ABSTRACT

Title of dissertation: DEVELOPMENT OF AN OBJECT-ORIENTED FRAMEWORK FOR MODULAR CHEMICAL PROCESS SIMULATION WITH SEMICONDUCTOR MANUFACTURING APPLICATIONS

Jing Chen, Doctor of Philosophy, 2006

Dissertation directed by: Professor Raymond A. Adomaitis
Department of Chemical and
Biomolecular Engineering

Chemical Vapor Deposition (CVD) processes constitute an important unit operation for micro electronic device fabrication in the semiconductor industry. Simulators of the deposition process are powerful tools for understanding the transport and reaction conditions inside the deposition chamber and can be used to optimize and control the deposition process.

This thesis discusses the development of a set of object-oriented modular simulation tools for solving lumped and spatially distributed models generated from chemical process design and simulation problems. The application of object-oriented design and modular approach greatly reduces the software development cycle time associated with designing process systems and improves the overall efficiency of the simulation process. The framework facilitates an evolutionary approach to simulator development, starting with a simple process description and building model complexity and testing modeling hypothesis in a step-by-step manner. Modular-

ized components can be easily assembled to form a modeling system for a desired process. The framework also brings a fresh approach to many traditional scientific computing procedures to make a greater range of computational tools available for solving engineering problems.

Two examples of tungsten chemical vapor deposition simulation are presented to illustrate the capability of the tools developed to facilitate an evolutionary simulation approach. The first example demonstrates how the framework is applied for solving systems assembled from separate modules by simulating a tungsten CVD deposition process occurring in a single wafer LPCVD system both at steady-state and dynamically over a true processing cycle. The second example considers the development of a multi-segment simulator describing the gas concentration profiles in the newly designed Programmable CVD reactor system. The simulation model is validated by deposition experiments conducted in the three-segment prototype.

To facilitate the CVD system design, experimental data archiving, and distributed simulation, a three-tier Java and XML-based integrated information technology system has also been developed.

DEVELOPMENT OF AN OBJECT-ORIENTED FRAMEWORK FOR MODULAR CHEMICAL PROCESS SIMULATION WITH SEMICONDUCTOR MANUFACTURING APPLICATIONS

by

Jing Chen

Dissertation 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
Doctor of Philosophy
2006

Advisory Committee:

Professor Raymond A. Adomaitis, Chair/Advisor
Professor Mark A. Austin
Professor Kyu Yong Choi
Professor Panagiotis Dimitrakopoulos
Professor Evanghelos Zafiriou

DEDICATION

To my father, mother, and brother, who always love and support me

To my husband, who loves and encourages me all the time

To my daughter, who brings me joy

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor, Prof. Raymond A. Adomaitis, for his invaluable guidance, encouragement and support during my graduate study. His knowledge, patience, and vision have provided me with lifetime benefits.

I am very grateful to all my teachers. Courses taught by them have provided me with the background and foundations for my thesis research. Special thanks to Profs. Evanghelos Zafiriou, Mark Austin, Kyu Yong Choi, and Panagiotis Dimitrakopoulos of my thesis committee for their fascinating lectures, academic guidance, time, valuable comments and encouragement.

I want to thank Dr. Jae-Ouk Choo for the beneficial discussions and help from the very beginning of my research. In addition, I would like to thank Ramaswamy Sreenivasan and Dr. Yuhong Cai for sharing experimental data.

I am so grateful to my husband, Jian Ma, for his love and encouragement. Thanks Mom and Dad for everything. Without your love, nurturing and support, I could never accomplish all this.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Semiconductor device fabrication consists of multi-step chemical and physical processes to create the silicon-based integrated circuit chips. These steps include wafer preparation, device fabrication, device test, and packaging. Device fabrication is among the most complex manufacturing steps in producing semiconductor integrated circuits, and typically includes cleans, photolithography, ion implantation, etching, thermal treatments, chemical vapor deposition (CVD), physical vapor deposition, molecular beam epitaxy, electroplating, chemical-mechanical polishing, wafer testing and backgrinding [90]. These processes, some of which take place on the molecular or atom-scale level, are modeled as chemical reaction engineering problems, and the operations themselves constitute the unit operations of microelectronic device fabrication. Along with these similarities to the more traditional chemical process modeling issues are some differences, including an understanding of the basic chemical and physical processes at work that is less well-developed compared to the petro-chemical industries, and a more rapidly evolving process equipment and product design time scale.

