#### *Step 7: Implementation of the remaining IEngine methods*

After the first six steps, the basic structure of engine and wrapper code is in place. The remaining task is to complete the implementation of the MyModelEngineWrapper class, created in step 2. In step 5, stub code has been auto-generated for the methods required by the *IEngine* interface that this class needs to implement. Here, the bodies of the methods have to be filled with code. The *IEngine* methods to be implemented are given in Table 5 in Appendix B.

Some of the methods can be completed by only changing the code in this class. The code will access the engine core by calling methods of the MyModelEngineDotNetAccess class. For some methods, however, changes will need to be made to the other classes or to the engine DLL. For each method it must be decided if the bulk of implementation should be located in the MyModelEngineWrapper class or in the engine core. There is no general answer to this question. Advantages and disadvantages are given in (Gijsbers et al., 2007a).

## 6.2 Estimation of Effort

Generally, the effort needed to migrate an existing modelling software into a corresponding OpenMI-compliant component depends on

- the tasks to be performed by the OpenMI component,
- personal preconditions and
- technical preconditions.

After having discussed these factors generally in sections 6.2.1 through 6.2.3, the effort related to the different steps of the migration process (cf. 6.1.3) is estimated in section 6.2.4. Finally, section 6.2.5 covers the possibly needed effort for migrating the models that are intended to be used within MIA-CSO.

### 6.2.1 Tasks to be performed by the OpenMI component

As mentioned in section 6.1.1, the scenarios, in which the model to be migrated is planned to be used, can be described in the form of well defined use cases. These use cases determine the types of models that the component shall be linked to and describe how exactly the model component shall be used in a linked system. A good approach could be to define a use case, which is able to perform exactly the tasks that can currently already be performed by manual, sequential linking of models.

The effort of model migration then definitely depends on the complexity of the defined use cases. The wrapper around the model engine needs to implement the *GetValues* method (see 6.1.1). That method must be able to return ‘best guessed’ values for requests that refer to times or locations that are out of the current temporal or spatial scope of the model. Although default implementations are given, additionally data operations for the use case-specific temporal or spatial mappings may become necessary.

For an efficient use of resources, the use cases shall be defined as simple as possible and only as complex as necessary to achieve the objectives of the project.

### 6.2.2 Personal preconditions

Migrating a model application is mainly a software engineering task. The effort needed for model migration highly depends on the available personal experience with programming in general and with the model software to be migrated in particular. It is highly recommended that migration is done by a person who profoundly knows the software code.

A good understanding of object oriented programming and of the interface concept seems to be indispensable. Experiences in the programming language C# are advantageous. Some time is needed to understand the concept and the structure of

OpenMI; it is important that the interfaces defined in the standard and the relationships between the classes that are contained in the SDK are understood in detail.

### 6.2.3 Technical preconditions

For model migration, the source code of the modelling software must be accessible. This is the case for freely available modelling softwares. Commercial modelling softwares can only be made OpenMI-compliant by the corresponding software companies.

Having access to the source code, a good understanding of the software to be migrated is required. A detailed documentation of the software should be available. At least, the software should be documented in terms of comments in the source code, which allow for an understanding of the code.

Furthermore, the effort needed to implement the OpenMI interface strongly depends on the structure of the existing software and on the software technologies (programming language) used.

A main precondition concerning the software structure is that the model engine to be migrated is separated from any graphical user interfaces and that there is a clear distinction between input, processing and output of data.

Concerning the programming language, a model engine that is written in Fortran will be relatively more difficult to migrate than a model engine that is written in C or C++. The last-mentioned languages are of the same family as C#, which is the recommended language for model migration. Being object oriented, C++ programs will be easier to migrate than C programs.

### 6.2.4 Effort related to the different steps of model migration

This section refers to the step-by-step procedure described in section 6.1.3.

Provided that access to the source code and to the programming environment of the model engine is given, creating a DLL instead of an EXE in *step 1* should not cause a problem. However, restructuring the engine core is assumed to be possibly a complex task. Providing a function that performs a single timestep and giving access to the current values of all needed model variables after each timestep may require profound structural changes.

The creation of C# classes in *step 2* is a formal task. Support is given by the software development environment. Class inheritance prevents from reimplementing functionality that already exists.

Time and effort needed to implement the classes that give access to the engine DLL in *step 3* and *step 4* depend on the programming language in which the engine is written. Concerning these steps, the OpenMI documentation contains examples, in which a Fortran DLL is accessed. Implementing the MyModelDLLAccess class should be easy as that class only performs one-to-one translations of procedure calls. Also the implementation of the MyModelEngineDotNetAccess class can be seen as a formal task. Following the given example, no further structural or logical considerations are needed.

*Step 5* and *step 6* are easy to perform as the programming environment gives support by auto-generating code. Implementation of the *IEngine* interface is completed in *step 7*.

The implementation of the *IEngine* interface in *step 7* strongly depends on the structure of the engine core, so it is not possible to give a general explanation of how each individual method should be implemented. This step is assumed to be one of the most complex.

### 6.2.5 Effort related to the models intended to be used in MIA-CSO

In the last sections the effort of migrating model applications into OpenMI components has been discussed in terms of general aspects and also related to the different steps involved.

Concerning the OpenMI scenario presented in section 4.1, that possibly could be applied within MIA-CSO, it is difficult to estimate the effort that would be needed to migrate those model engines, which are currently not OpenMI-compliant, into corresponding OpenMI components. The source codes of the river models HYDRAX and QSim are not freely available. The author does not have information on the structure of these programs and therefore the steps needed to make HYDRAX and QSim OpenMI-compliant are unknown. In the following, only some very general assumptions are made.

HYDRAX is an executable application (EXE), written in the programming language C. It is assumed that the model engine could be easily provided in the form of a DLL. It should not cause much effort to access that DLL from within the C# programming language. Model migration will mainly depend on how well the structure of the engine already meets the OpenMI requirements.

QSim is an executable application (EXE), written in the programming language Fortran. Again, it is assumed that the model engine could be easily provided in the form of a DLL. Compared to HYDRAX, providing C# wrapper classes that access the Fortran-DLL of QSim will be a more complex task. Furthermore, it is supposed that QSim is more complex than HYDRAX. Therefore, changing the structure of the software in order to meet the requirements of OpenMI is estimated to be more difficult than it is the case for HYDRAX.

An option could also be to make the GERRIS software, which already realizes the link between HYDRAX and QSim, OpenMI-compliant. GERRIS is an executable application (EXE), written in the programming language Visual Basic. It is assumed that the software could be easily transformed from Visual Basic to VB.NET, the corresponding language in the .NET environment. Provided that the GERRIS software could be compiled into a .NET assembly, it should be easy to access the GERRIS methods from within the C# wrapper classes that are provided in the OpenMI SDK. However, as GERRIS only runs complete simulations of HYDRAX and QSim models, and so is not able to perform a single simulation timestep, an OpenMI component of GERRIS could not be used in an OpenMI scenario that requires feedback between models.

# **Chapter 7: The InfoWorks Application Programming Interface (API)**

Some of the programming-specific terms used within this chapter, that are underlined at first usage, can be found in the glossary in Appendix A.

The InfoWorks software is shipped with a so called Type Library that defines a set of functions (methods) of the software that can be accessed from within other (e.g. self developed) software programs. The specification of methods including their names, return types and number, order and types of parameters that these methods accept is referred to as the Application Programming Interface (API; see 2.2.2).

This chapter describes how the InfoWorks Type Library/API has been tested. After having defined the objectives of the tests (see 7.1), the available material, the used environment and the applied testing strategy are described (see 7.2). Finally, section 7.3 summarizes the results that have been obtained by conducting the tests.

## **7.1 Objectives**

The objective of this sub-study was to find out if and to what extent the InfoWorks API can be used within the MIA-CSO project. It should be investigated how using InfoWorks' functionality through the API can facilitate and automate processes that, so far, have been performed manually. For example, automated model calibration as well as uncertainty and sensitivity analysis that require repeated simulation runs in a loop could be enabled. Therefore, the API should offer methods that allow to change model parameters, to run simulations and to export simulation results.

Using the API may be of particular interest as the first part of this study revealed that InfoWorks' current OpenMI implementation does not allow for the last-mentioned tasks. As shown in Chapter 5, an OpenMI component exported by InfoWorks only offers input exchange items for flow and level. Model parameters are not intended to be accessed via the OpenMI interface. So, model modification being a prerequisite for these more sophisticated tasks cannot be undertaken by access of InfoWorks through OpenMI.

The specific questions (to be answered in 7.3) are:

Question 1 (to be answered in 7.3.1): How does access to the InfoWorks Type Library through its API work technically? It is intended to be realized from within a C# program,

Question 2 (to be answered in 7.3.2): Which methods are offered by the API?

Question 3 (to be answered in 7.3.3): How do the methods work and do they allow for

- importing model and input data,
- changing model parameters,
- running simulations,
- exporting simulation results?

Question 4 (to be answered in 7.3.4): Which of the tasks to be dealt with in the MIA-CSO project can be performed by using the InfoWorks API, and which cannot?

## 7.2 Materials and Strategy

This section gives an overview of the available API documentation that was used (see 7.2.1), describes the programming environment in which the tests were performed (see 7.2.2) and covers the testing procedure (see 7.2.3).

### 7.2.1 InfoWorks API documentation

Starting point of the study was the review of an unpublished documentation about the InfoWorks API (WSL, 2007) that was received from the Wallingford Software support team. The structure of that document is shown in Table 2. Apart from this document, no further documentation describing the InfoWorks API was available.

Table 2: Structure of the API documentation provided by Wallingford Software Ltd. Terms in italics refer to names of methods and properties of the class that is defined by the InfoWorks API.

Chapter	Subchapter/ <i>Methods/Properties</i>
Prerequisites	
Terminology	
Basic Principles	Introduction Identifying objects in the database Identifying object types Getting started
Database Navigation	<i>Children, Parent, Exists</i>
Object Properties	<i>ShortDisplayName, DisplayName, Type, Value</i>
Object Creation	<i>NewObject, NewSim</i>
Running Simulations	<i>Run, WNRUn, RSRun</i>
Import	<i>Import</i>
Export	<i>Export, ExportResults, ExportResultsSnapshot, ExportWNResults, ExportRSResults</i>
Database Properties	<i>LocalRoot, LocalRootGUID</i>
Miscellaneous	<i>BrowseForObjects, GetNetwork, CompareNetworks, EngineeringValidation, CheckIn</i>
Examples	Creation of WS run and sim objects Creation of CS run and sim objects Creation of an RS run and sim object Exporting a results snapshot C# example Full worked example (Genetic Algorithm)
Appendix	A: Object types and codes B: Fields to get and set via value property C: Special cases for properties D: Export result codes

The document first introduces the used terminology and basic principles, then the different methods and properties of the InfoWorks class that the API provides, are explained.

The code examples, which are given then in the document, are primarily written in Visual Basic. The InfoWorks Type Library, however, can also be used from other languages like e.g. Visual C++, Visual Basic.NET and Visual C#.NET. A code example in the programming language C# (read: 'C sharp') is presented at the end of Wallingford's API documentation.

Finally, the appendices of the API documentation list specific codes that identify the different InfoWorks objects and that are used e.g. as method parameters.

### 7.2.2 Chosen programming language and environment

The computer platform used for this study was Microsoft Windows XP Professional Version 2002, Service Pack 3.

Assuming that the planned environment for model integration may access at least some models or other components through OpenMI, the preferred programming language to be used within MIA-CSO is C#. Not only is the OpenMI interface specification delivered in the form of C#-specific interface declarations, but also the Software Development Kit (SDK) being part of the OpenMI dissemination is available as C# source code.

Thus, apart from reproducing the VBA-examples given in the documentation, the API methods were tested in a C# environment. For C# software development, the Integrated Development Environment (IDE) SharpDevelop was used. The software SharpDevelop in the used version 2.2 requires the Microsoft .NET Framework in version 2.0 to be installed on the computer.

The .NET Framework Software Development Kit (SDK) was also installed. From the SDK, the Type Library Importer ‘tlbimp.exe’ (see step 1 in Appendix D) and the .NET Framework IL Dissassembler ‘ildasm.exe’ (see 7.3.2) were used.

SharpDevelop supports the use of the version control system SubVersion. That program and the corresponding client program TortoiseSVN, which supplies a graphical interface to SubVersion, were installed separately. The version control system is used to keep track of changes in the software source code.

All mentioned programs are free for download (see Table 3).

The testing of API calls has been performed from within a Windows console application project that has been created in SharpDevelop. A class has been written that encapsulates the calls of the API methods. By doing so, exceptions that may occur when calling the API can be caught and handled. Further classes have been developed that use the methods of this first class in order to perform more complex tasks (see 7.2.3).

Table 3: Free download locations for the used programs.

Program	Download location
Microsoft .NET Framework 2.0	<a href="http://www.microsoft.com/downloads/details.aspx?FamilyID=0856eacb-4362-4b0d-8edd-aab15c5e04f5">http://www.microsoft.com/downloads/details.aspx?FamilyID=0856eacb-4362-4b0d-8edd-aab15c5e04f5</a>
Microsoft .NET Framework 2.0 SDK	<a href="http://www.microsoft.com/downloads/details.aspx?FamilyID=fe6f2099-b7b4-4f47-a244-c96d69c35dec">http://www.microsoft.com/downloads/details.aspx?FamilyID=fe6f2099-b7b4-4f47-a244-c96d69c35dec</a>
SharpDevelop 2.2	<a href="http://www.icsharpcode.net/OpenSource/SD/Download">http://www.icsharpcode.net/OpenSource/SD/Download</a>
SubVersion	<a href="http://subversion.tigris.org/getting.html">http://subversion.tigris.org/getting.html</a>
TortoiseSVN	<a href="http://tortoisessvn.net/downloads">http://tortoisessvn.net/downloads</a>

### 7.2.3 Testing procedure

Wallingford's API documentation focuses on accessing the InfoWorks Type Library from Visual Basic (VB) programs (see 7.2.1). As for MIA-CSO, however, C# is the preferred programming language, the first task was to enable access to the library from within C# programs.

