

# An XML Application-Based Interface to Developing Modular System Simulations

Jens Nguema Weisflog

The  
Institute for  
**Systems**  
Research



**A. JAMES CLARK**  
SCHOOL OF ENGINEERING

ISR develops, applies and teaches advanced methodologies of design and analysis to solve complex, hierarchical, heterogeneous and dynamic problems of engineering technology and systems for industry and government.

ISR is a permanent institute of the University of Maryland, within the A. James Clark School of Engineering. It is a graduated National Science Foundation Engineering Research Center.

[www.isr.umd.edu](http://www.isr.umd.edu)

## ABSTRACT

Title of Document: AN XML APPLICATION-BASED  
INTERFACE TO DEVELOPING MODULAR  
SYSTEM SIMULATIONS

Jens Nguema Weisflog, Master of Science in  
Systems Engineering, 2008

Directed By: Dr Ray Adomaitis, Institute for Systems  
Research and Department of Chemical  
Engineering

We introduce a framework for the development of modular lumped and distributed parameter system models, the latter described by boundary value problems. The simulation of such systems requires careful analysis and a rigorous approach to development to provide both accuracy and computational efficiency. We explain the current implementation, which solves such systems in a MATLAB environment using object-oriented programming principles as part of the Modular Distributed Parameter System Analysis and Simulation (MDPSAS) package. We propose a mechanism for creating user-defined simulation elements using a web-based collaborative interface. The creation of a novel semantic vocabulary built into an XML application language called ModSimML is presented as a tool for data structuring and exchange. The development of a schema for the XML application formalizes our data model. The utility of this interface is described via an application to research in Biological Micro-Electro-Mechanical Systems (BioMEMS), whose simulations require assembly from modular components.

AN XML APPLICATION-BASED INTERFACE TO DEVELOPING MODULAR  
DISTRIBUTED PARAMETER SYSTEM SIMULATIONS.

By

Jens Nguema Weisflog

Thesis submitted to the Faculty of the Graduate School of the  
University of Maryland, College Park, in partial fulfillment  
of the requirements for the degree of  
Master of Science in Systems  
Engineering  
2008

Advisory Committee:  
Professor Raymond Adomaitis, Chair  
Professor Mark Austin  
Professor Panagiotis Dimitrakopoulos

© Copyright by  
Jens Nguema Weisflog  
2008

## Preface

This project was intended to help create a solution to a very generic problem. The engine behind MDPSAS, the classes, objects and methods that enable the implementation of a complete simulation are largely abstracted from end-users taken from a larger scientific community. It thus seemed as though the obligatory next step was to develop a user-friendly interface in order to scale-up the applicability of the MATLAB package. We believe this objective was only partially obtained because of the added requirement to understand the functionality and the potential of the XML application defined herein.

## Foreword

This project draws extensively on the work of Dr Ray Adomaitis and the students who have been a part of his team over the years. Although merely an extension of their work, I hope the solutions proposed and the ideas presented here form a basis for future enhancement and perhaps a better understanding of how to build and deploy complex simulations.

## Dedication

I would like to dedicate this work to the people who matter most in my life: my soon-to-be wife Jeanette, my parents and my three sisters, Sheryl, Rebecca and Sarah.

## Acknowledgements

Thank you very much to all those who have helped me complete this project, including my advisor, Ray Adomaitis, my committee members, Dr Mark Austin, a steadying influence, and Dr Panagiotis Dimitrakopoulos. I would also like to thank Sue Frazier of ISR for guiding me through the labyrinth of graduate school policy and deadlines.



# Table of Contents

Preface.....	ii
Foreword.....	iii
Dedication.....	iv
Acknowledgements.....	v
Table of Contents.....	vi
List of Tables.....	viii
List of Figures.....	ix
Chapter 1: Introduction.....	1
Background.....	1
Chemical Process Models.....	1
Distributed vs. Lumped Models.....	2
Literature Review.....	3
Mathematical Modeling of Physical Behavior.....	3
Numerical Techniques.....	4
Chapter 2: Modular System Simulation.....	5
Conceptualization.....	5
System Definition.....	5
A Novel Approach to Modularized System Simulators.....	5
The MDPSAS package.....	6
Classes and Object Formulation.....	7
Methods and Solution Architecture.....	8
Existing Usability Challenge.....	9
Chapter 3: Collaborative Web-Based Interface.....	10
Limitations of the Existing Interface.....	10
Usability and Re-Use.....	10
Separation of Concerns.....	11
Requirements for a new Graphical User Interface (GUI).....	11
Ease of Use.....	11
KPA.....	14
Table 2: Key Process Areas.....	14

Collaboration and Inter-Operability .....	14
Implementation of WikiMDPSAS .....	15
The Wiki Collaborative Environment.....	16
Chapter 4: ModSimML: An XML-Application.....	19
A New Semantic Vocabulary.....	19
Understanding Meta-Data .....	19
Data Structuring .....	20
Document Validation .....	21
The ModSimML Schema .....	21
Data Manipulation .....	23
Chapter 5: Application to Research in BioMEMS .....	25
Sample Simulation: A Tri-Modular System .....	25
Problem Formulation .....	25
Specifying a module .....	26
Specifying a modular system .....	27
Results.....	27
BioMEMS Simulation .....	29
Background.....	29
Abstraction as Modular Systems.....	30
Example: Duct Flow with One Reactive Site .....	30
Path for Future Collaboration .....	35
Chapter 6: Conclusion.....	36
Chapter 7: Future Work .....	38
Bibliography .....	40

## List of Tables

Table 1: Goal / Scenarios for Requirements Generation.....	12
Table 2: Key Process Areas.....	14

## List of Figures

Figure 1: Model Development Life Cycle.....	13
Figure 2: System Architecture.....	17
Figure 3: Development Interface System Class Diagram.....	18
Figure 4: Lumped Parameter Reactor Module Document Type Definition.....	22
Figure 5: Modsys Document Type Definition.....	22
Figure 6: Sequence Diagram of Expected Behavior.....	24
Figure 7: Sample Lumped Parameter System with 3 Modules.....	25
Figure 8: Simple Reactor Module Object XML.....	26
Figure 9: 1Mixer_1Separator_1Reactor Modsys Object XML.....	27
Figure 10: Simulation Convergence: MATLAB Command Window Output.....	28
Figure 11: Arc-length Continuation of Steady-State Solutions.....	29
Figure 12: Simple Duct Flow BioMEMs Model.....	31
Figure 13: Ductrxr Module Object XML.....	32
Figure 14: Simple_Duct_Flow Modsys Object XML.....	32
Figure 15: Extended Module DTD based on New Allowable Content.....	33
Figure 16a: Fluid Flow Profile and Mixing of Components A (Red) and B (Blue). Reactive Site Length = 1000 $\mu\text{m}$ .....	34
Figure 16b: Fluid Flow Profile and Mixing of Components A (Red) and B (Blue). Reactive Site Length = 100 $\mu\text{m}$ .....	34

# Chapter 1: Introduction

## Background

### Chemical Process Models

Chemical process design and simulation tools constitute a tremendous asset for any engineering and analysis department. Depending on a variety of selection criteria, manufacturing and processing plants require such instruments both for design and for operational purposes. If these engineering devices are integrated with business-level applications, the profitability of the processing units is greatly enhanced. Research efforts thus benefit from any improvements in accuracy, reliability, scalability, computational efficiency and usability.