Simulation tools are widely used in the semiconductor industry to supplement experiments and reduce the cost in developing new generations of products and in solving manufacturing problems [86]. However, it can be argued that simulation-

based design tools have failed to keep pace with equipment design evolution, because of the lack of truly flexible simulators that can model the exotic chemical mechanisms at work, as well as the large range of time and length scales characteristic of these manufacturing process systems.

1.1 Mathematical Modeling in Semiconductor Manufacturing

Physically based mathematical models for microelectronic processing provide a better understanding of process transport phenomena and reaction kinetics and can be used for advanced process control and optimization. The ultimate aim of process control is to improve equipment performance and overcome uniformity problems (temperature, film thickness, etc.). General automatic control approaches includes statistical process control (SPC), model-based control [102, 38, 8, 82], run-to-run control [16, 13], and real-time control [77, 26]. CVD processes, as one of the main unit operations of device fabrication, can be classified on the basis of different reactor categories: atmospheric pressure CVD (APCVD), low pressure CVD (LPCVD), cold-wall and hot-wall types, multiwafer or single-wafer, and different geometries such as vertical vs. horizontal, rotating disk, barrel, etc. The early efforts of reactor modeling work can be traced back to 1970s [39, 104, 107]. Kleijn [62] gave an overview of CVD reactor modeling studies from the modeling and simulation point of view, Badgwell *et al.* [7] reviewed the status of CVD process modeling in detail from the modeling and control point of view. Other excellent reviews of CVD process modeling can be found in the papers [47, 67, 49].

The modeling equations corresponding to the specific reactor types have evolved from the simple deposition reaction to more complicated models that consider surface reaction and transport phenomena in 1, 2 and 3 dimensions. Finding analytical solutions becomes infeasible with the increasing complexity of modeling equations. Typical numerical solution strategies to solve PDE (Partial Differential Equation) models include the finite volume method (FVM) [60, 61], finite element method (FEM) [57, 58, 80, 41, 53], finite difference method (FDM) [79, 55, 18, 106], orthogonal collocation method (OC) [56, 95, 6], and the use of model reduction techniques through proper orthogonal decomposition (POD) [105, 5, 10, 24, 59, 11, 3].

1.2 Multiscale Modeling

Technology computer-aided design (TCAD) models have been extensively used to develop and optimize semiconductor manufacturing processes. TCAD covers a wide region of semiconductor modeling areas which include front end process modeling, lithography modeling, device modeling, interconnect and integrated passive modeling, circuit element modeling, package simulation, materials modeling, and equipment/feature scale modeling [54]. With decreasing feature sizes in each generation of semiconductor devices, the understanding of underlying physical and chemical mechanisms in manufacturing processes grows in importance. Key phenomena often exist at different scales which are governed by different physical laws. Hence, the validity of conventional continuum-based or empirical models applied to these processes becomes questionable. The computational cost makes it impracti-

cal to treat all phenomena in a single finer scale level. To take into account their multiphysical nature, multiscale modeling methods now are regularly employed in semiconductor processes simulation.

Multiscale modeling has long been studied and extensively used in the various areas such as chemistry, materials, biology, and so on [36]. It provides a way for fundamental understanding and prediction of a process or material structure accurately. However, because multiscale modeling addresses a wide range of length and time scales inherent in a system, together with the lack of reliable fundamental data availability, it can be a challenge to find validated solutions to these types of simulation problems.

From the viewpoint of numerical methods, traditional techniques include multigrid methods, domain decomposition, fast multipole methods, adaptive mesh refinement, multiresolution methods, and the conjugate gradient method, which focus on solving microscale problems. The modern multiscale methods aim to reduce the computational complexity by using a scale separation approach. Representative methods of this technique are quasi-continuum method, Car-Parrinello method, superparametrization, heterogeneous multiscale method, coarse-grained Monte Carlo models, adaptive model refinement, and so on [34, 35]. From the implementation of the computation approach, the strategies can be classified as "parallel" and "serial" in terms of space or time scale. The representative methods of parallel length-scale approach include quasi-continuum method and the macro-scopic-atomistic-ab initio dynamics (MAAD) method. This approach implements different-scale techniques simultaneously in the same decomposed domain, whereas the serial approach imple-