Having realized the access to the library allowed to inspect the library and to find out about additional features that are not reported in Wallingford's API documentation.

A set of relevant methods and properties has then been tested. For these tests, existing InfoWorks CS models could be used.

Having gained first experiences with the properties and methods that can be accessed through the InfoWorks API, it was tested how method calls can be combined in order to perform more complex tasks. Therefore, a simple use case has been defined. Based on an existing InfoWorks sewer network containing a stormwater tank it was tested if API can be used to solve a simple optimization problem. The stormwater tank holding back combined sewer overflows should be designed in such a way that, for a given rain event, a certain overflow volume goes below a given total volume.

## 7.3 Results

Section 7.3.1 describes how the InfoWorks Type Library could be accessed from either within a VBA or from within a C# program. In section 7.3.2, the content of the InfoWorks library is investigated and a list of all available methods is given. Section 7.3.3 concentrates on specific tasks that are performed by individual API methods and reports on particularities of their usage. Finally, section 7.3.4 describes how a simple optimization problem could be solved by use of API calls.

### 7.3.1 Accessing the InfoWorks Type Library

When installing InfoWorks 9.5 on a Windows computer, the software registers the 'Wallingford Software InfoWorks 17.0 Type Library' in the Windows Registry. The InfoWorks Type Library contains the InfoWorks API definition. Often, type libraries are contained in a dynamic link library (DLL). In the case of InfoWorks, however, the type library is embedded directly in the executable program file 'infoworks.exe', which, when started in the usual way, runs the InfoWorks application with its graphical user interface. Table 4 lists the relevant registry entries that can be found on a Windows computer if InfoWorks is installed.

Table 4: Values that are registered in the Windows Registry if InfoWorks is installed. File paths may differ due to different installation paths of InfoWorks on different computers.

Registry Key	Registered Value
HKEY_LOCAL_MACHINE\SOFTWARE\Classes\TypeLib\{7861BEBB-646E-43F3-B90B-47E8885BD7A3}\11.0	Wallingford Software InfoWorks 17.0 Type Library
HKEY_LOCAL_MACHINE\SOFTWARE\Classes\TypeLib\{7861BEBB-646E-43F3-B90B-47E8885BD7A3}\11.0\0\win32	C:\Programme\InfoWorks95\infoworks.exe

Once a library is registered, it can be referenced and accessed from other programs. An easy way to access software libraries becomes possible if Microsoft Office is installed on the computer. The Office applications (preferably Microsoft Excel) give access to the programming language Visual Basic for Applications (VBA), for which a programming environment is provided. Appendix C describes step-by-step how the InfoWorks Type Library can be accessed from within the VBA environment provided by Microsoft Excel.

Corresponding to step 3 of that procedure described for VBA, also in SharpDevelop (see 7.2.2), references to libraries can be added to a programming project. This is done by means of a dialog box (see Figure 23). The latter opens when choosing ‘Add Reference’ from the context menu (right mouse click) of the ‘References’ item in the ‘Projects’ view (see Figure 22).

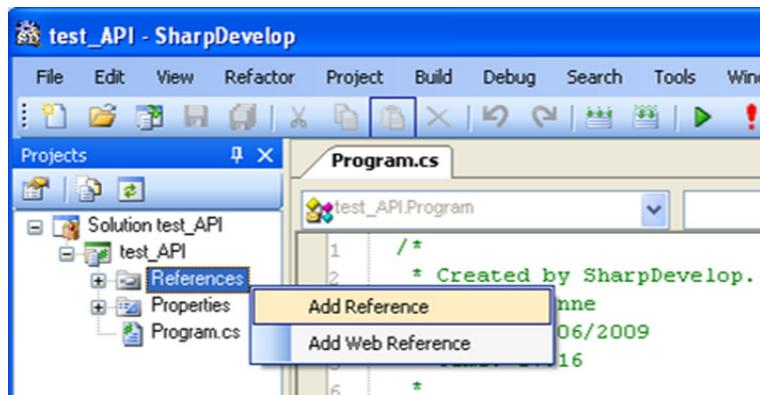


Figure 22: Opening the ‘Add Reference’ dialog in SharpDevelop.

The ‘COM’ tab of the ‘Add Reference’ dialog contains a list of all registered type libraries. If InfoWorks is installed, the list contains the entry ‘Wallingford Software InfoWorks 17.0 Type Library’ (or corresponding entries for different program versions; cf. step 3 in Appendix C). As indicated in the bottom area of the dialog box and as described above, the type library ‘InfoWorksLib’ is contained in the executable program file ‘infoworks.exe’.

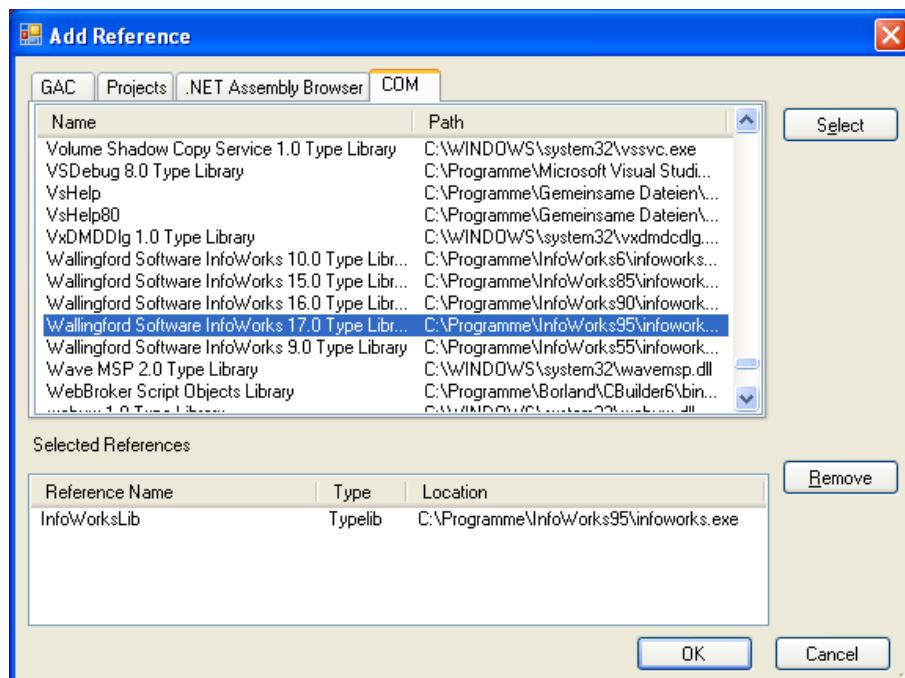


Figure 23: ‘Add Reference’ dialog box for adding references in SharpDevelop.

Trying to add a reference to the InfoWorks Type Library in this way failed. SharpDevelop complained about an unregistered library. This problem could not be resolved. However, an alternative procedure was found that granted access to the library. This procedure is given in Appendix D, again in the form of step-by-step instructions.

### 7.3.2 Content of the InfoWorks Library

This section gives an overview about the classes, methods and properties that are defined by the InfoWorks API and that are contained in the InfoWorks Type Library.

Additional to the (incomplete) documentation of the InfoWorks API (see 7.2.1), information about the API could now, having realized the access to the InfoWorks Type Library (see 7.3.1) be obtained directly from inspecting the library.

Using VBA, the content of the original InfoWorks Type Library (contained in the 'infoworks.exe' file) can be viewed by means of the VBA Object Browser (see step 4 in Appendix C). For use within SharpDevelop, the original InfoWorks Type Library was converted into a corresponding .NET assembly (see step 1 in Appendix D). The .NET assembly may be seen as the .NET equivalent of the Type Library which, in the .NET environment, can be easily accessed from any .NET language (like C#, VB.Net, etc.). The newly created .NET assembly could be analyzed by inspecting it with the .NET Framework IL Dissassembler tool (see 7.2.2 and Figure 24).

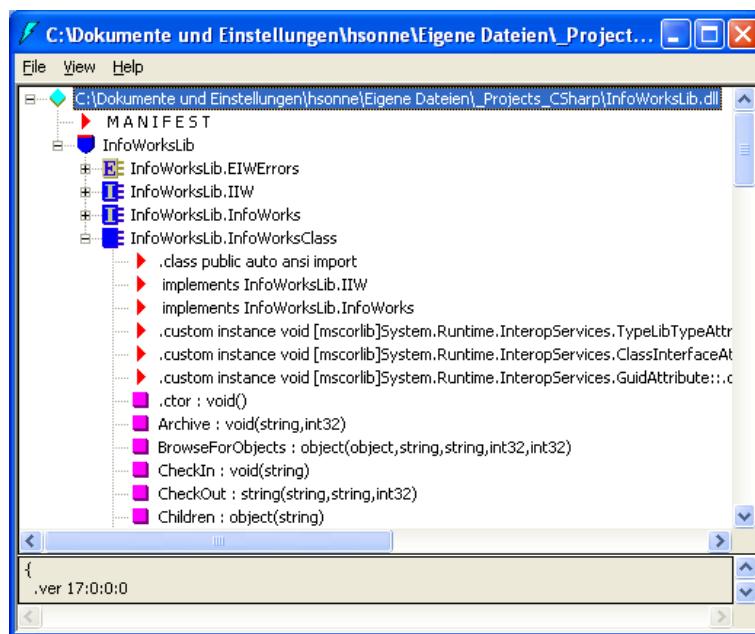


Figure 24: Inspecting the InfoWorks library by means of the .NET IL Dissassembler tool.

In the 'InfoWorksLib' assembly, all the InfoWorks functionality that is made accessible through the API is contained in the 'InfoWorksClass' being the only class defined in the assembly. An object of this class represents the InfoWorks application itself. All available methods are contained in this class. It has to be taken attention that the 'InfoWorksClass' is not contained in the original InfoWorks Type Library (contained in the 'infoworks.exe' file). There, the corresponding class is called just 'InfoWorks'.

In the assembly as well as in the original type library, there are no separate classes representing the different elements that play a role in an InfoWorks model (like networks,

rainfall events, waste water specifications, inflows, levels, runs, etc.). This means that the InfoWorks API does not follow an object-oriented approach. According to the object-oriented concept, methods and properties that correspond to the same kind of object would be grouped together to classes (e.g. network class, rainfall event class, waste water class, etc.). Concrete representations of these classes (i.e. one specific network, one specific rainfall, etc.) would be referred to as objects of the corresponding classes.

In the terminology of the API documentation, however, the term ‘object’ is used in a different context (see e.g. ‘Creation of CS run and sim objects’ in 7.2.1, Table 2). There, it refers to the above mentioned InfoWorks elements and is not used in the programming-specific sense. This may confuse when working with the InfoWorks Type Library together with the API documentation.

Inspecting the InfoWorks Type Library, or, more precisely, the ‘InfoWorksLib’ assembly, revealed that there are more methods available through the API than are described in the documentation. Table 7 and Table 8 of Appendix E list all methods and properties, respectively, that are actually contained in the InfoWorksLib assembly. It is indicated, for which of them a description can be found in the documentation.

### 7.3.3 Calling individual API methods

The preliminary steps needed to access the InfoWorks Type Library or rather the corresponding .NET assembly from within a C# program in SharpDevelop are (see also Appendix D):

1. Adding a reference to the .NET assembly ‘InfoWorksLib’ to the SharpDevelop project (see Appendix D, step 4).
2. Declaring the usage of the library at the beginning of the source code file with  

```
using InfoWorksLib;
```

(see code example in Appendix D, step 5).
3. Creating an object (also called ‘instance’) of the ‘InfoWorksClass’.

In the code example given in the InfoWorks API documentation, this is (analogously) done by:

```
InfoWorksClass iw;  
iw = new InfoWorksClass();
```

In the C# test program written within this study, this was slightly modified to:

```
InfoWorks iw;  
iw = new InfoWorksClass();
```

In both versions, a new instance of the InfoWorksClass is created and a reference to that instance is stored in the variable ‘iw’ (second line). However, in the upper version ‘iw’ is declared as being of type ‘InfoWorksClass’, whereas in the lower version, it is declared as being of type ‘InfoWorks’ (first line in each case).

As indicated by the blue symbol labelled ‘I’ (Interface) in Figure 24, the type ‘InfoWorks’ represents an interface and not a class. That interface corresponds to the actual InfoWorks API. An interface always needs to be implemented by a class, in which the declared interface members are filled with concrete program code. In this case, this is realized by the ‘InfoWorksClass’, which implements the ‘InfoWorks’ interface. This distinction between class and interface holds only true for the created .NET assembly and not for the original InfoWorks Type Library

(contained in the ‘infoworks.exe’ file) where only the class InfoWorks exists.

Having declared the variable ‘iw’ as ‘InfoWorks’ in the lower version allows ‘iw’ to refer to an object of any class that implements the ‘InfoWorks’ interface. Thus, the second version is more general than the first one, which requires ‘iw’ to hold a reference to an object of exactly the ‘InfoWorksClass’. Both versions perform exactly the same. However, the second version was preferred because only that version made the auto-complete function of the SharpDevelop editor work. The auto-complete function suggests method names and property names while typing and gives a preview of required parameters.

Now, having an instance of the InfoWorksClass, the different methods and properties of that object can be accessed by means of the variable ‘iw’.

#### 4. Initializing the InfoWorksClass object.

```
iw.Initialise(0, "", "");
```

Before running any other methods it is necessary to initialize InfoWorks by calling the ‘Initialise’ method. In the API documentation, the former version of that method ‘InitForTest’ is explained, which seems to do exactly the same. The first parameter (‘0’) determines which part of the InfoWorks software (InfoWorks CS = 0, InfoWorks WS = 2, InfoWorks RS = 4, InfoNet = 6) is meant to be initialized. The second and third parameters are username and password for Oracle or SQL Server. They can be left blank (like here) if the actual InfoWorks installation is based on a Microsoft Jet Database.