The most commonly available process modeling software packages use flowsheeting interfaces. Examples of such programs include Aspen Plus<sup>1</sup>, PRO/II<sup>2</sup> and CHEMCAD<sup>3</sup>. Although practical for deriving order-of-magnitude estimates of steady-state solutions, they often lack the flexibility to deal with processes that cover a large range of time and length scales [1].

---

<sup>1</sup> Aspen Plus<sup>TM</sup>: product by Aspen Technology

<sup>2</sup> PRO/II<sup>TM</sup>: product by SimSci-Essor

<sup>3</sup> ChemCAD<sup>TM</sup>: product by Chemstations, Inc.

These programs typically propose proprietary black-box models for unit operation simulations, including reactors, heat exchangers, mixers and separators. The models are built on top of a set of routines combined with physical property and reaction databases. This has the advantage of increasing usability but does not lend itself to customization, often necessary in process modeling because of regular deviations from expected behavior. The exposure to customization demands an understanding of the numerical methods deployed to provide accurate multi-scale modeling solutions to process engineering problems [2].

#### Distributed vs. Lumped Models

A certain measure of model complexity can be derived from the dimension of the system's state and parameter spaces. In a lumped parameter system, spatially distributed variables are approximated as single scalars. But many complex systems have an infinite-dimensional state-space and the behavior of the states is described by partial differential equations. Modeling these distributed parameter systems requires treating each dimension continuously, "by a continuous integration, a transform method (e.g. Laplace, Fourier, Bessel) or by discretization" [3]. We will review some these analytical techniques later in this chapter.

## Literature Review

### Mathematical Modeling of Physical Behavior

Chen and Adomaitis (2006) present an overview of the available literature on multi-scale modeling and modular flowsheet tools [2]. The works of Lu and Kaxiras (2004), Maroudas (2000) and Raimondeau and Vlachos (2002) are representative of the mechanisms involved in computing multi-scale solutions for material and energy balances and reaction engineering problems [4,5,6].

Hillestad and Hertzberg (1986) present the three classes of flowsheet simulators: equation-based simultaneous, sequential modular and simultaneous modular. In a modular system, each module has an associated set of algorithms and material properties whereas all equations are solved simultaneously in an equation-based simulator [7].

In order to optimize both flexibility and computational efficiency, Fagley and Carnahan (1990) and Lee and Yoon (1994) note that a modular approach which solves modules sequentially while coupling certain modules into clusters that are solved simultaneously should be used. This strategy would provide more

transparency by allowing the user to make adjustments at the module level without affecting computing performance [8,9].

#### Numerical Techniques

The problems involved in chemical process modeling and optimization carry a range of complexity depending on the application. Of particular interest is the higher end of that spectrum, in dealing with systems of Non-Linear Algebraic Equations (NAEs), Ordinary Differential Equations (ODEs) and Partial Differential Equations (PDEs) as well as non-linear Boundary Value Problems (BVPs). Regardless of type, equation sets can be formulated in matrix form and their solutions could be computed in a variety of ways.

Spectral decomposition techniques permits the representation of a solution, in terms of a sequence of spatially defined functions. Polynomial collocation techniques can be used to discretize Boundary Value Problems. Quadrature-based projection methods help generate collocation-discretized equations that can be solved at the collocation points for the solutions to BVPs. Eigenfunction expansions can solve time-dependant linear BVPs and, in conjunction with the Galerkin projection method, nonlinear problems as well. In both cases, spectral filtering methods allow the minimization of error [10]. For a specified set of parameters, the iterative Newton-Raphson procedure can be used to compute solutions to sets of NAEs and the Runge-Kutta numerical integration is a common approach to solving nonlinear ODEs.



## Chapter 2: Modular System Simulation

### Conceptualization

#### System Definition

The first step towards developing a conceptual model for a system is to understand the problem domain and break the system down into its components. A system should have a set of inputs and outputs, variables, parameters and modeling equations. The line between a variable (subject to dynamic state changes) and a parameter (a system characteristic) is often blurred. However, a computationally efficient system should be neither under-specified nor over-specified (no more variables than modeling equations) to ensure model convergence. Additionally, the most common cause for simulation convergence failure is that the problem is ill posed, which means that user specification of inputs within acceptable ranges is necessary [11].

#### A Novel Approach to Modularized System Simulators

Chen and Adomaitis (2006) propose a novel approach enabling the creation of simulators in an “evolutionary” framework [2]. The framework utilizes a library of routines built in MATLAB to implement global spectral projection and nonlinear

equation solving methods. These methods incorporate many of the numerical techniques mentioned earlier in order to solve physical process modeling problems [12]. It stands to reason that a focus on a flexible and extensible approach is most consistent with an object-oriented programming philosophy. As described by Chen and Adomaitis:

*Simulation problems are broken into modular components, where a module typically consists of a sub-element of a single manufacturing process. (...) The modules can be solved and analyzed individually, which is an asset in tracking the source of solution divergence or other numerical problems. Assemblies of modules can be formed by combining the modular model elements and defining how information is exchanged between modules.*

In order to ensure that a module's information is properly encapsulated, each module in the system should behave like a "black box". In accordance with object-oriented programming best practices, the different objects of the system communicate only by message passing [13]. The standardization of the module-to-module interface provides opportunities for re-use and improves system scalability.

#### *The MDPSAS package*

Commercial simulation software packages are supported by proprietary source code in order to secure intellectual property and ensure company viability. Although

easy to use, this solution is limited from the standpoint of a user who requires more access into the underlying algorithms for either research or performance purposes. The MDPSAS package proposes the benefits that go along with an open, collaborative development environment due to its open source format, at least within the context of University of Maryland research.

An extensive discussion of all the various methods developed within MDPSAS is most certainly better suited for the package's user's guide. The objective of this section is thus merely to succinctly present an overview of the package's salient features.

#### Classes and Object Formulation

As with commercial flowsheeting software, each object in the simulation development environment can be thought of as an instance of a unit operations constructor class. The MDPSAS package allows the creation of objects in the form of instances of the *naemodel*, or non-linear algebraic equation model, class. This superclass has access to all the numerical solution techniques implemented in the MDPSAS library and serves as the template for object or sub-class definition. Indeed, sub-classes of the *naemodel* class can be specified by using ad-hoc decomposition techniques to establish meaningful categories within the problem domain. The formulation of such categories takes advantage of the encapsulation property

conferred by an object-oriented framework. Two examples are provided later as sample applications.

Objects of the same class have the same methods and attributes. Unit operations modules inherit their properties from the *naemodel* superclass, and thus exhibit similar behavior. Objects are linked together by binomial relation statements whose combinations and permutations uniquely define an integrated modular simulation. In terms of the implementations, objects are related through instantiations of a *relation* class and object definitions, along with these relations, can be grouped together into an instance of a *modsys*, or modular system, class. Therefore, system integration is the procedural combination of component objects into a modular system for which steady-state simulation solutions can be computed.

#### Methods and Solution Architecture

From an overall standpoint, each module class has a residual method, which must be overloaded when defining subclasses of *naemodel* because the modeling equations are unique to the derived class. Modules classes also have analysis and debugging tools such as plotting methods. The data types of the parameters and variables which uniquely define an object are instances of an associative array class (*assocarray*) which holds both entity name and value and encapsulates methods of its own. As specified earlier, a pool of spectral projection methods are available to help solve chemical engineering problems of various nature, but generically identified as

boundary value problems (BVPs). These methods are used to discretize complex partial differential equations into simpler forms for which solver methods were either developed or already available in recent MATLAB library releases. The constructor method of the *modsys* class serves as the coordinator of the information exchange between modules and a repository to which the solver methods regularly update variable values on the simulation's path to convergence. This component architecture ensures flexibility in manipulating child classes while providing the structure required for solution usability [2,12].