ments different-scale techniques sequentially in different level discretization domains, and the information obtained from finer scales is the input of coarser scales. Kinetic Monte Carlo (KMC) and hyperdynamics methods can be categorized into serial and parallel approaches, respectively, in terms of time-scale [72]. These various multiscale methods are not suited for all kinds of applications, and different methods are limited for solving different problems. E and Enquist proposed a general framework called heterogenous multiscale method (HMM) for designing and analyzing multiscale methods, which can be applied to a wide variety of applications [36]. The main feature of HMM is linking models at different scales. The knowledge-based incomplete macroscale model is supplemented by the microscale model which provides missing numerical data information.

An overview of current status of multiscale simulations of materials can be found in Lu's paper [71]. Maroudas [72] highlighted the elements of a multiscale methodological approach and the ways to link the elements together. He also clearly stated the opportunities and challenges for chemical engineers on multiscale materials modeling in the semiconductor industry. A broader range of chemical reactors, relevant to issues in semiconductor processing is covered in Raimondeau and Vlachos' paper [91], where they also present a multiscale hierarchical computational framework for modeling homogeneous-heterogeneous reactors. Braatz *et al.* [17] give an excellent review on multiscale simulations in semiconductor process, multiscale applications, solution methods and issues of design and control of these systems. They discuss the challenges and requirements associated with the design and control of multiscale system tools. The information transfer between simulation

modules and parameter estimations is also addressed.

1.3 Software Approaches

Chemical process design, development and simulation, all of which encompass reactor design, equipment sizing and rating, steady-state or dynamic process simulation, chemically reacting and multi-phase flow simulation, or plant optimization and control, generate large sets of nonlinear algebraic and/or differential equation systems to be solved. Generally, three approaches can be taken to solve the resulting systems. One is for the user to develop the entire simulation program using a high-level programming language: this approach not only requires the user to understand the chemical and transport phenomena in the particular process very well, but also demands knowledge of the appropriate numerical methods and programming skills. The second approach is to use commercial or other specialized software designed for specific applications. Examples of these simulation packages for semiconductor processing simulations include the CHEMKIN [1] Collection for simulations of combustion, catalysis, corrosion, plasma etching, and CVD process; PHOENICS-CVD [2] for different types of CVD reactors and processes, such as rapid thermal processes, hot-wall batch reactors, and cold-wall single wafer reactors; and MPSalsa [3] for complex chemically reacting flow simulations, atmospheric chemistry modeling, surface catalytic reactors and 3D CVD simulation. Process simulators, such as Athena [4],

[1] CHEMKIN: a software developed by Sandia National Laboratories

[2] PHOENICS-CVD: a software product of CHAM Ltd.

[3] MPSalsa: product of Sandia National Laboratories

[4] Athena: product of SILVACO International

TSUPREM-4, Taurus-Process and FLOOPS-ISE [5], have been developed to simulate and optimize semiconductor manufacturing processes.

The third approach is to meet the user's needs in the middle between the first two approaches, i.e., using a relatively general commercial or other engineering software product to obtain simulation results where the full simulation is developed from a combination of built-in modules and user specified elements. These more general engineering simulation tools include flowsheet tools and PDE (Partial Differential Equations) solvers. Flowsheet tools are used for simulation, design, optimization and control of a complete plant or process with the simulator built-up from model modules describing unit operations. The PDE solvers are used for simulating distributed parameter systems focusing on the detailed transport phenomena within a contiguous spatial region, e.g. the dynamics of fluid flow through a specified enclosure. Representative commercial flowsheet tools are CHEMCAD [6], PRO/II [7], and Aspen Plus [8], while FLUENT, FIDAP [9], FLOW-3D [10], and COMSOL Multiphysics, formerly FEMLAB [11] are typical PDE-type applications. These software tools offer users a graphic interface to set up problems and run the simulations, and users generally must specify the physical domain, the transport and chemical processes, and boundary conditions without the need for detailed knowledge of fluid dynamics

[5]TSUPREM-4, Taurus-Process, FLOOPS-ISE: products of Synopsys, Inc.

[6]CHEMCAD, product of Chemstations, Inc.

[7]PRO/II: product of SimSci-Esscor

[8]Aspen Plus: product of Aspen Technology, Inc.