A full list of methods and properties is given in Appendix E. The following concentrates on methods that are related to the specific tasks, which need to be performed within the scenario given in section 4.2. Differences between documented and observed behaviour of methods or properties are reported and, if available, additional hints are given. The initially posed questions (see 7.1) are answered. The code examples given below assume that the preliminary steps above have been performed.

It has to be kept in mind that for many methods that did not work as expected, it could not be judged, if this was due to a wrong usage of the method or due to any other reason (e.g. incomplete implementation, misleading naming of methods or method parameters, etc.). Unfortunately, the poor documentation of the InfoWorks API (see 7.2.1) did only allow for a trial-and-error testing approach.

#### Methods on InfoWorks objects

- Objects in the InfoWorks database are identified by a so called scripting path. Beginning at the root, the scripting path describes the way to the object in the object tree. Because it is possible to have objects of the same name but different types in the same Model Group (e.g. a Network and an Event with the same name), at each point in the tree the name of the object is prefixed with the type of the object. An example for a scripting path is

```
>CG~SPREE2011_OMI_Test>RUNG~API_Test>RUN~run2
```

The different levels of the tree are separated by the ‘>’ character while in each case the object name is separated from its type code by the ‘~’ character. The type codes ‘CG’ and ‘RUNG’ refer to ‘Catchment Group’ and ‘Run Group’, respectively.

- The Children method allows to get a list of objects that belong to a group of objects, e.g. Run objects within a Run Group. The following code snippet gives an example of how each child object can be accessed in a loop over the array that is returned by the Children method:

```
object[] children;
children = (object[]) iw.Children(">CG~SPREE2011_OMI_Test");
for (int i = 0; i < children.GetUpperBound(0); i++) {
    // Do something with children[i], e.g.:
    Console.WriteLine(children[i].ToString());
};
```

- The property Exists can be used to check if an object to which the scripting path is given, does exist in the InfoWorks database.
- New objects can be created by the NewObject method.
- The API contains a Delete method, which is not described in the documentation. Within the tests, sometimes the Delete method worked, sometimes it failed. It was figured out that the deletion of objects is not allowed under certain circumstances. This can be checked in the InfoWorks GUI. In the context menu of an object chosen in the Catchment Group window (which appears when right clicking onto that object) the entry 'Delete' is disabled or not. For example, the deletion of a Network object is disabled if the network is used in a run. If the Delete option is disabled, the API method Delete fails. Trying to delete an Inflow object always failed.
- It seems that there is no API method being able to make a copy of an existing InfoWorks object. It would be useful to have such a method for copying Run objects. In section 7.3.4, a workaround will be described.
- For checking in and checking out networks, the InfoWorks API contains the methods CheckIn and CheckOut, respectively. Only the CheckIn method is mentioned in the InfoWorks API documentation. Calling CheckIn via API worked.
- As opposed to the InfoWorks GUI, the InfoWorks API does not offer a method that performs a simple validation but only the method EngineeringValidation that runs a more detailed validation. The EngineeringValidation method asks for the scripting path to an Engineering Validation (EV) object. Calling the method with the path to a default EV object, that has been created by means of the InfoWorks GUI, worked.
- There have been general problems with Inflow objects. Invoking methods (e.g. Children, get\_DisplayName, get\_Type, Delete) on an Inflow object failed, although the scripting path of the Inflow object was received by the BrowseForObjects method which lets the user choose only existing objects.

### *Importing data*

- For importing several kinds of input data, the InfoWorks API offers the Import method. Among other parameters, the 'parent' object must be given, of which the imported object will be a 'child'. When, for example, importing a Rainfall Event, the parent object must be a Rainfall Group. The Import method also accepts a format specifier, which, in some cases, allows to alter the behaviour of the method. When specifying the import file, always the full path to the file should be

given. Placing the file in the current directory and specifying it only by name failed.

- Importing Rainfall Events: Independent from the given format specifier (it shall be left blank), the event will be imported from a file in the normal event file format, that is, the ‘red’-Format. Rainfall data from CSV-files cannot be imported.
- Importing Networks: Networks can be imported from CSV-files. There are two different format specifiers: While the specifier ‘CSV’ will create a new network from the imported CSV-file, ‘CSVUP’ will update an existing network. The results of test imports with different format specifiers and different ‘parents’ are given in the following table:

<b>Format</b>	<b>‘Parent’</b>	<b>Result</b>
CSV	Catchment Group	The imported network is created as a new Network within the Catchment Group.
CSV	Network	This is not intended. The ‘parent’ should always be a Catchment Group if a new network shall be imported (with ‘CSV’).
CSVUP	Network (checked in)	The changed Network is created one level deeper than the ‘parent’; and it is checked out.
CSVUP	Network (checked out)  E.g. in Figure 25, left: ‘Net_A#1’	The changed Network is created at the same level as the ‘parent’, and it is checked out.  E.g. in Figure 25, left: ‘Net_A#1_rev1’ and ‘Net_A#1_rev2’, resulting from updating ‘Net_A#1’ twice.
CSVUP	Network (created with 'Check out and branch')  E.g. in Figure 25, right: ‘myNetwork_new_1’	The changed Network is created one level deeper than the ‘parent’, and it is checked out.  E.g. in Figure 25, right: ‘myNetwork_new_1_rev1’ and ‘myNetwork_new_1_rev2’, resulting from updating ‘myNetwork_new_1’ twice.

Imported networks need to be validated before they can be used within a simulation run. Therefore, the API method `EngineeringValidation` can be used (see above in this section).

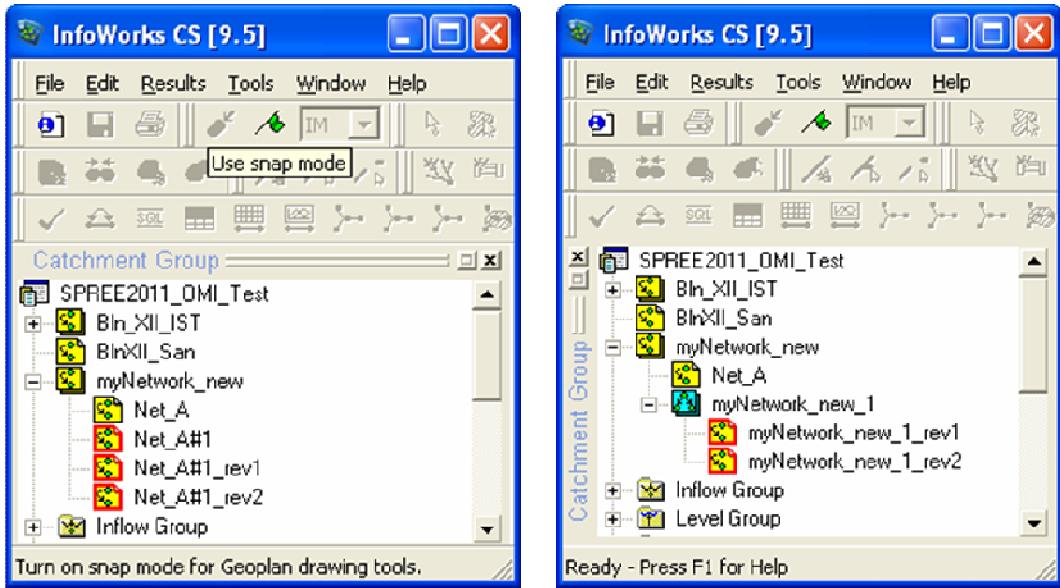


Figure 25: Network tree after importing networks with the API Import-method.

- Importing Inflow: No matter which format specifier is chosen, only files in ‘qin’-format can be imported. Trying to import inflow data from a CSV-file (as it is possible via the InfoWorks GUI), created an Inflow object, but that did not contain any data.

#### *Changing data*

- In InfoWorks, all information about the model is contained in a database, consisting of different tables for the different types of objects (e.g. Rainfall Event, Inflow, Level, Network, Run). The columns (fields) of a table represent the variables that specify the objects in that table. The InfoWorks API allows access to specific fields and tables. Which fields of which tables can be accessed via the API, is documented in Appendix B of the InfoWorks API documentation. Concerning InfoWorks CS, the only table that allows access to many of its fields is the Run table. In the InfoWorks GUI, the ‘Schedule Hydraulic Run’ dialog (cf. Figure 6 in section 5.2.2) can be seen as the panel that gives access to the Run table. Thus, the Run parameters that can be set in this dialog are mainly the parameters that can be accessed through the InfoWorks API.
- The fields, to which access is allowed, can be got and set by means of the Value property that is part of the API. The Value property accepts two parameters (scripting path to the object to be changed and field name of the variable to be changed). As properties in C# are not able to accept parameters, the .NET assembly, that has been created out of the InfoWorks Type Library (see 7.3.2), contains the corresponding methods get\_Value and set\_Value for accessing the object variables (cf. Table 8 in Appendix E).

#### *Running a simulation*

- The Run method performs an InfoWorks CS simulation run. The method accepts the scripting path of the Run object that specifies the simulation to be run. When the Run method is called, the Simulation Controller (cf. step 5 in Appendix C)

appears and runs the complete simulation. There are corresponding run methods (RSRun and WNRun, respectively) for running InfoWorks RS and InfoWorks WS models, respectively (cf. Appendix E).

#### *Exporting data*

- The InfoWorks API provides two different methods that export data to files: the method Export for exporting different kinds of InfoWorks objects (e.g. Rainfall Events, Selection Lists, Networks) and the method ExportResults for the export of simulation results.
- The Export method accepts a parameter that determines the desired output format. For example, InfoWorks CS Networks can be exported to the file formats CSV (Comma Separated Values), MIF (MapInfo Interchange Format), SHP (ESRI shape files), GDB (ArcGIS personal GeoDatabases) and DSD (HydroWorks).
- Exporting Networks: Using the Export method for CSV export, the whole Network is exported to exactly one CSV file containing the data of all database tables that describe the Network. Using the InfoWorks GUI, the export of Networks can be configured (e.g. export to multiple files instead of one file). However, the Export method provided by the InfoWorks API does not seem to allow for those changes in the export behaviour.
- Exporting simulation results: The ExportResults method can export simulation results to CSV, QIN or Hyx style files. Concerning CSV export, the attribute to be exported (e.g. depth, flow, mass concentrations of COD, BOD, etc.) and the link end (upstream or downstream) need to be specified. The list of links, for which the results are requested, can be given to the method in the form of a comma separated list or as the path to a corresponding Selection List object. According to the API documentation and according to the tests, the export of results is restricted to link attributes.

Having obtained the above results and recalling Question 3, posed in section 7.1, it can be summarized that the methods provided by the InfoWorks API allow for:

- importing model and input data from files. However, less file formats are supported compared to import via the InfoWorks GUI.
- changing model parameters. However, this is mainly limited to Run parameters. For changing other parameters, a workaround is needed (see 7.3.4).
- running simulations. However, only complete simulations can be invoked. It is not possible to access the model (e.g. asking for current values of model state variables) while running.
- exporting simulation results to files. However, it seems that only properties of Links (e.g. conduits, pumps, weirs) can be exported. It is unknown if there is a possibility to export properties of Nodes (e.g. the water level in a manhole).

#### **7.3.4 Performing more complex tasks by calling API methods**

This section describes first how tasks, for which the InfoWorks API does not directly offer appropriate methods or properties, can be performed indirectly by adequate workarounds. Following, the main procedure is given that is basically performed within the C# program being able to solve the simple optimization problem formulated in the use case

of section 7.2.3 by using InfoWorks functionality through the InfoWorks API. Within the same program, all API tests have been performed. In order to give an impression about the structure of the program, its class diagram is shown in Figure 26 in Appendix F.

#### *Workaround 1: Changing model parameters*

Network parameters which cannot be accessed directly by the Value method (see 7.3.3) could successfully be changed indirectly by

- exporting the whole network to a CSV file with the Export method,
- extracting those lines out of the file that represent the Network elements to be changed,
- modifying those fields of the extracted lines that contain the parameters to be changed,
- writing a new file containing the modified lines plus corresponding table headers,
- importing the generated file with the Import method and with the format specification set to 'CSVUP' (update).

The given procedure has been implemented in a C# class named IWNNetwork, which provides GetValue and SetValue methods. Each of the methods accepts as parameters the type of network table and an identification of the elements, as well as the name of the field to be changed. In addition, the SetValue method receives the new value of the network parameter to be changed. The SetValue method implements basically the steps listed above.

#### *Workaround 2: Copying Run objects*

In order to perform a new simulation with a modified network, it would be useful to copy an existing, pre-defined Run object into a new Run object of which then just the Network property (name of the network to be simulated) needed to be changed. As an API method for copying objects was not available (see 7.3.3), the following procedure was realized:

- Create a new Run object by means of the NewObject method.
- Copy all values of the original Run object into the newly created one.

In the second step, those values are copied that are documented in the API documentation. However, there might be more parameters. Also concerning the second step, it was experienced that at least setting the Inflow property did not work. The scripting path of an Inflow object being set in the original Run object is said to be non-existent although it actually does exist (see also reported problems with Inflow objects in 7.3.3).

The above procedure is implemented in two methods called MakeRunVersion and SetParameterSetting, respectively. These methods are defined in the IWRUN class (cf. Appendix F).

#### *Solving a simple optimization problem by use of the InfoWorks API*

The developed C# program that solves the optimization problem defined in section 7.2.3 basically performs in the following steps:

1. Choose a prepared InfoWorks Run (current Run).

2. In the Network of the current Run, set the volume of the stormwater tank (→ Workaround 1) to an initial value (if this step is performed for the first time) or increase the volume by a constant value (else).
3. Copy the current Run object into a new Run (→ Workaround 2). In the new Run, specify that the network to be simulated is the new, modified network.
4. Perform the new Run, simulating the new, modified network.
5. From the simulation results of the Run, export the tank overflows to a file.
6. Calculate the total overflow volume from the exported file.
7. If the overflow volume is still above a certain threshold, return to step 2, else quit.