#### Existing Usability Challenge

One of the outstanding challenges of this framework lies in the enabling of simulation development in a method-neutral environment. Conceivably, end-users could opt for different approaches to modular system simulation building. In a top-down methodology, the target system is defined as whole and then decomposed into several layers to maximize rigidity, whereas in a bottom-up approach the building blocks are created piece by piece until a modular system can be defined which logically connects the components [14]. This perceived need for both flexibility and clarity requires an open development environment impartial to user expertise, to the extent that this is possible, and preferred model building approach. We focus on the design of a development interface, which potentially addresses these concerns.

## Chapter 3: Collaborative Web-Based Interface

### Limitations of the Existing Interface

#### Usability and Re-Use

The MDPSAS interface uses a standard MATLAB hierarchical structure to organize the source code into classes. The single development interface is the MATLAB programming environment, intended to provide a means for engineers familiar with the MATLAB programming language to deploy accurate simulations based on the efficient use of a library of numerical algorithms. Indeed, MATLAB has a strong presence among engineers and applied mathematicians, which makes the product an attractive platform [15]. Although many computational examples have demonstrated solution robustness, the need for a more interactive, seamless environment for development and distribution of simulation packages has been identified. Because of the desired range and the expected extensibility of the application, the design methodology should actively encourage user engagement and code re-use. From the user's standpoint, the existing interface for simulation development lacks some of the visibility features that were intended to counteract the limitations of closed source flowsheeting tools.

## Separation of Concerns

The central tenet of the separation of concerns principle in software engineering holds that abstraction should be used effectively to hide software complexity. The application is assumed to chiefly address a basic concern for which the underlying algorithms were derived (in this case, the spectral projection and nonlinear equation solution implementations). So-called special purpose concerns add extra functionality and improve the performance and usability of the core algorithms. These additional concerns are separated from the basic concern in an effort to take all stakeholders into consideration and make the source code easier to write and modify [16]. We believe an extra layer of abstraction, provided by a familiar, easy to manipulate interface, is an achievable objective, which would serve as an enhancement to the MDPSAS development environment.

### *Requirements for a new Graphical User Interface (GUI)*

#### Ease of Use

There are two goal-scenario entities that drive the generation of interface requirements.

	Goal	Scenario
Module Goal-Scenario	To create a new module	<ul style="list-style-type: none"> <li>- Specify object parameters, variables and initial guesses</li> <li>- Construct</li> <li>- Obtain steady-state solution</li> </ul>
System Goal-Scenario	To create a new system	<ul style="list-style-type: none"> <li>- Specify object instances and relations</li> <li>- Construct</li> <li>- Obtain steady-state solution</li> </ul>

Table 1: Goal / Scenarios for Requirements Generation

The path to a developed system is complete when all individual objects and relations are fully specified. Specification entails that the user states the module (*naemodel* object) parameters and initial variable values as the attributes that define the instance of the template class. The center of data interchange among objects of a simulation solution is the modular system, which acts as a mediator between objects and the solution space. As mentioned previously, the user must specify properties of the binomial relations between objects in a multi-component system in order to instantiate the *modsys* class. Essentially, this involves detailing the flow path between consecutive objects.

The previously defined goal scenario statements generalize the typical use-cases for the MDPSAS simulation development framework. Creating a modular system requires undergoing an ad-hoc building process but the life cycle can loosely be viewed as an iterative pseudo-prototype model of system development because of the emphasis on extensibility.



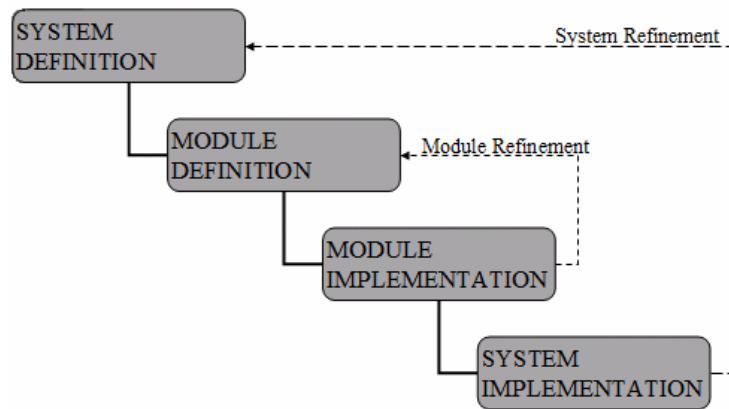


Figure 1: Model Development Life Cycle

We derive requirements from our assessment of design objectives and based on the limitations of the current solution environment. We decompose them into four Engineering Key Process Areas (KPA), which correspond to a group of related activities within the Capability Maturity Model [17]. These interface requirements are intended to resolve system usability. Consequently, the KPAs are grouped into categories that mimic the conceptual system life cycle. The core focus is thus to improve usability during the conceptual phases of analysis, development, testing and maintenance.

<i>KPA</i>	Id	Requirement Description
Analysis	1.1	Required Inputs Shall Be Made Explicit
	1.2	Outputs Shall Be Clearly Available to the User
	1.3	Simulation Overview (Path to Solution) Shall Be Provided
Development	2.1	Minimum Required User Involvement Shall Be No More Than 2 Steps (Define / Run)
	2.2	Module Definition Shall Require No More Than 5 Data Entries
	2.3	System Definition Shall Require No More Than 5 Data Entries
	2.4	Simulation Execution Shall Provide Feedback at No Less Than 5 Second Intervals
Testing	3.1	Module and System Definition Validation Shall Require No More Than 5 Steps of User Involvement
	3.2	System Shall Provide Result Analysis Mechanisms to User
	3.3	System Shall Provide User-Friendly Debugging Environment
Re-use	4.1	System Shall Encourage Multi-User Collaboration
	4.2	System Shall Permit Easy Access
	4.3	System Shall Have Good Documentation
	4.4	Source Code Shall Have Clear Comments Throughout

Table 2: Key Process Areas

#### Collaboration and Inter-Operability

In addition to usability requirements, we propose that interface enhancement should address the increasingly dominant concerns of collaboration-friendliness and inter-operability. Although there are many competing viewpoints to take into consideration, the combination of available resources and emphasis on user involvement lead to the selection of a web-based technology for the development of a new user interface.

## Implementation of WikiMDPSAS

Internet access is already ubiquitous and yet still growing. Familiarity with web technology is also high, especially amongst the research community, the project's chief target population. Many packages, including Mu's PDE.mart, capitalize on advances in network-based computing to create equation solvers with a web-browser interface [18].

Free resource material is readily available, along with open solutions to help implement this project. This includes an active and available online development community, adding to the attractiveness of a web-oriented solution. These different factors contribute to the conjecture that, in the long term, a solution that focused on web-awareness would yield the most benefits.