[9]FLUENT, FIDAP: products of Fluent Inc.

[10]FLOW-3D: product of Flow Science Inc.

[11]COMSOL Multiphysics, FEMLAB: products of COMSOL, Inc.

and computational techniques. While these are powerful tools, this approach can offer less flexibility in choosing the specific computational tools (e.g. solvers), fewer possibilities of customizing the problem physical domain, limitations on model reduction, control and other specialized applications, code reusability, and the cost of purchasing and updating commercial software.

1.4 Modular Approach for Flowsheet Tools

Steady state or dynamic process flowsheet simulators can be broadly classified into three categories: simultaneous (equation-based), sequential modular, and simultaneous modular. An overview of these approaches can be found in the paper of Hillestad and Hertzberg [48]. Equation-based simulators collect all modeling equations from each unit associated with a process and solve them simultaneously [83, 97, 101, 50, 51]. While the resulting simulators can be computationally efficient, problems can be encountered by users when model elements must be modified, both in terms of analyzing convergence problems as well as the difficulty with which these changes are made. Furthermore, computational efficiency can suffer in those cases where a unique relationship between model elements is lost when all equations are placed into a single module.

Modular-based strategies, widely used in many different applications [43, 85, 33, 66, 69], incorporate models of individual equipment elements or unit operations as a set of modules connected by material, energy, and information streams all communicating with a physical properties database. This approach allows the use of

different types of models (such as lumped or distributed models) for each chemical process and determines the solution to each module using the most appropriate algorithms [87]. The solution strategy used in sequential modular simulators is to determine a solution to each module separately and in a sequence determined by process material flow, incorporating an outer iteration loop if the process has recycle streams. While computationally straightforward to implement, this approach suffers from computing inefficiencies, and the need to frequently update local module states. To address these computational drawbacks, the simultaneous modular simulators use a hybrid approach, incorporating simultaneous and sequential simulation concepts by partitioning a process into several modules or module-clusters consisting of coupled units [40, 68]. The module-clusters that have strong interdependence are solved simultaneously, while the process as a whole is solved sequentially from cluster to cluster.

1.5 Object-Oriented Approach for PDE Solvers

The primary challenge to the development of numerical PDE solvers is how sets of the partial differential equations generated from distributed systems can be solved rapidly, robustly, and reliably. The application of object-oriented programming (OOP) techniques to fluid dynamics software design in conjunction with the development of parallelized computational techniques have greatly improved the computing efficacy, the flexibility of these computing environments, and the degree of code reusability and extensibility. Cai *et al.* [19, 20] demonstrated the advantage

of using multiple processor computing platforms with OOP techniques and developed a parallel PDE solver based on the existing serial libraries of Diffpack [64], an object-oriented PDE solver for scientific computing applications. Ramirez *et al.* [92, 93] proposed an object-oriented framework using a design patterns methodology to solve CFD problems efficiently on high performance parallel computers. Peskin *et al.* [84] designed a software package to solve transport phenomena problems on moving boundary domains using the Galerkin finite element method. Langtangen *et al.* [65] built a simulator based on Diffpack to solve nonlinear, coupled PDEs using independent solvers for different equations in the system, i.e., different solvers for implicit or explicit equations and compound systems.

1.6 Our Approach

Because the primary goal of our simulator development is to provide a flexible and extensible structure for solving a wide range of chemical process simulation problems, the proposed framework makes extensive use of object-oriented programming techniques. Our approach is to break simulation problems into modular components, where a module typically consists of a subelement of a single manufacturing process, such as a wafer heater, reaction chamber, or reaction network; 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; this makes it possible to define modular systems com-

bining lumped and distributed parameter models. The defined modules become a reusable part of modeling library.

The properties of modularity are related to object-oriented features of programming languages. Object-oriented programming supports encapsulation, inheritance and polymorphism. Objects are categorized into classes and class hierarchies, and each class contains attributes describing objects of that class and operations defining their behavior. Encapsulation packs data and the operations that manipulate the data into a single named object. Inheritance enables the attributes and operations of a class to be inherited by all subclasses and the objects that are instantiated from the subclasses. Polymorphism allows the use of a common name for a method that acts differently among objects of different classes [88].