Alltogether, it can be summarized that it was possible to use the API for some of the MIA-CSO purposes. The export of node properties could not be realized by means of InfoWorks API calls. In some cases, node properties can be deduced from the properties of connected links. Accessing InfoWorks through its API, a simple optimization problem could be solved in an automated manner.

## **Chapter 8: Summary and Conclusions**

The overall objective of MIA-CSO is to develop a model-based planning instrument for impact based CSO control. The objective of this study was to examine the potential and the drawbacks of different model coupling techniques that may be taken into account within the MIA-CSO project.

- The models InfoWorks CS (Wallingford Software Ltd.) and HYDRAX/QSim (German Federal Institute of Hydrology, BfG) that are intended to be used within MIA-CSO as well as two different techniques for model coupling (Open Modelling Interface – OpenMI and Application Programming Interfaces – API) are described (Chapter 2).
- It is presented how impacts of combined sewer overflows (CSO) on receiving waters have been simulated in Berlin, so far (Chapter 3).
- According to the objectives of MIA-CSO (statistical analysis, uncertainty and sensitivity analysis), chapter 4 describes in the form of two possible scenarios, how the intended integrated modelling system could be realized:
  - scenario one is based on the application of the OpenMI,
  - scenario two is based on the application of model-specific APIs.
- Practical tests are reported, in which simple scenarios, consisting of available OpenMI components, have been set-up and carried out by use of the OpenMI Configuration Editor. Potential and drawbacks of the OpenMI implementation in InfoWorks CS are described (Chapter 5).
- A wrapping procedure is described that helps making existing models comply with the OpenMI standard. Assumptions are made about the effort involved (Chapter 6).
- Experiences with the InfoWorks Type Library, which defines the InfoWorks API, are reported and the available methods and properties are documented. It is explained how specific tasks can be handled by means of API calls (Chapter 7).

General Results and conclusions are:

- Concerning model coupling techniques, sequential linking, where models run one after the other can be distinguished from parallel linking where models advance simultaneously in time and where models may exchange data at each timestep. In the current modelling practice applied in Berlin, models run sequentially. Simulation, data export and import as well as data transformation are realized manually. By automating these processes, the modelling procedure can be improved in terms of efficiency, error-proneness and accuracy.

Results and conclusions concerning the OpenMI:

- The OpenMI is a generic interface specification that standardizes the parallel linking approach. It defines the manner in which models are run and data are exchanged between models (at each timestep). The process of making existing models compliant to the OpenMI standard (migration) can be seen as teaching the models to ‘speak the same language’.

- Migration of a model does not per se guarantee that the model can be used in any OpenMI scenario. The useability of OpenMI components strongly depends on
  - how models implement the OpenMI and
  - which exchange items and additional data operations they offer.
- Migration requires the model's source code to be available. The migration process is supported by software tools that are provided by the OpenMI Association. The effort needed to migrate an existing modelling software into a corresponding OpenMI-compliant component depends on the tasks to be performed by the OpenMI component, personal preconditions and technical preconditions. It strongly depends on the structure of the existing model engine.
- A model will only be made OpenMI-compliant if there is an incentive for the model developer to do so.
- In software development, detailed planning and well-chosen structures can prevent from difficulties in modifying a software when changes or extensions become necessary. The OpenMI provides a pre-defined structure, developed by experts in environmental modelling. It may be a great help for software engineers who are requested to couple models, to have such a structure to follow. Following a structure does not only mean a restriction but can also support the developer.
- Gaining experience in the OpenMI is assumed to be a kind of expert knowledge that may be strongly requested in the near future. This is expected as the OpenMI is already used within many research projects (even outside Europe).
- Of the model softwares used to date within MIA-CSO, only InfoWorks CS supports the OpenMI. The current OpenMI implementation in InfoWorks only allows for flows and water levels as input quantities.
- If OpenMI is implemented or not, will depend on the personal interests of the model developers and on their ability in doing the migration. At least within near future, it is unprobable that the models Hydrax and QSim that are intended to be used within MIA-CSO will be migrated to the OpenMI standard.
- Further developments at KWB will therefore concentrate on additional tools for the MIA-CSO planning instrument, which allow for e.g. statistical analysis, automated calibration, uncertainty and sensitivity analysis. By doing so, the OpenMI approach can be followed, which will make own developments also attractive to others.

Results and conclusions concerning API:

- Through Application Programming Interfaces (API), a software allows a selection of its internal functions to be called from other softwares. An API is specific to the software that it is provided by. An API often allows for automated access to those functions of a software that normally are invoked by a Graphical User Interface (GUI).
- Compared to the OpenMI scenario, in the API scenario models are processed sequentially. However, model preparation, data import, start of simulation and data export are automated using the provided API functionality.

- The InfoWorks API documentation provided by Wallingford Software Ltd. is incomplete. This can be problematic when InfoWorks shall be accessed through its API.
- This report describes step-by-step how InfoWorks CS can be accessed through its API from within the programming languages VBA and C#, respectively.
- The InfoWorks API allows for importing model data and input data from files. Changing of model parameters can be realized indirectly by a sequence of data export, data manipulation and data import. The same holds true for e.g. rainfall series data. Only complete simulations can be invoked. Being limited to properties of links (e.g. conduits, pumps, weirs), simulation results can be exported to files.
- The possibility to change internal model parameters through the InfoWorks API allows for applying this interface for e.g. statistical analysis, automated calibration, uncertainty and sensitivity analysis.

## Appendix A: Glossary of programming-specific terms

Here, some of the terms that are commonly used in programming and that are used in this report, are introduced.

Term	Description
Class	In object-oriented programming, a class describes the structure and behaviour of <u>objects</u> . This is done in the form of <u>properties</u> and <u>methods</u> . A class may be seen as a template for creating objects.
Exception	An exception can be seen as an <u>object</u> that describes an error. The exception can be ‘thrown’ through an application until it reaches a part of the software which ‘feels responsible’ for that kind of error, ‘catches’ and handles it.
Function	Functions are sub-routines that are able to return a value. In object-oriented programming, functions are referred to as <u>methods</u> .
Instantiation	See description of <u>object</u> .
Method	Methods are <u>functions</u> . They can be called in order to perform a process on an <u>object</u> . Calling methods may also change the <u>properties</u> of an object.
Object	An <u>object</u> is a discrete software entity that represents a concrete realization of a <u>class</u> . Objects only exist during the run-time of the software. They need to be created. The creation of objects is also referred to as <u>instantiation</u> .
Property	<u>Properties</u> represent the state of an <u>object</u> . In the <u>class</u> , of which the object is a representative, properties are defined in terms of internal variables that can be got (read) or set (written).
Visual Basic for Applications (VBA)	Visual Basic for Applications (VBA) is a programming language developed by Microsoft that is closely related to Visual Basic. It can be used to automate applications that offer an API from within Microsoft Office applications like Excel, Access, ...
Windows Registry	The Windows Registry is a database which stores settings and options for Microsoft Windows operating systems. An editor which gives access to the registry can be started with ‘regedit’ from the Windows command prompt.

## Appendix B: The IEngine interface

The following table lists the methods defined by the *IEngine* interface. That interface needs to be implemented by the wrapper class *MyModelEngineWrapper* if the wrapper pattern recommended by the OpenMI Association Technical Committee is used for model migration (see 6.1.2 and 6.1.3, steps 5 and 7, in the main text). The descriptions of the methods have been directly taken from comments in the C# source code. See (Gijsbers et al., 2007a) for details on return values, parameters and types.

Table 5: Methods defined in the IEngine interface being provided in the wrapper package of the OpenMI SDK.

<b>Model description methods</b>	
string GetModelID()	Returns the model id. The model id identifies the populated model component. Example: 'River Rhine'.
string GetModelDescription()	Returns the model description. The model description is a description of the populated model component.
ITimeSpan GetTimeHorizon()	Returns the time horizon for the populated model component. The time horizon for a model is typically the same as the simulation period, which normally depends on the available input data. When your model is running in the OpenMI environment, the model component must be able to return values within the time horizon.
<b>Exchange item methods</b>	
int GetInputExchangItemCount()	Returns the number of input exchange items for the populated model component.
int GetOutputExchangItemCount()	Returns the number of output exchange items for the populated model component.
OutputExchangItem GetOutputExchangItem( int exchangItemIndex)	Returns a specific output exchange item from the populated model component.
InputExchangItem GetInputExchangItem( int exchangItemIndex)	Returns a specific input exchange item from the populated model component.
<b>Execution control methods (inherited from IRunEngine)</b>	
void Initialize(Hashtable properties)	Initialize will typically be invoked just after creation of the object that implements the <i>IRunEngine</i> interface.

bool PerformTimeStep()	This method will make the model engine perform one time step.
void Finish()	This method will be invoked after all computations are completed. Deallocation of memory and closing files could be implemented in this method.
void Dispose()	This method will be invoked after all computations are completed and after the Finish method has been invoked.
<b>Time methods (inherited from IRunEngine)</b>	
ITime GetCurrentTime()	Gets the current time of the model engine.
ITime GetInputTime( string QuantityID, string ElementSetID)	Gets the time, for which the next input is needed for a specific quantity and element set combination.
ITimeStamp GetEarliestNeededTime()	Gets the earliest needed time, which can be used to clear the buffer. For most time stepping model engines this time will be the time for the previous time step.
<b>Data access methods (inherited from IRunEngine)</b>	
void SetValues( string QuantityID, string ElementSetID, IValueSet values)	Sets values in the model engine.
IValueSet GetValues( string QuantityID, string ElementSetID)	Gets values from the model engine.
<b>Component description methods (inherited from IRunEngine)</b>	
double GetMissingValueDefinition()	In some situations a valid value cannot be returned when the <i>IRunEngine.GetValues</i> method is invoked. In such case a missing value can be returned. The <i>GetMissingValuesDefinition</i> method can be used to query which definition of a missing value applies to this particular model component. Example of missing value definition could be: -999.99
string GetComponentID()	Gets the component id. The component id is the name of the non-populated component. This is typically the product name of your model engine.

string GetComponentDescription()	Gets a description of your component. This description refers to the non-populated component. This is typically a description of what your component does and which methods are used. E.g. ‘Finite element based ground water model’.
-------------------------------------	---

## Appendix C: Step-by-step procedure to access the InfoWorks API from within VBA

This appendix gives step-by-step instructions that can be followed in order to access the InfoWorks API from within the Microsoft Visual Basic environment (VBA). A VBA editor is available in Microsoft Excel. Table 6 contains menu commands that have to be invoked in Microsoft Excel in order to perform the actions that are referenced in the instructions given below.

Table 6: Microsoft Excel menu commands that are required when following the step-by-step procedure for accessing InfoWorks through its API.

Action	Active Window	Sequence of menu items to select
Open the VBA Editor	Microsoft Excel	English: Tools→Macro→Visual Basic Editor (Alt+F11) German: Extras → Makro → Visual Basic-Editor (Alt+F11)
Start the References dialog box	Microsoft Visual Basic	English: Tools → References... German: Extras → Verweise...
Start the VBA Object Browser	Microsoft Visual Basic	English: View → Object Browser (F2) German Ansicht → Objektkatalog (F2)
Insert a new module	Microsoft Visual Basic	English: Insert → Module German: Einfügen → Modul
Run the current procedure	Microsoft Visual Basic	English: Run → Run Sub/UserForm (F5) German: Ausführen → Sub/UserForm ausführen (F5)

### Step-by-step procedure:

1. Start Microsoft Excel, open a new workbook and save the workbook under a name of choice.
  2. Open the Microsoft Visual Basic (VBA) Editor (according to Table 6).
  3. Start the References dialog box (according to Table 6).
- Tick the entry 'Wallingford Software InfoWorks 17.0 Type Library' or another InfoWorks Type Library that corresponds to the program version installed on your computer:
- ‘10.0 Type Library’ corresponds to InfoWorks version 6.0.
  - ‘11.0 Type Library’ corresponds to InfoWorks version 6.5.
  - ‘12.0 Type Library’ corresponds to InfoWorks version 7.0.
  - ‘13.0 Type Library’ corresponds to InfoWorks version 7.5.

'14.0 Type Library' corresponds to InfoWorks version 8.0.

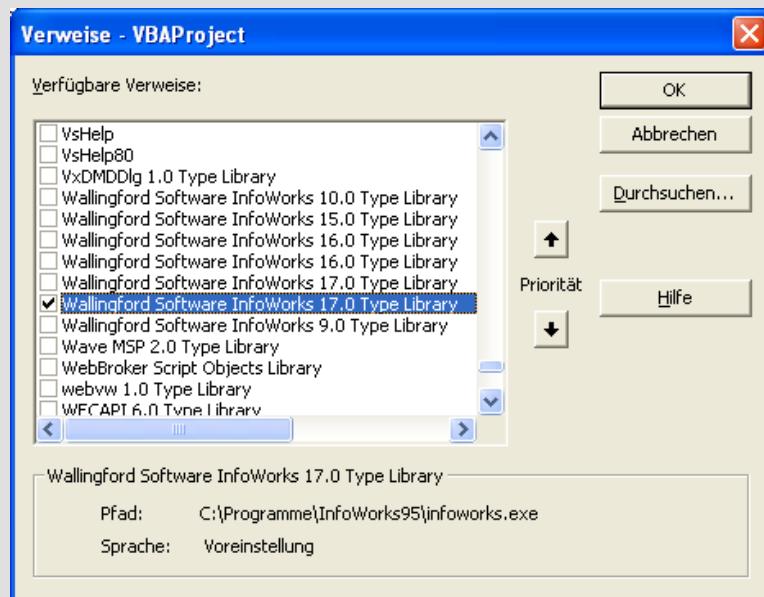
'15.0 Type Library' corresponds to InfoWorks version 8.5.