The proposed interface is inspired by two concepts closely associated with web 2.0: extensibility and collaboration. We choose to build simulation models on top of the Extensible Markup Language (XML) by developing a novel XML application named ModSimML. ModSimML defines and describes the human-readable semantic vocabulary that will be used for data interchange, simulation execution and data archiving. The implementation makes use of XML parsing functionalities available in MATLAB and inherited from the publicly available (open source) Xerces-JAVA

XML Parser<sup>4</sup>. ModSimML specifies the format of the inputs and outputs that will be used to create modules and develop modular systems.

This approach may not be as aesthetically pleasing as the graphical depiction of simulation components that commercial flowsheet software generally provides. Nonetheless, the ability to combine an XML format with scalable-vector graphics, another XML application, leaves open the possibility to use XML Stylesheet Transformations to create add-ons for graphical purposes. This methodology was selected because it lends itself more easily to simulation data interchange on the web. Indeed, module and system information stored in XML format are posted on a Wiki designed for both project archiving and information exchange, located at <http://wikimdpsas.wikidot.com/>.

#### The Wiki Collaborative Environment

A wiki is a website that allows users to easily create new pages for information sharing purposes. Object pages are instantiated from templates much like objects are built from classes in object-oriented programming. Consequently, users can simply post data for recently configured modules or modular systems and the templatization of the online repository ensures that the new resource is rapidly shared among peers. The single semantic syntax ensures data consistency and integrity. The form-like appearance of XML increases usability since electronic forms are often

---

<sup>4</sup> Information Available at: <http://xerces.apache.org/xerces2-j/>

considered to be “the most natural form of system description” (Vilz et al, 2006) due to end-user familiarity and ease of transition to a semantic model [19]. Additionally, the XML platform encourages the development of plug-ins, add-ons, applets and other third-party software built on top of a single application interface format.

Valid data posted on the Wiki can be extracted in a quasi-automatic manner into the simulation constructor routines for model execution. At the physical layer, this system has a client-server architecture with the specification data held on the server while the source code and therefore the execution is operated on the client.

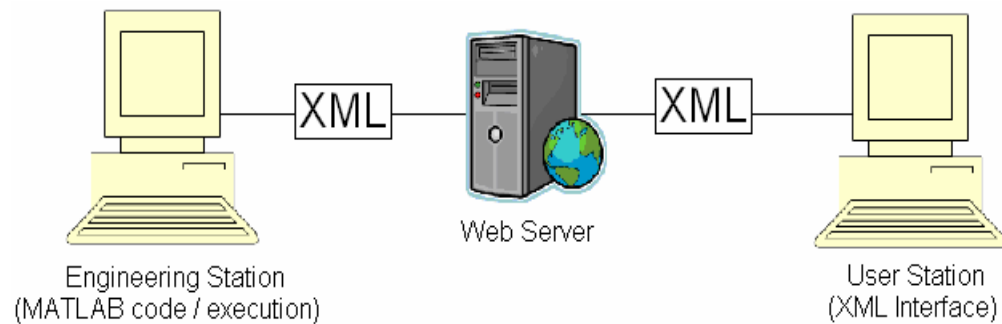


Figure 2: System Architecture

Conceptually, the complete interface system is the combination of a problem space mapped to an XML implementation structure and a solution space mapped to a MATLAB implementation structure.

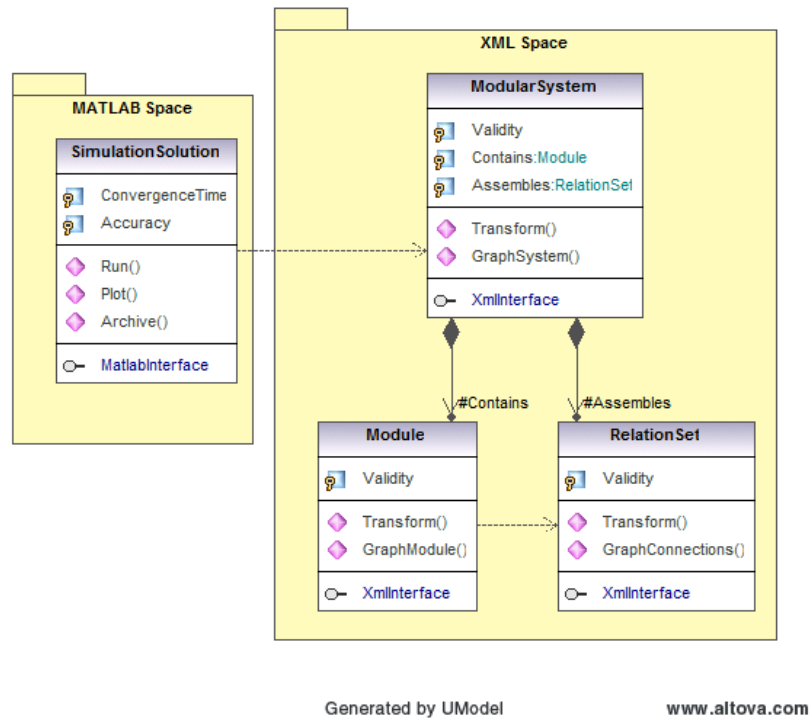


Figure 3: Development Interface System Class Diagram

We believe the Wiki’s accessibility, features and simplicity of use favorably reconcile the three components contributing to user satisfaction with a web interface: features in the web environment, user’s information seeking tasks and information seeker characteristics. Indeed, the web environment features are essentially limited to the information seeking tasks (the creation and editing of modules and objects) and we propose that an information seeker, typically an individual involved in the project, should be properly motivated by the simplicity and the lack of feature creep [20].

## Chapter 4: ModSimML: An XML-Application

### *A New Semantic Vocabulary*

#### Understanding Meta-Data

We formulate an ontology as part of the ModSimML application which captures the salient features of the objects we wish to represent. This approach is consistent with the collaboration-focused software engineering practices, which are becoming the norm in today's economy [21]. The specification of the XML application is drawn from an understanding of the meta-data, or “data about data”, which conceptually defines the problem domain. In the case of chemical engineering boundary value problems and their simulation solutions, the meta-data would include such tags as “modules”, “modular systems”, and “relations”. The main advantage of this data model-oriented framework is the extensible nature of the markup language, which allows us to add data structures as the supported application matures. In addition, XML is an open standard adopted by the W3C consortium and consequently has pre-built parsing and validation functionalities which simplify software development. It is fast becoming an ideal format for capturing and representing data formats [22].

## Data Structuring

The XML platform proposes object abstraction into a hierarchical nodal structure [23]. In light of the conceptual need to identify components as either modules or modular systems, we establish two XML document templates whose respective root nodes are <module> and <modsys>.

The <module> tree has child node <title> as its unique identifier, which is conceptually akin to the primary key of a relational database. The other child nodes, in the context of lumped parameter systems, are <feed> and <reactant>, thereby specifying the flow path. The cardinality for such systems is one module to one or more feeds and one module to one or more reactants. The feed and reactant are flows whose child nodes are the upstream and downstream properties of <flowrate> and a <componentArray> of one or more <component> child nodes. A <reaction> may be specified as an attribute, and therefore a child node, of the feed because of the general position that parameters are inputs to the objects in contrast to calculated variables, which are outputs.

The <modsys> tree also has a unique <title> identifier. Its child nodes are <modules>, encompassing the subset of available modules with the system, and <connections>, which hold the information about the integrated model connectivity grid. The cardinality for a valid modular system is one <modules> to two or more



<module> child nodes, and one <connections> parent node to one or more <connection> child nodes.