'16.0 Type Library' corresponds to InfoWorks version 9.0.

'17.0 Type Library' corresponds to InfoWorks version 9.5.

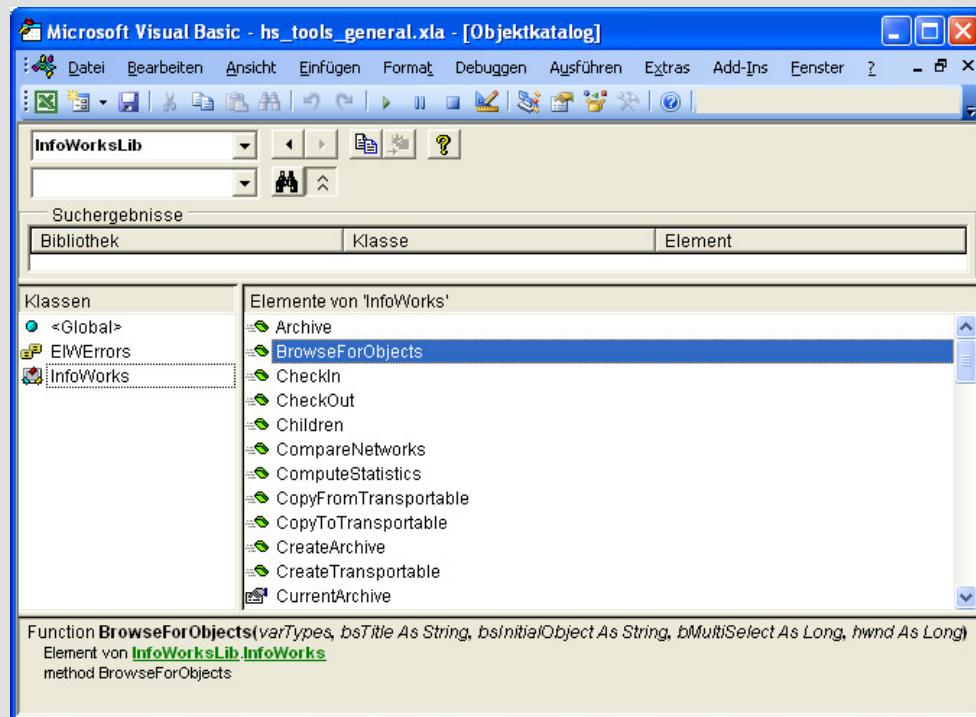
If there is no such entry, then InfoWorks has not been installed correctly.

Quit the dialog box with 'OK'.



4. Start the VBA Object Browser (according to Table 6).

Choose the object library 'InfoWorksLib' from the dropdown box in the upper left corner of the window:



The classes and class members (functions/methods and properties) contained in the library can be investigated. The prototype (name, type, types of parameters) of the selected class member is shown in the bottom section of the window.

5. Write and run a very simple program that creates an instance of the InfoWorks class (being the only class in the library), initializes that instance, lets the user choose a prepared InfoWorks simulation run and runs the chosen simulation:
  - Insert a new module to the VBA project (according to Table 6).
  - In the appearing editor window, enter the following example code:

```
Sub main()

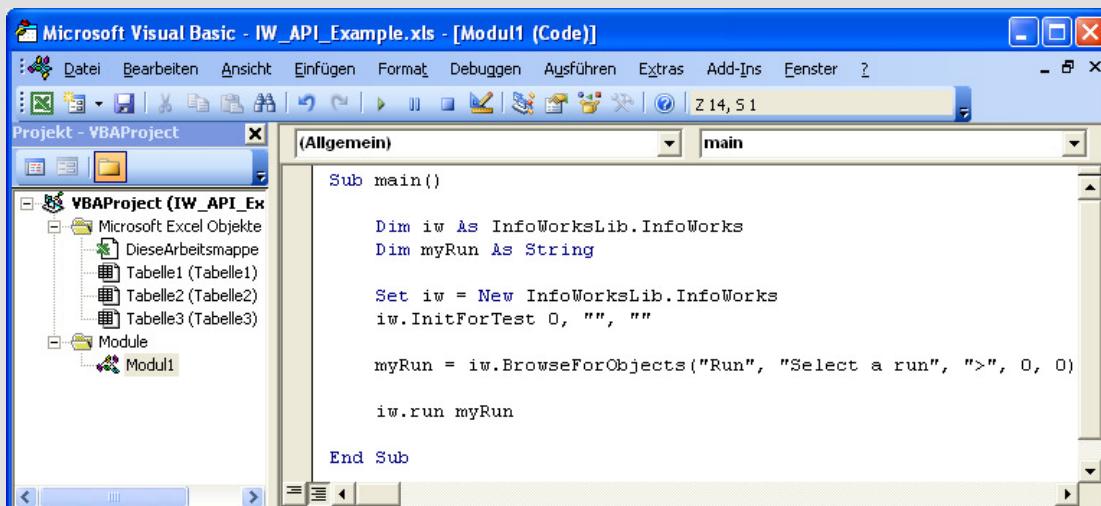
    Dim iw As InfoWorksLib.InfoWorks
    Dim myRun As String

    Set iw = New InfoWorksLib.InfoWorks
    iw.InitForTest 0, "", ""

    myRun = iw.BrowseForObjects("Run", "Select a run", ">", 0, 0)

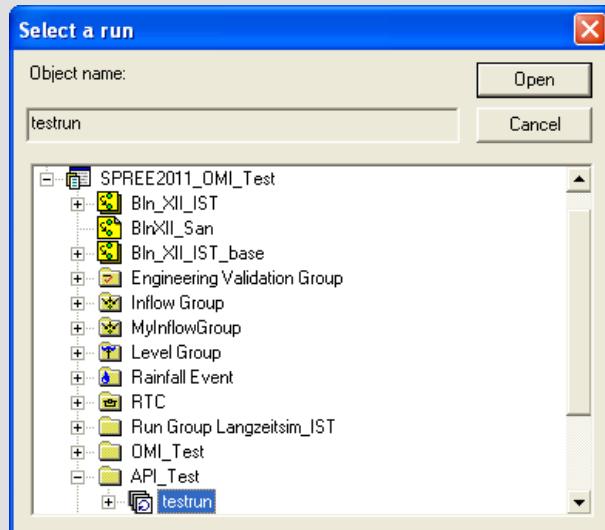
    iw.run myRun

End Sub
```

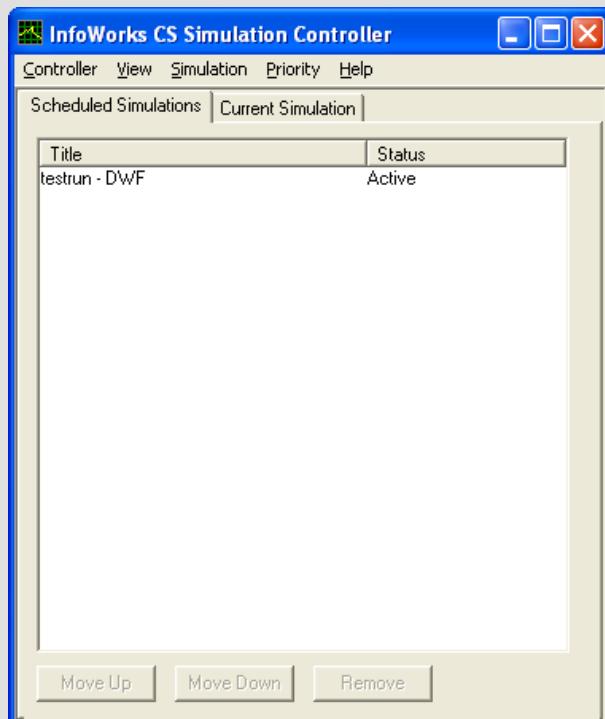


- Run the current procedure (according to Table 6). Therefore, the cursor must be placed inside the body of the entered main() procedure.

- The InfoWorks Object Browser dialog appears (with the title ‘Select a run’ as defined in the corresponding method call). Select an InfoWorks CS run and confirm with ‘Open’:

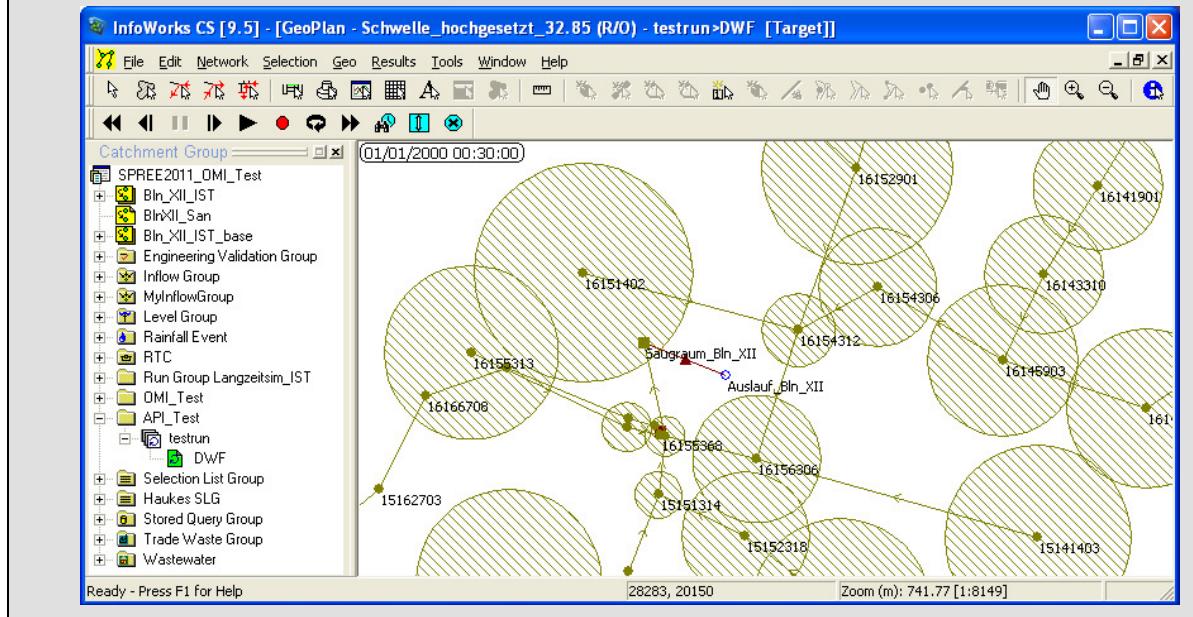


- The dialog box disappears. The InfoWorks CS Simulation Controller appears, monitoring the simulation progress of the simulation run that was selected in the last step:



- The InfoWorks CS Simulation Controller will disappear when the simulation is completed.

6. Start the InfoWorks CS application and view the results of the performed simulation:



Once having set the reference to the InfoWorks Type Library (step 3), a VBA program could be written that creates and uses an object of the InfoWorks class (step 5). The InfoWorks object offers the methods and attributes that are defined by the InfoWorks API. In the programming example, a prepared InfoWorks run was first selected by use of an InfoWorks-own dialog box and then simulated. See Appendix D for a complete list of methods and properties that are provided by the InfoWorks API.

With knowledge of VBA and by using the documentation of the InfoWorks API, the simple example program given in step 5 may be extended in order to perform more complex tasks.

## Appendix D: Step-by-step procedure to access the InfoWorks API from within C#

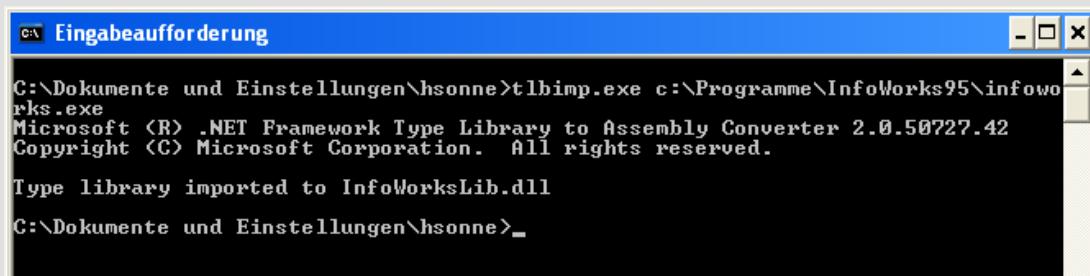
This appendix gives step-by-step instructions that can be followed in order to access the InfoWorks API from within SharpDevelop, the C# development environment used for this study (see 7.2.2).

### Step-by-step procedure:

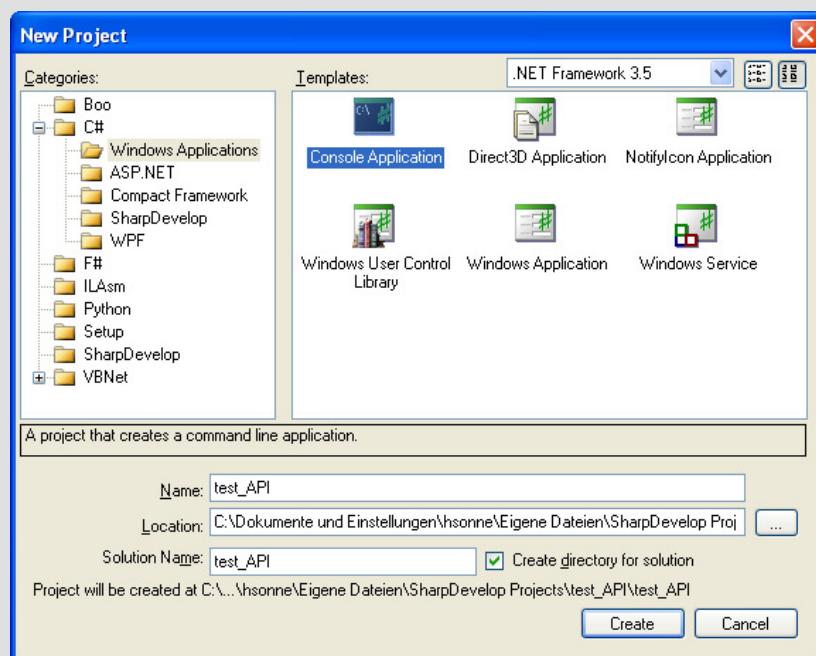
1. Use the Type Library Importer tool ‘tlbimp.exe’ to convert the type library contained in ‘infoworks.exe’ into the .NET assembly ‘InfoWorksLib.dll’.