### Document Validation

A simulation solution cannot be computed unless a valid system configuration is specified. XML validation involves matching configuration documents to application language-specific schemas using a variety of widely available tools. The two most common XML schema languages are the Document Type Definition (DTD) and the aptly named XML Schema language. Both schema specifications define the allowable document content based on expected hierarchy and data attributes. Among the freely available tools, user-created XML documents can be validated on the web using Brown University's STG XML Validation form<sup>5</sup>.

#### The ModSimML Schema

We propose the following Document Type Definition files for the extensible ModSimML application language. The first document describes the allowed content of a lumped parameter reactor module and the second document applies to all modular system definitions.

---

<sup>5</sup> Located at: <http://www.stg.brown.edu/service/xmlvalid/>

```
Module - Notepad
File Edit Format View Help
<!ELEMENT Module (title | feed | reactant)* >
<!ELEMENT title (#PCDATA)>
<!ENTITY % flow "(flowrate | componentArray | reaction)*">
<!ELEMENT feed %flow;>
<!ELEMENT reactant %flow;>
<!ELEMENT flowrate (#PCDATA)>
<!ATTLIST flowrate
  id CDATA #REQUIRED>
<!ELEMENT componentArray (component*)>
<!ATTLIST componentArray
  id CDATA #REQUIRED>
<!ELEMENT component (#PCDATA)>
<!ATTLIST component
  idx CDATA #REQUIRED>
<!ELEMENT reaction (intrinsicRate)>
<!ELEMENT intrinsicRate (#PCDATA)>
```

Figure 4: Lumped Parameter Reactor Module Document Type Definition

```
Modsys - Notepad
File Edit Format View Help
<!ELEMENT Modsys (title | modules | connections)* >
<!ELEMENT title (#PCDATA)>
<!ELEMENT modules (module*)>
<!ELEMENT module "(type | name)*">
<!ATTLIST module
  id CDATA #REQUIRED>
<!ELEMENT type (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT connections (connection*)>
<!ELEMENT connection "(upstream | downstream)*">
<!ATTLIST connection
  id CDATA #REQUIRED>
<!ELEMENT upstream (#PCDATA)>
<!ELEMENT downstream (#PCDATA)>
```

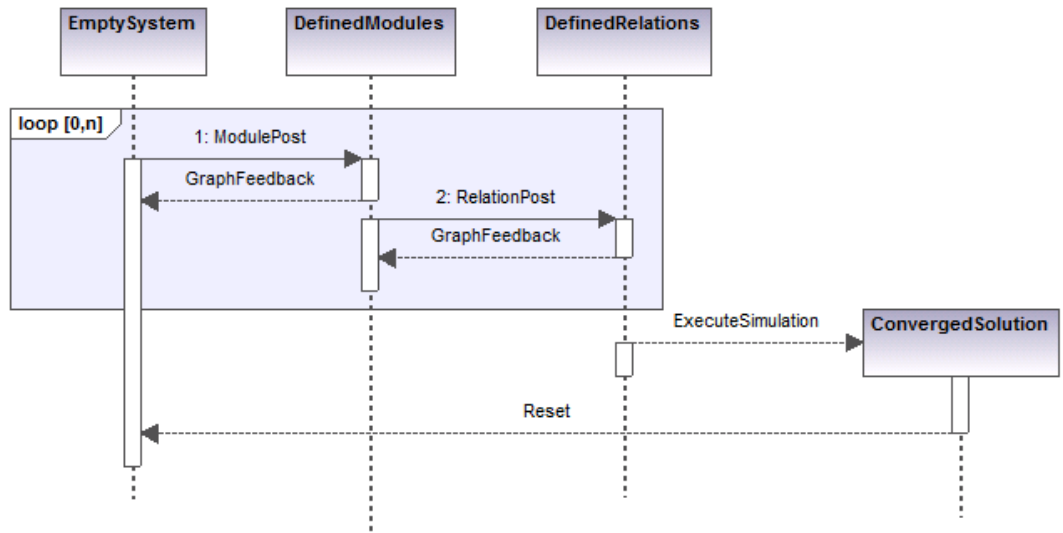
Figure 5: Modsys Document Type Definition

The two DTD files explicitly define the allowed data structures as previously described. Note that the level of detail required for the schema to be complete

includes the definition of such attributes as the “id” or index “idx” of elements. Additionally, because of the extensible nature of the XML data format, data structures can be added to the ModSimML schema specification as a better understanding of how parameters affect simulation results forces an expansion of the XML application.

### Data Manipulation

Based on the data posted on the web, we extract information using XML Stylesheet Language (XSL) operations to create a document that is valid against ModSimML’s schema. Assuming valid configuration format, the simulation development environment has parsing capabilities implemented in MATLAB that are used to instantiate the constructor classes of the user-specified objects. Thus modular systems described in ModSimML are parsed and simulated using the MDPSAS package tools and utility tools can then be used to analyze and archive results. A simulation solution is executed by running the program from the MATLAB execution prompt.



Generated by UModel

www.altova.com

Figure 6: Sequence Diagram of Expected Behavior

## Chapter 5: Application to Research in BioMEMS

### Sample Simulation: A Tri-Modular System

#### Problem Formulation

A sample illustrative system combines a simple reactor, a simple mixer and a simple separator. Each unit can be viewed as an instance of an object of type `simplerxr`, `simplemxr` and `simplesep`, each of which is an implementation of the `naemodel` template. Each module is considered a lumped parameter system, since no variables are spatially distributed. The overall system has a feed stream  $I$  and a product stream  $P$ .

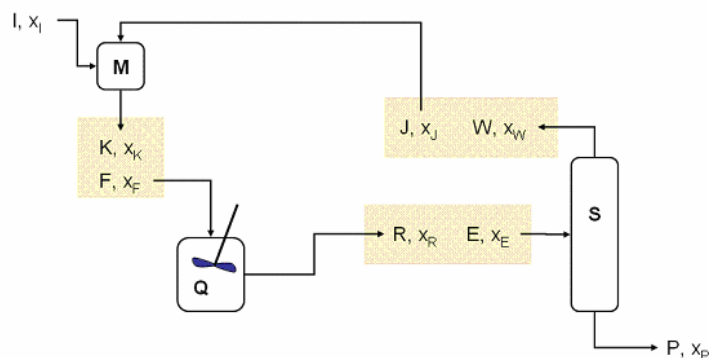


Figure 7: Sample Lumped Parameter System with 3 Modules

## Specifying a module

We declare a `simplerxr` module whose data attributes as specified in variable `var` and parameter associative arrays are as follows:

```
var    = assocarray({'xR' [0.1; 0.45; 0.45] 'R' 1});  
param = assocarray({'xF' X 'F' 1 'k' 50});
```

The parameter value `X` associated to the name `xF` is an array built from data posted on the wiki in a module title `Med_Feed_Flow`. After XSL transformations parse the XHTML page posted on the wiki, the XML configuration file valid against the ModSimML schema for lumped parameter systems presented earlier is as follows:



```
<?xml version="1.0" encoding="utf-8" ?>  
- <Module>  
  <title>Med Feed Flow</title>  
  - <feed>  
    <flowrate id="F">15</flowrate>  
    - <componentArray id="xF">  
      <component idx="1">1</component>  
      <component idx="2">0</component>  
      <component idx="3">0</component>  
    </componentArray>  
    - <reaction>  
      <intrinsicRate>50</intrinsicRate>  
    </reaction>  
  </feed>  
  - <reactant>  
    <flowrate id="R">1</flowrate>  
    - <componentArray id="xR">  
      <component idx="1">1.000000e-001</component>  
      <component idx="2">4.500000e-001</component>  
      <component idx="3">4.500000e-001</component>  
    </componentArray>  
  </reactant>  
</Module>
```

Figure 8: Simple Reactor Module Object XML

## Specifying a modular system

Similarly, we establish relationships which define the modular system by utilizing information stored on the Wiki in a modular simulation titled 1Mixer\_1Separator\_1Reactor. The corresponding configuration file, valid against the modsys schema, is:

```
- <Modsys>
  <title>1Mixer_1Separator_1Reactor</title>
  - <modules>
    - <module id="1">
      <type>Reactor</type>
      <name>High_Feed_Flow_r</name>
    </module>
    - <module id="2">
      <type>Mixer</type>
      <name>High_Feed_Flow_m</name>
    </module>
    - <module id="3">
      <type>Separator</type>
      <name>High_Feed_Flow_s</name>
    </module>
  </modules>
  - <connections>
    - <connection id="1">
      <upstream>2</upstream>
      <downstream>1</downstream>
    </connection>
    - <connection id="2">
      <upstream>1</upstream>
      <downstream>3</downstream>
    </connection>
    - <connection id="3">
      <upstream>3</upstream>
      <downstream>2</downstream>
    </connection>
  </connections>
</Modsys>
```

Figure 9: 1Mixer\_1Separator\_1Reactor Modsys Object XML

## Results

After execution, the following figure depicts the MATLAB command window output during the convergence of the modular system when using the Newton method.

```

simultaneous solution
assocarray object "modules":
M:      mixer object: 1-by-1
Q:      simplrxr object: 1-by-1
S:      simplesep object: 1-by-1
Update/residual norm = 2.5679 / 1.1459
Update/residual norm = 0.52834 / 1.1279
Update/residual norm = 0.87841 / 1.0553
Update/residual norm = 1.1393 / 0.96661
Update/residual norm = 1.8569 / 0.56018
Update/residual norm = 1.1161 / 0.2937
Update/residual norm = 1.2003 / 0.056872
Update/residual norm = 0.32755 / 0.0041311
Update/residual norm = 0.10055 / 0.00069251
Update/residual norm = 0.018483 / 1.8489e-005
Update/residual norm = 0.0004521 / 1.0924e-008
Update/residual norm = 2.7351e-007 / 3.7491e-015
Update/residual norm = 6.7368e-014 / 4.6775e-016

simplrxr object "Q"
Variables
-----
xR: 0.1000
    0.6917
    0.2083
R : 2.4008
Parameters
-----
k : 50
xF: 0.5165    0.4835    -0.0000
F : 2.4008
simplesep object "S"
Variables
-----
xw: 0.1714
    0.8286
    0

XP:      0
        0.5000
        0.5000
W : 1.4008
P : 1
Parameters
-----
f : 0.5
xE: 0.1000
    0.6917
    0.2083
E : 2.4008
mixer object "M"
Variables
-----
xK: 0.5165
    0.4835
    -0.0000
K : 2.4008
Parameters
-----
xI: 1
    0
    0
I : 1
xJ: 0.1714
    0.8286
    0
J : 1.4008

```

Figure 10: Simulation Convergence: MATLAB Command Window Output

The results from arc-length continuation of the steady-state solutions is also depicted:



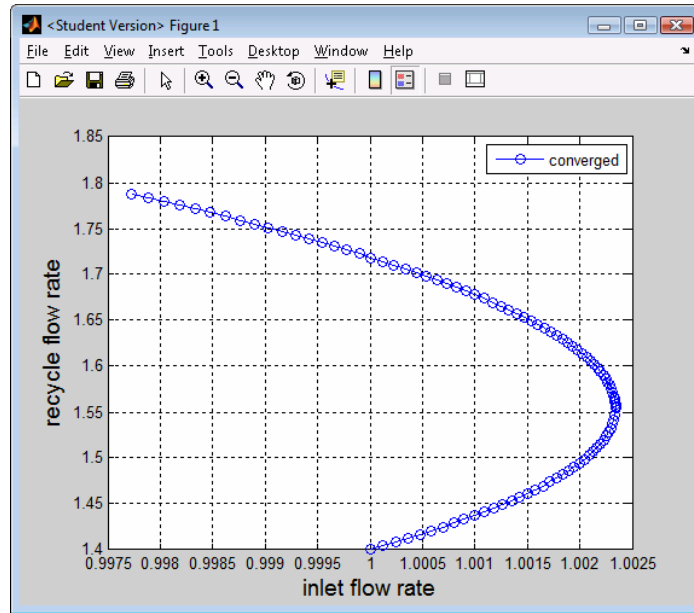


Figure 11: Arc-length Continuation of Steady-State Solutions

### BioMEMS Simulation

#### Background

Micro-Electro-Mechanical systems (MEMs) is the term conferred to a range of procedures and applications generally targeted to embedded systems at the microscopic scale. BioMEMS in particular is a subcategory of the field whose related technologies are confined to the bioengineering field. Currently research in BioMEMs aims at developing applications for drug delivery systems, biomedical sensors and bio-chemical analytical instruments [24]. In the case of biosensors, for instance, they offer the promise of faster bio-agent detection at a lower cost per test, though currently existing products have yet to gain market share on older, and still popular laboratory-based assay techniques [25].

## Abstraction as Modular Systems

A typical BioMEM chip is a network of micro-fluidic channels etched by various processes to behave as an integrated circuit of bio-chemical reactions. A bioMEM system can be conceptually decomposed into duct sections, typically with rectangular cross-sections, and identified by their reaction sites. These sections are sequentially ordered as an arrangement of reacting ducts such that we can formulate 2-dimensional models for each section. Because the state variables and some of the parameters used to define boundary conditions require spatially discretized definitions, additional simulation inputs defining the modules are necessary relative to the lumped parameter modeling modules. For example, in addition to feed and reactant specifications, we declare the length of cross-sections, the use of a reactive site and the number of collocation points employed for the discretization of partial differential equations.

### Example: Duct Flow with One Reactive Site

We pose an illustrative problem where two non-reacting duct modules serve as the inlet and outlet to two middle ducts, one without a reactive site and one with a reactive site, each contiguously connected in series. A single reactant is introduced as a pure component into the system with a simple first-order reaction as follows:

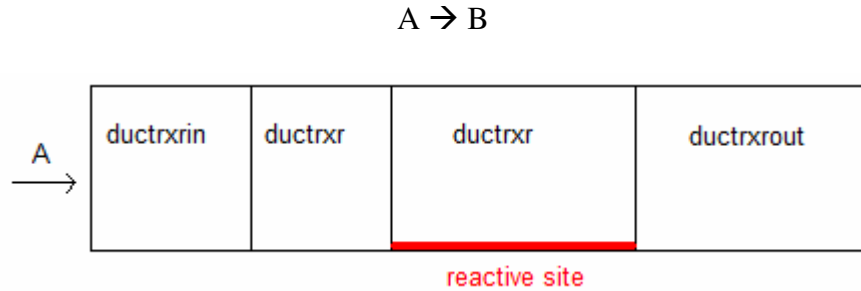


Figure 12: Simple Duct Flow BioMEMs Model

The inlet module is a pre-reactor section used chiefly to set the boundary conditions of the overall system and the outlet similarly sets the overall system's boundary. The short pre-reaction section allows us to concentrate a large number of collocation points just upstream of the reactive section, resulting in better numerical performance of the overall simulation.