The program ‘tlbimp.exe’ is part of the Microsoft .NET Software Development Kit (SDK) that needs to be installed (see 7.2.2). On the computer used for this study, the file ‘tlbimp.exe’ can be found in

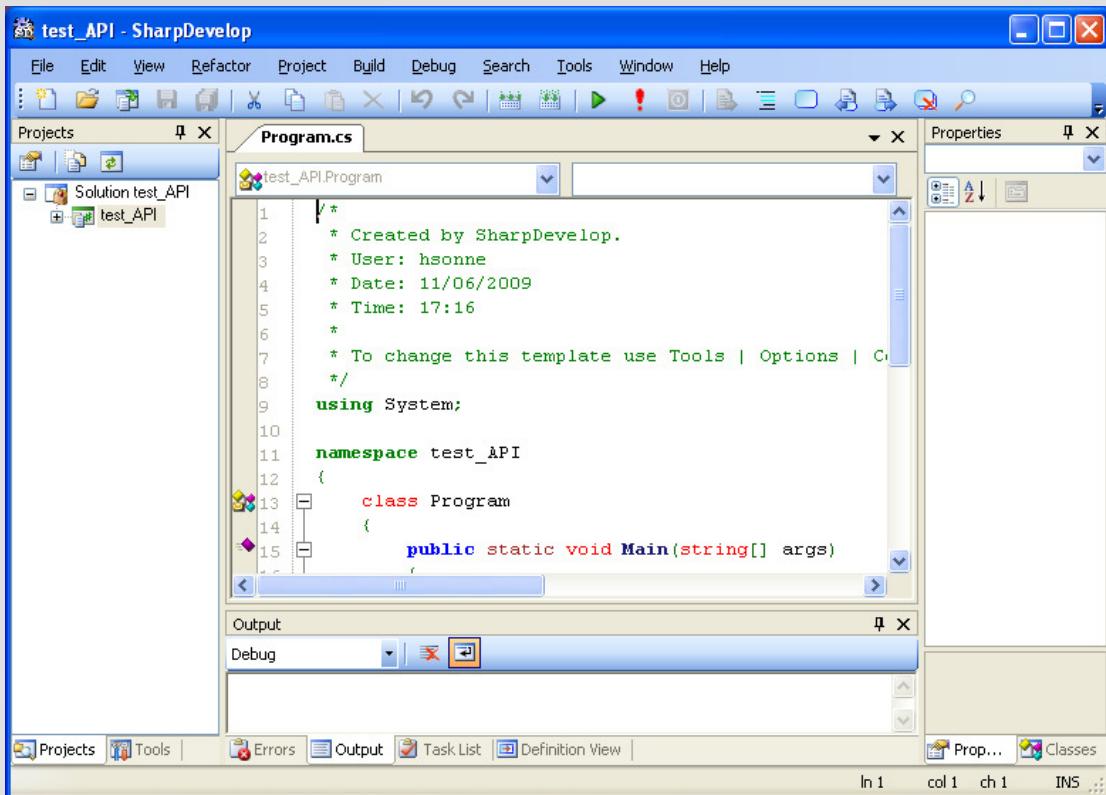
C:\Programme\Microsoft .NET\SDK\v2.0\Bin



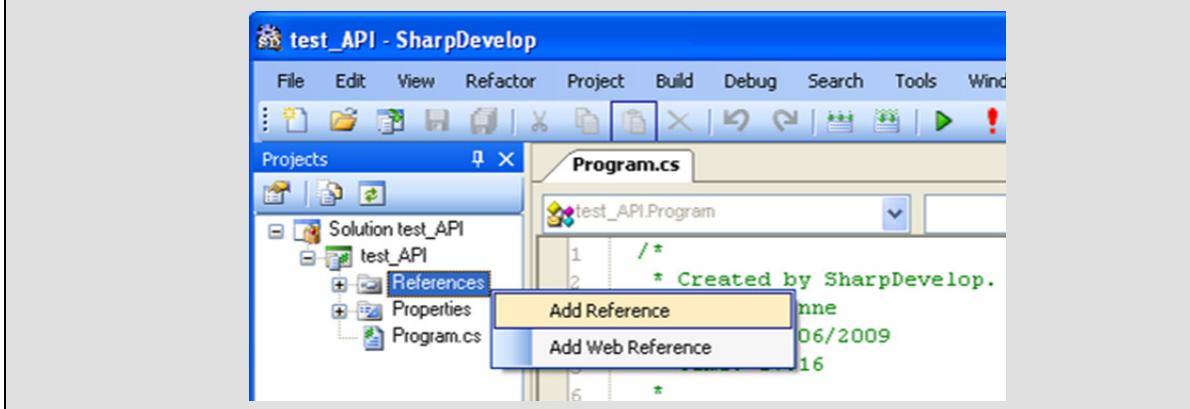
2. Start SharpDevelop. The instructions given here relate to SharpDevelop version 3.0.0.
3. Open an existing Project/Solution or create a new one. For the following steps, it is assumed, that a new Windows console application named ‘test\_API’ is created. This is done by opening the ‘New Project’ dialog (‘File → New → Solution...’), selecting the corresponding item and choosing the appropriate name and location:



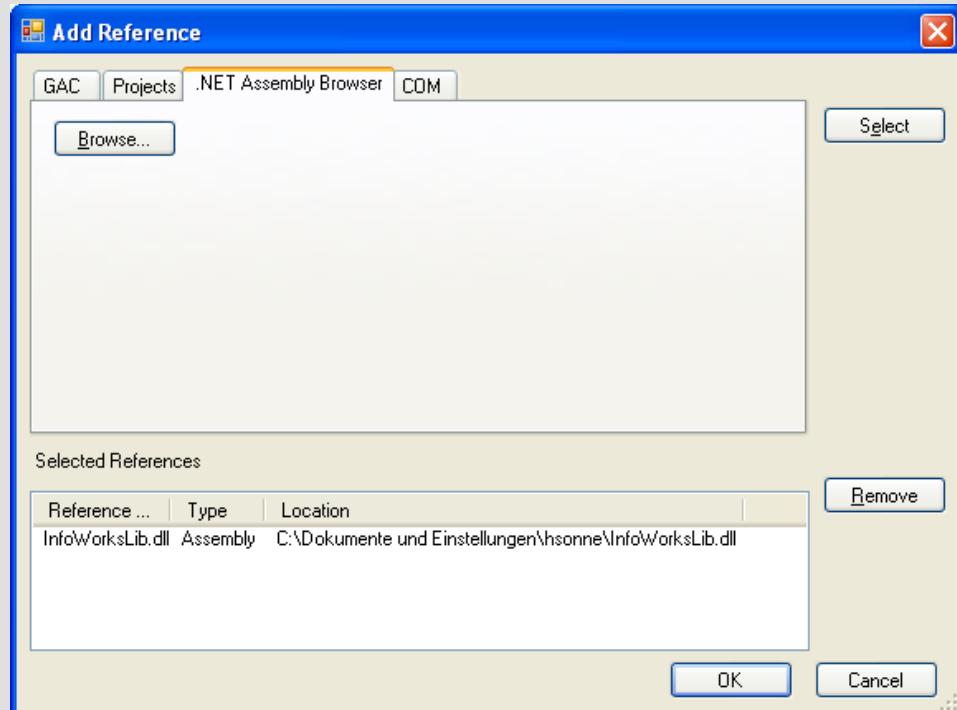
After confirming with 'Create', the SharpDevelopment environment may look like this:



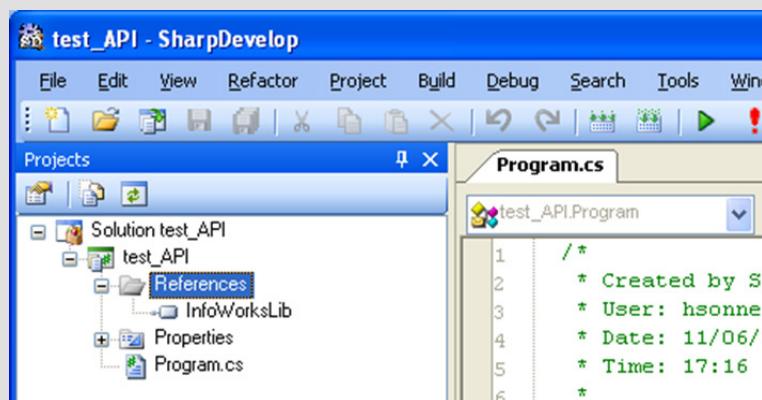
4. Add a reference to the .NET assembly 'InfoWorksLib.dll', that was created in step 1. Therefore, in the 'Projects' view (if needed to be displayed by View → Projects), choose 'Add Reference' from the context menu (right mouse click) of the 'References' item of your project:



The dialog box ‘Add Reference’ opens. Activate the ‘.NET Assembly Browser’ tab and select the .NET assembly file ‘InfoWorksLib.dll’, that was created in step 1, from the dialog that opens when pressing ‘Browse...’. The selected assembly file is displayed in the bottom part of the ‘Add Reference’ dialog.



Confirm the selection with ‘OK’. The reference to the InfoWorks library ‘InfoWorksLib’ has been added to the project/solution ‘test\_API’:



5. Write and run a simple C# program that creates an instance of the ‘InfoWorksClass’ (being the only class in the library ‘InfoWorksLib’), initializes that instance, lets the user choose a prepared simulation run and runs the chosen simulation:

- In the main program file ‘Program.cs’, change the code to the following:

```
using System;
using InfoWorksLib;

namespace test_API
{
    class Program
    {
        public static void Main(string[] args)
        {
            InfoWorks iw;
            string myRun;

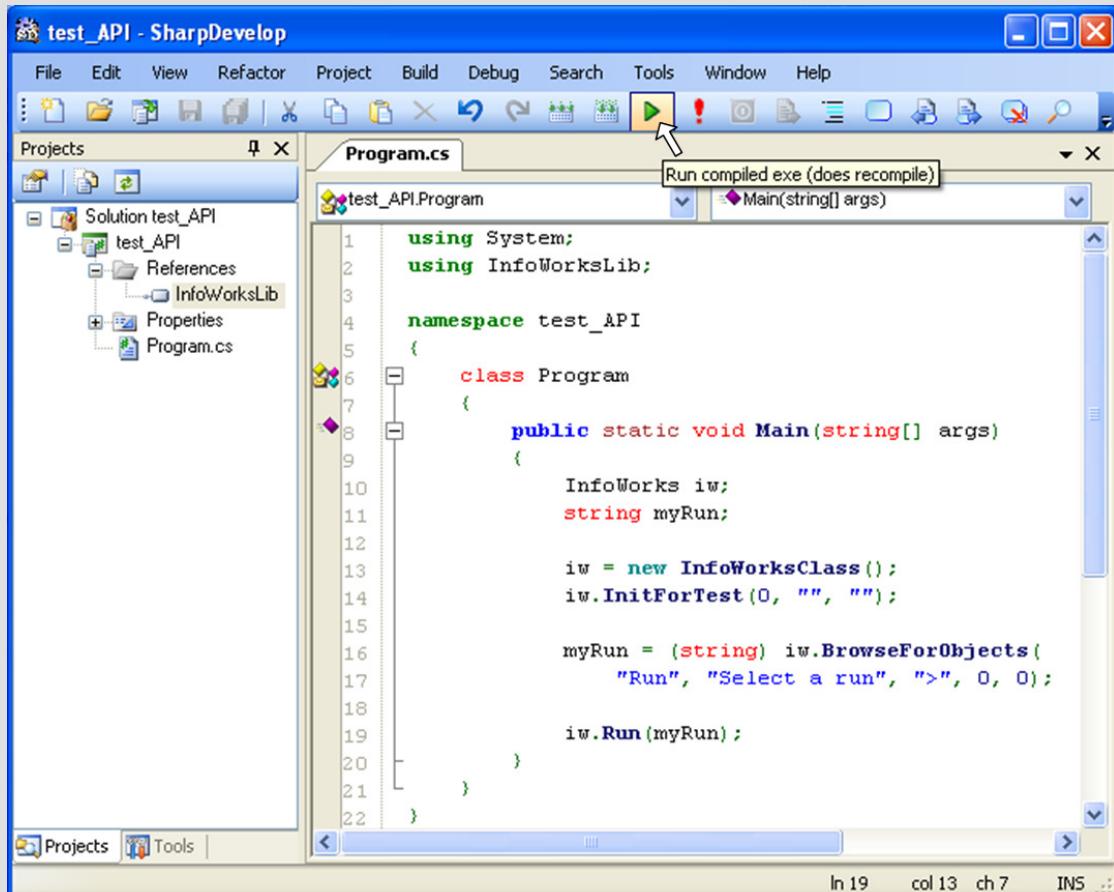
            iw = new InfoWorksClass();
            iw.InitForTest(0, "", "");

            myRun = (string) iw.BrowseForObjects(
                "Run", "Select a run", ">", 0, 0);

            iw.Run(myRun);
        }
    }
}
```

- Most important in the code is the statement ‘using InfoWorksLib’ in the second row, which makes the content of the InfoWorks library accessible in the current source file.