The configuration XML documents for one module (ductrxr) and the modular system (Simple\_Duct\_Flow) are as follows:

```

<?xml version="1.0" encoding="utf-8" ?>
- <Module>
  <title>ductrxr</title>
  - <feed>
    <flowrate id="A">1</flowrate>
    - <componentArray id="xA">
      <component idx="1">1</component>
    </componentArray>
    - <reaction>
      <reactiveSite>yes</reactiveSite>
    </reaction>
  </feed>
  <collocPoints>15</collocPoints>
  <startLoc dim="y">0</startLoc>
  <endLoc dim="y">1000</endLoc>
</Module>

```

Figure 13: Ductrxr Module Object XML

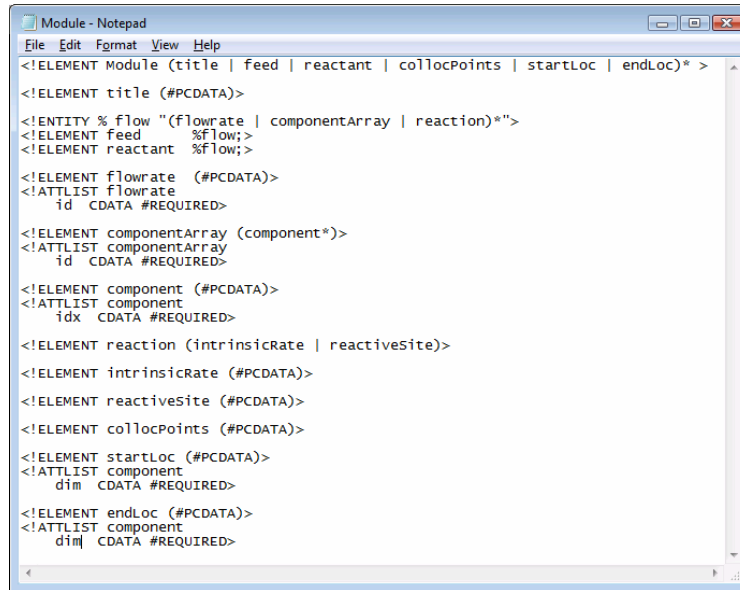
```

- <Modsys>
  <title>Simple_Duct_Flow</title>
  - <modules>
    - <module id="1">
      <type>Duct_Reactor_in</type>
      <name>ductrxrin</name>
    </module>
    - <module id="2">
      <type>Duct_Reactor_r</type>
      <name>ductrxr</name>
    </module>
    - <module id="3">
      <type>Duct_Reactor_nr</type>
      <name>ductrxr_nr</name>
    </module>
    - <module id="4">
      <type>Duct_Reactor_out</type>
      <name>ductrxr_out</name>
    </module>
  </modules>
  - <connections>
    - <connection id="1">
      <upstream>1</upstream>
      <downstream>2</downstream>
    </connection>
    - <connection id="2">
      <upstream>2</upstream>
      <downstream>3</downstream>
    </connection>
    - <connection id="3">
      <upstream>3</upstream>
      <downstream>4</downstream>
    </connection>
  </connections>
</Modsys>

```

Figure 14: Simple\_Duct\_Flow Modsys Object XML

Note that the appearance of new elements in the configuration file for ductrxr requires an extension to the module Document Type Definition file in order to check the validity of the XML against the ModSimML schema. The extended schema is provided.



```
Module - Notepad
File Edit Format View Help
<!ELEMENT Module (title | feed | reactant | collocPoints | startLoc | endLoc)* >
<!ELEMENT title (#PCDATA)>
<!ENTITY % flow "(flowrate | componentArray | reaction)*">
<!ELEMENT feed %flow;>
<!ELEMENT reactant %flow;>
<!ELEMENT flowrate (#PCDATA)>
<!ATTLIST flowrate
  id CDATA #REQUIRED>
<!ELEMENT componentArray (component*)>
<!ATTLIST componentArray
  id CDATA #REQUIRED>
<!ELEMENT component (#PCDATA)>
<!ATTLIST component
  idx CDATA #REQUIRED>
<!ELEMENT reaction (intrinsicRate | reactiveSite)>
<!ELEMENT intrinsicRate (#PCDATA)>
<!ELEMENT reactiveSite (#PCDATA)>
<!ELEMENT collocPoints (#PCDATA)>
<!ELEMENT startLoc (#PCDATA)>
<!ATTLIST startLoc
  dim CDATA #REQUIRED>
<!ELEMENT endLoc (#PCDATA)>
<!ATTLIST endLoc
  dim CDATA #REQUIRED>
```

Figure 15: Extended Module DTD based on New Allowable Content

The following results were obtained for the 2-dimensional flow profile and spatial distribution of the feed component along the duct, with the variation of one design parameter (the reactive site length) based on information posted on the Wiki.

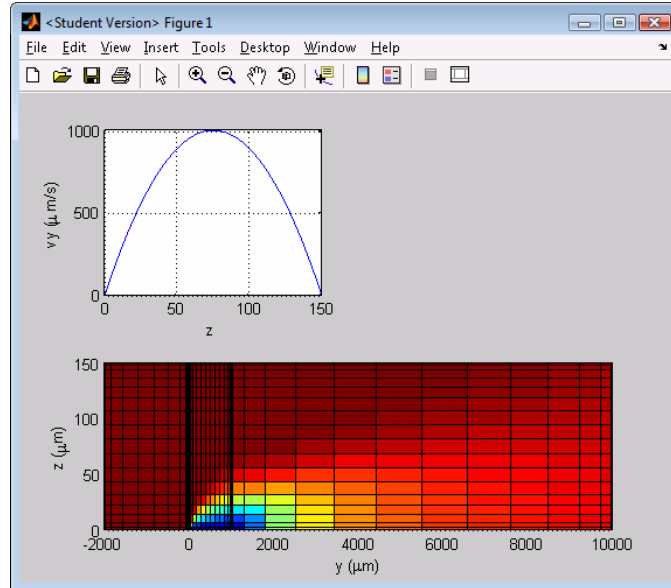


Figure 16a: Fluid Flow Profile and Mixing of Components A (Red) and B (Blue). Reactive Site Length = 1000  $\mu\text{m}$

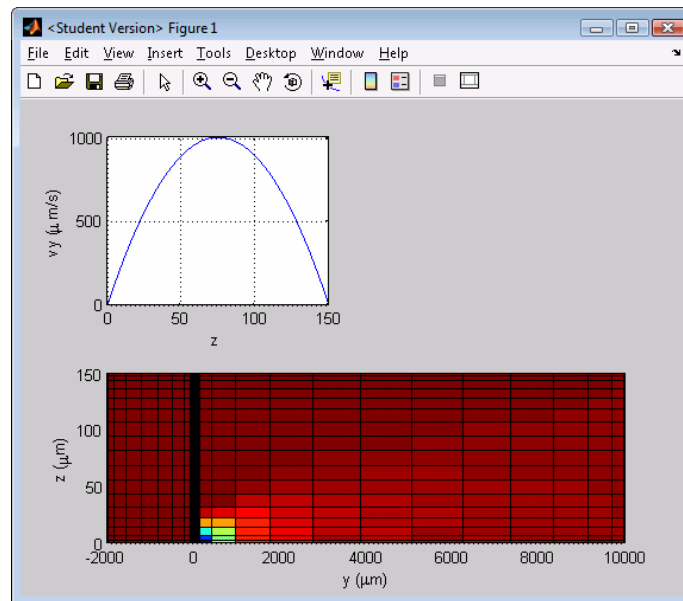


Figure 16b: Fluid Flow Profile and Mixing of Components A (Red) and B (Blue). Reactive Site Length = 100  $\mu\text{m}$

## Path for Future Collaboration

The results of different simulation runs as well as information about modules to test can be posted on WikiMDPSAS. A user can simply select one of two component types (“ModCreator” for modules and “SimCreator” for modular systems). The user then specifies the properties of a new object by using the template provided in the editor. Using the title of the components as inputs, multiple simulation runs can be computed and contrasted.

## Chapter 6: Conclusion

This project intended to develop a framework for modular system simulation development operating from a web-based interface. The simulations involved concerned mostly boundary-value problems, weighted residual methods, and nonlinear equation solving techniques, implemented in requiring implemented in MATLAB as part of the MDPSAS package. Problem specification involved the definition of component modules and the relationships between them, which uniquely describes the modular system.

The objective was to improve usability by reducing the semantic gap between users and developers. Despite a robust implementation, from the end-user's viewpoint, the main limitations of the current MDPSAS package were in the interface. The focus on the interface system forced us to think of the transformations required to enable cross-viewpoint cooperation, with the principal stakeholders here being the researcher and the developer.

The solution was found in the specification of an Extensible Markup Language application language labeled ModSimML, which serves as the data model. This XML specification confers the double advantage of improving visibility at the logical abstraction layer, by describing simulations in a human-readable and interpretable format, as well as providing an interface standard at the application



level. A Wiki website was created to enable users to collaborate in the definition of new modular systems posted in XML. We believe the use of an open format combined with a meaningful vocabulary increases the usability and the extensibility of the MDPSAS package.

## Chapter 7: Future Work

The project would greatly benefit from more development both on the server-side and on the client. A client application could be developed which would allow the users to assemble modular systems in a graphically friendly interface, and then upload the newly created components or systems to the web. Such an application could alternatively be developed as an applet that would run on the server.

We could use the Java Architecture for XML Binding (JAXB) tool to generate JAVA classes from the XML representations of modules and modular systems defined on the server. These classes could then be used to generate software specification diagrams. The Violet GUI is an example of an open source application which we could integrate into the MDPSAS development environment to provide the capability to build diagrams in UML. This enhancement would facilitate the process of systematically verifying the validity of the constructed models, an existing limitation of the current approach, particularly with respect to rules of connectivity between modules. The implementation of a better method of defining classes of parameters and variables would also facilitate model checking. Finally, we could develop a units schema that would ensure consistency in the unit system used within the configuration documents.

All these improvements would enhance the collaboration and data standardization goals this project seeks to resolve.

## Bibliography

- [1] Peters, M.S., Timmerhaus, K.D. & West, R.E. *Plant Design and Economics for Chemical Engineers*. McGraw-Hill, 2003
- [2] Chen, J. and Adomaitis, R. “An object-oriented framework for modular chemical process simulation with semiconductor processing applications”, *Computers & Chem. Eng.*, vol. 30, pp. 1354-1380, 2006
- [3] McCann Science, *Distributed Parameter Systems* [online]. Available: <http://www.mccannscience.com/distributed.htm>, 2003 [Accessed: October 2007]
- [4] Lu, G., & Kaxiras, *An overview of multiscale simulations of materials*. <http://arxiv.org/abs/cond-mat/0401073>. arXiv:condmat/0401073, v1, 2004 [Accessed: October 2007]
- [5] Maroudas, D., “Multiscale modeling of hard materials: Challenges and opportunities for chemical engineering”, *AIChE Journal*, vol. 46(5), pp. 878–882, 2000
- [6] Raimondeau, S., & Vlachos, D. G., “Recent developments on multiscale, hierarchical modeling of chemical reactors”. *Chemical Engineering Journal*, vol. 90(1–2), pp. 3–23, 2002
- [7] Hillestad, M., & Hertzberg, T., “Dynamic simulation of chemical engineering systems by the sequential modular approach”. *Computers and Chemical Engineering*, vol. 10(4), pp. 377–388, 1986
- [8] Fagley, J. C., & Carnahan, B., “The sequential-clustered method for dynamic chemical plant simulation”. *Computers and Chemical Engineering*, vol. 14, pp. 161–177, 1990
- [9] Lee, K. J., & Yoon, E. S., “The flexible modular approach in dynamic process simulation”. *Computers and Chemical Engineering*, vol. 18(Suppl.), pp. s761–s765, 1994
- [10] Adomaitis, R., *Numerical Methods for Chemical Engineers*. University of Maryland, 2007
- [11] Turton, R. et al, *Analysis, Synthesis, and Design of Chemical Processes, 2<sup>nd</sup> Ed.* Prentice-Hall, 2003
- [12] Adomaitis, R., “Objects for MWR”, *Computers and Chemical Engineering*, vol. 26, pp. 981–998, 2002

- [13] Eckel, B., *Thinking in Java, 4<sup>th</sup> Ed.* Prentice-Hall, 2006
- [14] Dennis, A. et al, *Systems Analysis and Design with UML Version 2.0, 2<sup>nd</sup> Ed.* Wiley, 2005
- [15] Mathews, J & Fink, K., *Numerical Methods Using MATLAB, 4<sup>th</sup> Ed.* Prentice-Hall, 2004
- [16] Alencar, P. and Lucena, C., “A Logical Theory of Interfaces and Objects”, *IEEE Transactions on Software Engineering*, vol. 28(6), 2002
- [17] Austin, M., *Lecture Notes for ENSE 623.* University of Maryland, 2007
- [18] Mu, M., “PDE.Mart: A Network-Based Problem-Solving Environment for PDEs”, *ACM Transactions on Mathematical Software*, Vol. 31(4), 2005.
- [19] Vilz, J. et al, “Data Conceptualisation for Web-Based Data-Centred Application Design”, *E. Dubois and K. Pohl (Eds.): CAiSE 2006*, LNCS 4001, pp. 205 – 219, 2006.
- [20] Zhang, P. et al, “Websites that Satisfy Users: A Theoretical Framework for Web User Interface Design and Evaluation”, *Proceedings of the 32nd Hawaii International Conference on System Sciences*, 1999
- [21] Wongthongtham, P. et al, “Towards 'Ontology'-based Software Engineering for Multi-site Software Development”, *3rd IEEE International Conference on Industrial Informatics*, pp. 362-365, 2005
- [22] Rajugan, R. et al, “Semantic Modeling of e-Solutions Using a View Formalism with Conceptual & Logical Extensions”, *3rd IEEE International Conference on Industrial Informatics*, pp. 286-293, 2005
- [23] Harold, E.R., *XML 1.1 Bible 3<sup>rd</sup> Ed.*, Wiley, 2004
- [24] Maluf, N., *An Introduction to Microelectromechanical Systems Engineering*, Artech House, 2000
- [25] Campitelli, A., and Parton, E., “BioMEMS: Marrying ICs and Biotech”, *Solid State Technology*, Vol. 45 Issue 7, p87, 2002