- Recompile and run the program code by choosing ‘Debug → Run (F5)’ from the main menu or by pressing the arrow symbol in the tool bar on top:



- The started program should perform exactly as the corresponding VBA program that is described in Appendix C, step 5. The only difference is that while the program is running, a Command Prompt window is displayed.  
Note: When selecting a simulation run from the Object Browser dialog that already was simulated, the Simulation Controller will not appear and the selected run will not be invoked.

6. Start the InfoWorks CS application and view the results of the performed simulation (cf. Appendix C, step 6).

As direct referencing to the InfoWorks Type Library ‘infoworks.exe’ failed (see 7.3.1), it was necessary to create a .NET assembly out of the original type library (step 1). Once having set the reference to the newly created assembly (step 4), a C# program could be written that creates and uses an object of the ‘InfoWorksClass’ (step 5). That object offers the methods and attributes that are defined by the InfoWorks API. In the programming example, a prepared InfoWorks run was first selected by use of an InfoWorks-own dialog box and then simulated. See Appendix D for a complete list of methods and properties that are provided by the InfoWorks API.

With knowledge of C# and by using the documentation of the InfoWorks API, the simple example program given in step 5 may be extended in order to perform more complex tasks.

## Appendix E: Methods and properties of the InfoWorks library

The InfoWorks Type Library which describes the InfoWorks Application Programming Interface (API) is embedded in the InfoWorks application file ‘infoworks.exe’. By use of the Type Library Importer tool ‘tlbimp.exe’, contained in the Microsoft .NET Software Development Kit (SDK) the type library has been converted into the .NET assembly ‘InfoWorksLib.dll’. That assembly, which can be inspected by the .NET IL Disassembler tool ‘ildasm.exe’, contains the class ‘InfoWorksClass’. In the following, the methods (see Table 7) and properties (see Table 8) contained in this class, are documented.

As the library does not give only access to InfoWorks CS but, as far as they are licenced, also to the other parts of the InfoWorks software (e.g. InfoWorks RS, InfoWorks WS), class elements can be found that explicitly relate to these parts of the software.

Table 7: Methods of the ‘InfoWorksClass’, being the only class defined in the .NET assembly ‘InfoWorksLib.dll’. The .NET assembly has been created out of the InfoWorks Type Library, which is embedded in the ‘infoworks .exe’ program file. Parameters indicated by ‘[in][out]’ are input parameters that may be changed by the method. Parameters indicated by ‘[out]’ are output parameters, in which values – additional to the (single) return value of the method – are returned. All the other parameters are pure input parameters. The column ‘tested?’ indicates if the method has been tested within this study. The column ‘doc?’ indicates if the method is documented (or at least mentioned) in the InfoWorks API documentation provided by Wallingford Ltd. The last column indicates if the last mentioned document contains corresponding example code in Visual Basic (VB) or C#.

Return type	Name and parameters of method	tested?	doc?	Example?
void	<b>Archive</b> ( string bsPath, int32 bRecursive)	-	-	-
object	<b>BrowseForObjects</b> ( object varTypes, string bsTitle, string bsInitialObject, int32 bMultiSelect, int32 hwnd)	✓	✓	VB C#
void	<b>CheckIn</b> ( string bsNetworkName)	✓	✓	-
string	<b>CheckOut</b> ( string bsParentName, string bsNewName, int32 bBranched)	-	-	-
object	<b>Children</b> ( string bsPath)	✓	✓	VB
void	<b>CompareNetworks</b> ( string bsPath1, string bsPath2, string bsReportPath)	-	✓	-

Return type	Name and parameters of method	tested?	doc?	Example?
void	<b>ComputeStatistics(</b> string bsNetwork, string bsStatsTemplate, string bsFileName) <b>)</b>	-	-	-
void	<b>CopyFromTransportable(</b> string bsPathOfTransportable, string bsNodeToCopyFrom, string bsNodeToCopyTo, string bsObjectType, int32 bWithChildren) <b>)</b>	-	-	-
void	<b>CopyToTransportable(</b> string bsPathOfTransportable, string bsNodeToCopyFrom, string bsNodeToCopyTo, string bsObjectType, int32 bWithChildren) <b>)</b>	-	-	-
void	<b>CreateArchive(</b> string bsPath, int32 bCompact) <b>)</b>	-	-	-
void	<b>CreateTransportable(</b> string bsPathOfTransportable) <b>)</b>	-	-	-
void	<b>Delete(</b> string bsPath) <b>)</b>	✓	-	-
void	<b>EngineeringValidation(</b> string bsPathNet, string bsPathEV, string bsReportPath) <b>)</b>	✓	✓	-
void	<b>Export(</b> string bsPath, string bsFormat, string bsFileName) <b>)</b>	✓	✓	VB
void	<b>ExportGISInteractive(</b> string bsPath, string bsFormat, string bsFolderName, [in] [out] int32& pUserUnits, [in] [out] int32& pUserSystemType, string bsTime, [out] string& pTime, [out] string& pTimeSubDir) <b>)</b>	-	-	-
void	<b>ExportRSResults(</b> string bsSim, string bsSel, string bsFormat, string bsFileName) <b>)</b>	-	✓	VB C#

Return type	Name and parameters of method	tested?	doc?	Example?
void	<b>ExportResults</b> ( string bsSim, string bsFormat, string bsFileName, string bsAttribute, string bsSelection, int32 bGaugeStep, int32 bDownstreamEnd)	✓	✓	VB
void	<b>ExportResultsSnapshot</b> ( string bsSim, string bsFormat, string bsFolderName, string bsTime)	-	✓	VB
void	<b>ExportWNControl</b> ( string bsNetwork, string bsControl, string bsDemandScaling, string bsFileName)	-	-	-
void	<b>ExportWNResults</b> ( string bsSim, string bsFormat, string bsFileName)	-	✓	VB
string	<b>GetLayerLabel</b> ( int32 lLayer, int32 lIncarnation, int32 bNetworkNotResults, int32 bUserSystemType)	-	-	-
string	<b>GetLayerName</b> ( int32 lLayer, int32 lIncarnation, int32 bNetworkNotResults, int32 bUserSystemType, int32 bUseTabNames)	-	-	-
int32	<b>GetNumberOfLayers</b> ( int32 lIncarnation, int32 bNetworkNotResults)	-	-	-
string	<b>GetRegistryString</b> ( string bsSection, string bsEntry)	-	-	-
string	<b>Import</b> ( string bsParent, string bsName, string bsType, string bsFormat, object[]& saFileNames)	✓	✓	VB
void	<b>ImportCCTVData</b> ( string bsNet, string bsFile)	-	-	-

Return type	Name and parameters of method	tested?	doc?	Example?
void	<b>ImportExternalNetwork(</b> string bsCatGrp, string bsGIUDName, string bsPath, string bsName) <b>)</b>	-	-	-
void	<b>InitForTest(</b> int32 nAppType, string bsUserName, string bsPassword) <b>)</b>	✓	✓	VB C#
void	<b>Initialise(</b> int32 nAppType, string bsUserName, string bsPassword) <b>)</b>	✓	-	-
string	<b>LocalRootGUID()</b>	✓	✓	VB
void	<b>MSDEConnectExisting(</b> string bsServer, string bsDatabase, string bsUID, string bsPWD, string bsLocalRoot) <b>)</b>	-	-	-
string	<b>MakeValidPathComponent(</b> string bsPathComponent) <b>)</b>	-	-	-
void	<b>NewMasterDatabase(</b> string NewDatabasePath) <b>)</b>	-	-	-
string	<b>NewObject(</b> string ParentPath, string ObjectName, string ObjectType) <b>)</b>	✓	✓	VB
string	<b>NewObjectTransportable(</b> string bsPathOfTransportable, string ParentPath, string ObjectName, string ObjectType) <b>)</b>	-	-	-
string	<b>NewRSSim(</b> string bsRunPath, string bsSimName, string bsBoundaryPath, string bsRulesPath) <b>)</b>	-	-	-
void	<b>NewRainfallEvent(</b> string bsName, string bsParameters) <b>)</b>	-	-	-
string	<b>NewSim(</b> string bsRunPath, string bsSimName, string bsRainfallOrFlowSurveyName) <b>)</b>	✓	✓	VB

Return type	Name and parameters of method	tested?	doc?	Example?
string	<b>NewWNSim</b> ( string bsRunPath, string bsSimName, string bsDSPPath, string bsCONPath)	-	-	-
void	<b>PolygonAreaTakeOff</b> ( string bsNetworkName, string bsCategoryName, string bsHTMLLogFile, int32 bSystemSurfaceMethod, int32 bUncodedSplit, string bsUncodedSystem, int16 nUncodedSurface)	-	-	-
void	<b>PolygonDeleteAll</b> (string bsNetworkName)	-	-	-
void	<b>PolygonExportCSV</b> ( string bsNetworkName, string bsCsvName, string bsCategoryToExport, int16 nPrecision)	-	-	-
void	<b>PolygonImportCSV</b> ( string bsNetworkName, string bsCsvName, string bsCategoryToImport)	-	-	-
void	<b>PolygonIntersect</b> ( string bsNetworkName, string bsCategoryNameA, string bsCategoryNameB, string bsCategoryNameC, string bsHTMLLogFile)	-	-	-
void	<b>PutRegistryString</b> ( string bsSection, string bsEntry, string bsValue)	-	-	-
void	<b>RSRun</b> (string bsPath)	-	✓	VB
void	<b>Rename</b> ( string bsPath, string bsNewName)	-	-	-
void	<b>Restore</b> ( string bsPath, int32 bRecursive)	-	-	-
void	<b>Run</b> (string bsPath)	✓	✓	VB
void	<b>SetFlagAll</b> ( string bsNetworkName, string bsObjectName, string bsFieldName, string bsFlag)	-	-	-

Return type	Name and parameters of method	tested?	doc?	Example?
void	<b>Update</b> ( string bsPath, string bsFormat, string bsFileName)	-	-	-
void	<b>WNOverviewToCSV</b> ( string bsNetwork, string bsOverviewSet, string bsFileName, int32 bAppendToFile)	-	-	-
void	<b>WNRun</b> (string bsPath)	-	✓	VB

Table 8: Properties of the 'InfoWorksClass' being the only class defined in the .NET assembly 'InfoWorksLib.dll'. The .NET assembly has been created out of the InfoWorks Type Library, which is embedded in the 'infoworks.exe' program file. Some of the properties accept parameters (third column). As opposed to Visual Basic, properties in C# are not able to accept parameters. Thus, when using the assembly from C#, the properties listed here are accessed in the form of corresponding methods named `get_<property>` and `set_<property>` where `<property>` is the name of the property. These methods accept the property parameters as method parameters plus, in case of the set-method, one more parameter representing the new value that the property shall take. For example, in C# the property `Value` with the parameters `string bsPath, string bsFieldName` is accessed via the methods `object get_Value(string bsPath, string bsFieldName)` and `void set_Value(string bsPath, string bsFieldName, object pVal)`, respectively. The column 'tested' indicates if the property has been tested within this study. The column 'doc?' indicates if the property is documented (or at least mentioned) in the InfoWorks API documentation provided by Wallingford Ltd. The last column indicates if the last mentioned document contains corresponding example code in Visual Basic (VB) or C#.

Type	Name of Property	Property parameters	tested?	doc?	Ex.?
string	CurrentArchive	-	-	-	-
int32	DialogParent	-	-	-	-
string	DisplayName	string bsPathName	✓	✓	VB
int32	Exists	string bsName	✓	✓	VB
string	LocalRoot	-	-	✓	VB
string	MasterDatabase	-	✓	-	C#
string	Network	string bsPath	-	✓ <sup>1</sup>	-
string	Parent	string bsPath	✓	✓	VB
string	ShortDisplayName	string bsPath	✓	✓	VB
int32	ShowProgress	-	-	-	-
string	Type	string bsPathName	✓	✓	VB
object	Value	string bsPath, string bsFieldName	✓	✓	VB
string	Version	-	-	-	-

<sup>1</sup> In the API documentation, the method `GetNetwork(string bsPath)` is described.



## Appendix G: Evaluation of simulation results obtained by running an OpenMI-linked system

The following figures refer to the simulations carried out within Scenario B, described in section 5.4.

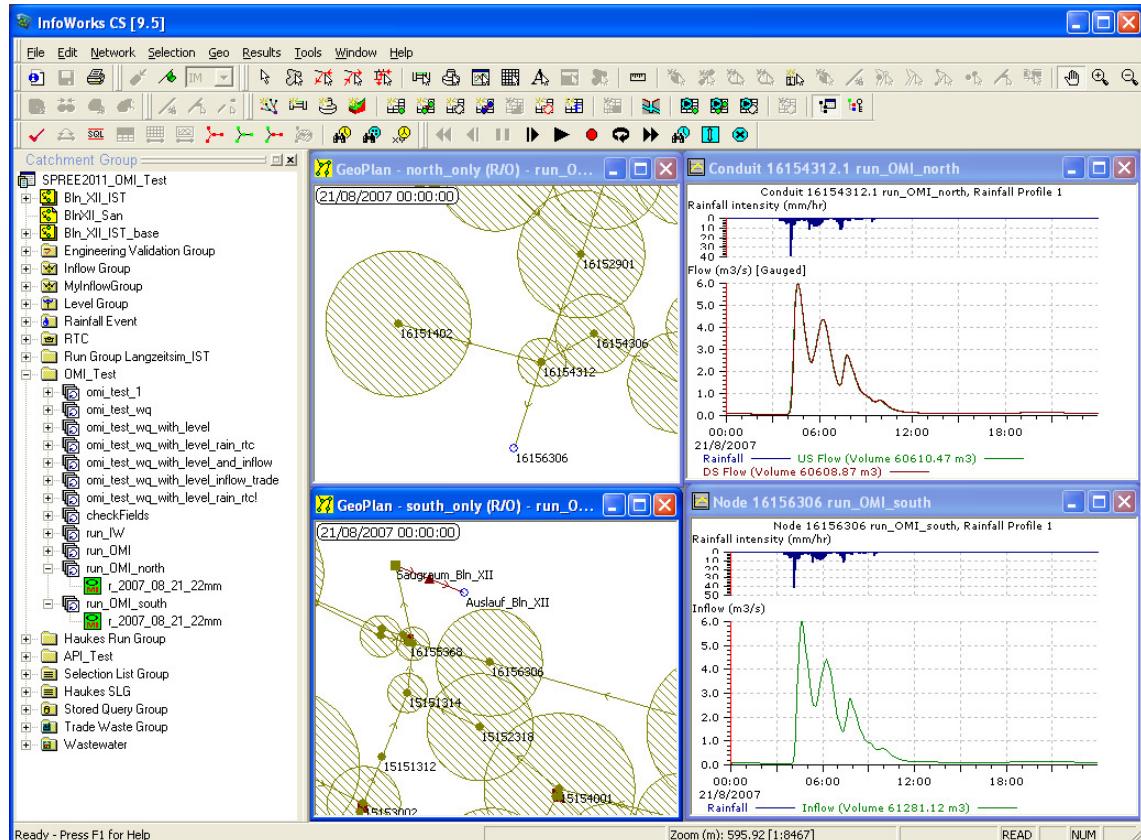


Figure 27: Comparison of simulated flows leaving the northern system (graph in upper right sub-window) with the inflow that has entered the southern system (graph in lower right sub-window).

Data transfer has been realized by OpenMI. Both model Runs ('run\_omi\_north' and 'run\_omi\_south'; see object tree of the Catchment Group, left) have been invoked by the OATC Configuration Editor in which the link between northern and southern network was realized (see 5.4).

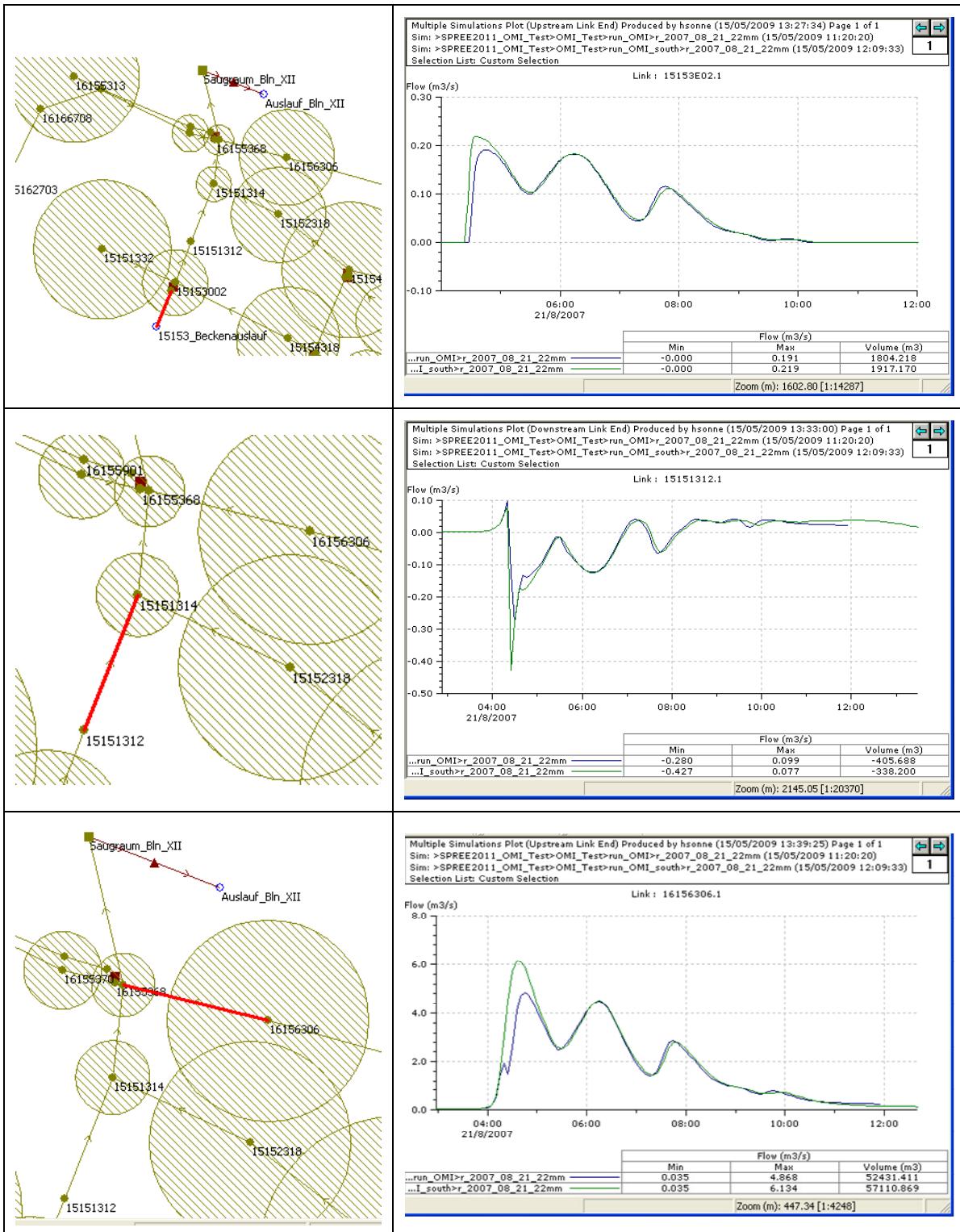


Figure 28: Comparison of flow hydrographs simulated in the original, entire InfoWorks CS model (case 'ORIG', blue graphs) and in the OpenMI-coupled system of two separated sub-models (case 'OPENMI', green graphs). For the sewer section that is directly connected to the outlet, OPENMI calculates a higher peak of the first wave than ORIG (top). Some sewer sections upstream too, a difference in the first (negative) peak can be observed (middle). The deviation of flows is also visible in the conduit that directly follows the node which connects the northern with the southern sub-system (bottom).

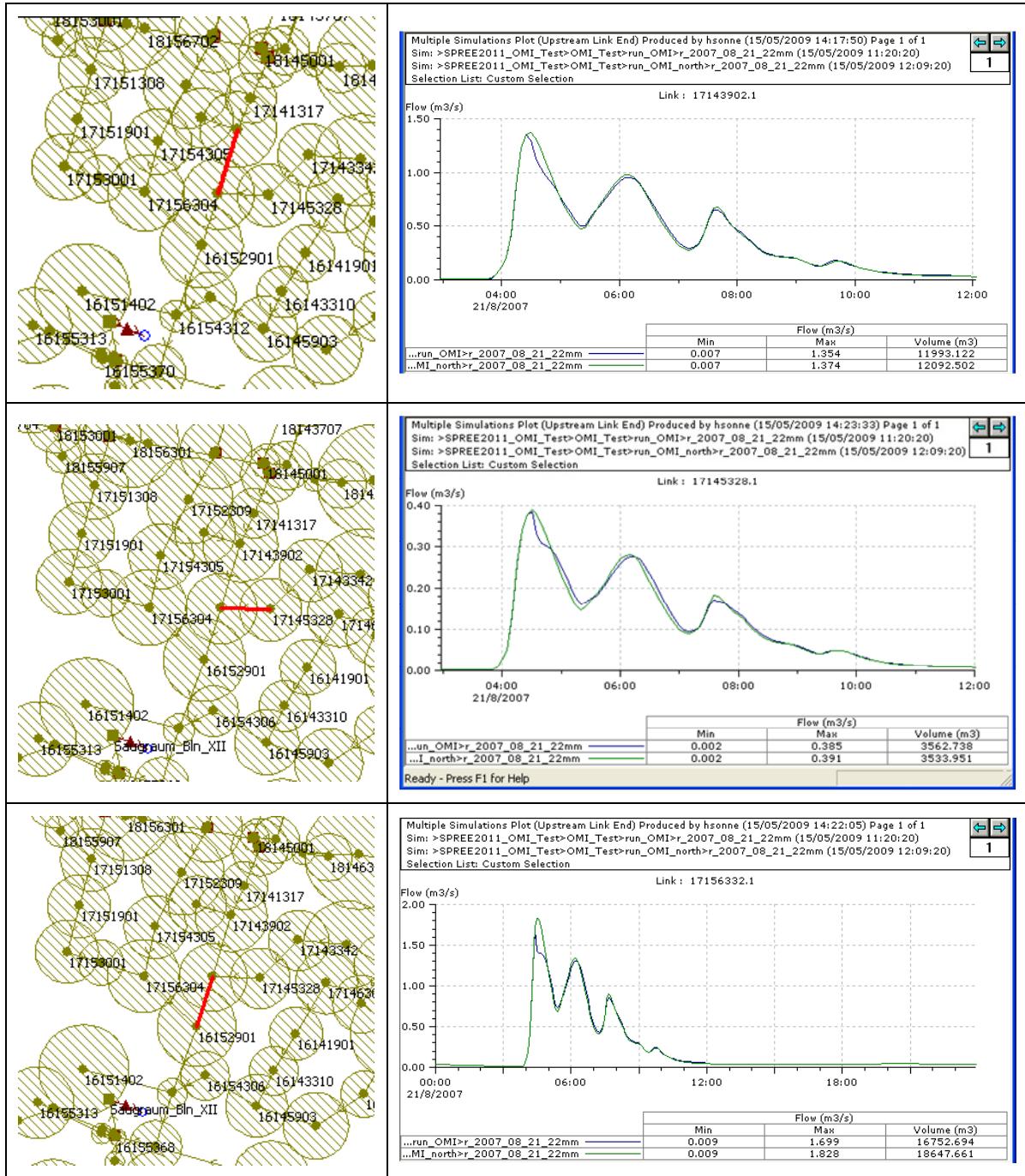


Figure 29: Further backtracking of the differences in flows that were simulated in the OpenMI-linked system (green graphs) and in the original, entire InfoWorks CS model (blue graphs), respectively (see 5.4 in the main text for an explanation).

## Appendix H: OpenMI Demo Models

### Demo Models:

The OpenMI Association Technical Committee (OATC) develops and disseminates demo models which can be downloaded from:

[http://public.deltares.nl/download/attachments/7405691/  
OpenMIDemoModels.zip](http://public.deltares.nl/download/attachments/7405691/OpenMIDemoModels.zip)

Five different OpenMI components (.omi-files) are provided. The components are based on three different model engines (provided as .NET assemblies) which are also given. All these components comply with the OpenMI 1.4 standard and thus, can be run in the OpenMI Configuration Editor 1.4.0.0.

Namespace: Oatc.OpenMI.Tools

.NET Assembly	Component (.omi)
DataMonitor.dll	DataMonitor

Namespace: Oatc.OpenMI.Examples.ModelComponents.SpatialModels

.NET Assembly	Component (.omi)
RiverModel.dll	LargeRiverModel
GroundWaterModel.dll	LocalGroundWaterModel LargeGroundWaterModel RegionalGroundWaterModel

### Models contained in the OpenMI Software Development Kit (SDK):

The Software Development Kit (SDK) of the OpenMI release 1.4.0.0 contains further models. The OpenMI SDK is available at:

[http://sourceforge.net/project/downloading.php?  
group\\_id=136874&filename=Oatc.SDK\\_1.4.0.0.zip](http://sourceforge.net/project/downloading.php?group_id=136874&filename=Oatc.SDK_1.4.0.0.zip)

The SDK contains the following components which depend on corresponding .NET assemblies.

Namespace: org.OpenMI

.NET Assembly	Component (.omi)
Tools.EventLogger.dll	EventLogger.omi
Utilities.AdvancedControl.dll	IterationController.omi OptimizationController.omi SSD.omi

Namespace: Oatc.OpenMI.Examples.ModelComponents

.NET Assembly	Component (.omi)
SimpleRiver.Wrapper.dll	SimpleRiverRhine.omi

However, the required .NET assemblies are not part of the SDK and so, the components are not ready for use. The model engines are delivered in the form of C# source code. Before the OpenMI components can be used, the assemblies need to be built by compiling the corresponding source code.

## Bibliography

- Gijsbers P., Gregersen J., Westen S., Dirksen F., Gavardinas C. and Blind M. (2007a). Part B - Guidelines for the OpenMI (version 1.4), in *OpenMI Document Series*, edited by I. Tindall.
- Gijsbers P., Westen S., Rob Brinkman and Curn J. (2007b). Part F - org.OpenMI.Utilities technical documentation for the OpenMI (version 1.4), in *OpenMI Document Series*, edited by J. Gregersen and P. Sinding.
- Gregersen J., Westen S., Hummel S. and Brinkman R. (2007). Part C - the org.OpenMI.Standard interface specification For OpenMI (Version 1.4), in *The OpenMI Document Series*, edited by P. Gijsbers.
- openmi-life.org (2007a). *Report Defining the Scheldt Use case a : linking hydraulic sewer and river models*, OpenMI-LIFE - Task B1.
- openmi-life.org (2007b). *Report Defining the Scheldt Use case b : linking an upstream nontidal river model to a downstream tidal river model*, OpenMI-LIFE - Task B1.
- Schumacher F., Gebauer U., Pawlowsky-Reusing E., Meier I., Schroeder K., Leszinski M. and Heinzmann B. (2007). *Integrated Sewage Management, Project acronym: ISM, Sub-study: Water quality simulation of river Spree and its canals (reach Charlottenburg) under consideration of combined sewer overflows for a storm event in September 2005*, Final Report, KompetenzZentrum Wasser, Berlin.
- Sonnenberg H. (2008). *Literature Review on the Open Modelling Interface and Environment (OpenMI); Project acronym: SAM-CSO*, Project Report, Berlin Centre of Competence for Water, Berlin, Germany.
- WSL (2007). The InfoWorks/InfoNet COM Interface, all rights reserved Wallingford Software Ltd., 31 pages, document provided by Wallingford Software Ltd. in 2007.