1.7 Thesis Organization and Contributions

In this thesis, we first discuss the design and implementation of the simulation framework, then demonstrate the framework functionalities through the applications on semiconductor CVD process. The thesis concludes with final remarks and recommendations for future work. Details are as follows:

Chapter 2: Introduce the framework architecture and discuss the detailed structure of each individual package through a simplified model of heat transfer in a single wafer CVD system. The software and the examples discussed in Chapter 2 and Chapter 3 can be found at the website:

http://www.isr.umd.edu/Labs/CACSE/A_team/software/mcps.

Chapter 3: Model and simulate a tungsten deposition process in a cold-wall single wafer CVD reactor system both at steady state and dynamically over a true processing cycle to demonstrate the functionalities of the modular simulation framework.

Chapter 4: Discuss the modeling and simulation of gas concentration profiles in the newly designed programmable CVD reactor system. The model has been validated by the experiments conducted on the three-segment prototype.

Chapter 5: Discuss the development of a Java and XML based three-tier information system to facilitate CVD system data archiving, analysis and management.

Chapter 6: Conclude the thesis with final remarks and future work.

Chapter 2

Framework of Object-oriented Simulation Tools

2.1 Overview of Framework Architecture

Starting with a general problem solving environment (MATLAB [1]), we develop a simulation framework incorporating elements of flowsheet and PDE solvers, that facilitate a flexible, modular, model building and analysis approach, allowing an evolutionary approach to simulator development. We develop this modular approach in the context of semiconductor manufacturing process simulation, where the granularity of modularization is less clearly defined than in the unit operations of traditional chemical processes. A central feature of our framework is the incorporation of object-oriented techniques for implementing weighted residual methods, which allows the solution of PDE based models.

As shown in Figure 2.1, the main packages in our simulation framework include a physical property database, modular components, system and relation classes and solver tools. The property database package works like a library for physical properties of common semiconductor processing gases. The modular components package includes standalone modules that encapsulate subsystem design information and modeling equation definitions. The function of the system and relation packages is to facilitate integration of independent modules to form user-defined systems.

[1] MATLAB: product of The MathWorks, Inc.

Systems of this type are solved using the methods offered in the solver packages.

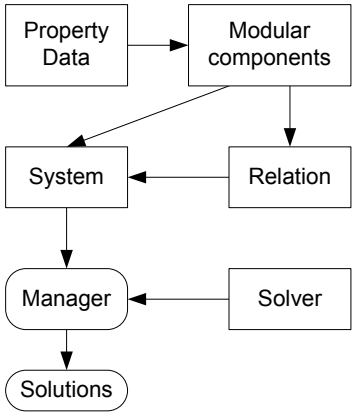

Figure 2.1: The main packages in the simulation framework

We present our software design concepts in the context of a design patterns methodology. A design pattern is a good solution to a general or commonly recurring software design problem. The use of design patterns has multiple benefits; saving the designer time and effort by making the communication between designer and user or other programmers easier and clearer, making the design more flexible, extensible and reusable, and by avoiding details in the early stage of design giving the designer a higher-level view of the problem and on the process of design [42, 30, 99].

Although design patterns provide a general approach to object-oriented software design, the particular implementation can differ based on the choice of programming language; some patterns are not needed, while other patterns are more easily expressed in one language than another. Our simulation framework is implemented using a programming environment for scientific computing (MATLAB) that has a number of object-oriented features. We use the mediator pattern for

communication among objects of module classes. The facade pattern is used to provide a simple interface to a subsystem of weighted residual method tools. We also have used the strategy and adaptor patterns in the solver package to provide different algorithms and integrate other numerical packages. The patterns used in our system design will be explained in the following sections.

2.2 Simulation Framework Implementation

We focus on solving models of the form:

$$\frac{\partial x}{\partial t} = M(x, p)$$

$$0 = N(x, p)$$

where M, N denote nonlinear algebraic or differential operators, x is a data structure containing model variables, and p is a data structure defining model parameters. Our goal is to develop a framework where these equations can be easily incorporated into a modeling module and subsequently solved, both for dynamic and steady state solutions. It will be described later in this paper how these modules can be interconnected to form larger modeling equation systems. The class diagram of proposed framework is depicted in Figure 2.2.

2.2.1 Modular Components

Compared to the classic modeling of large and complex systems, modular approaches break down a complete chemical processing system into smaller building blocks, reducing the complexity of model building, simplifying maintenance, making

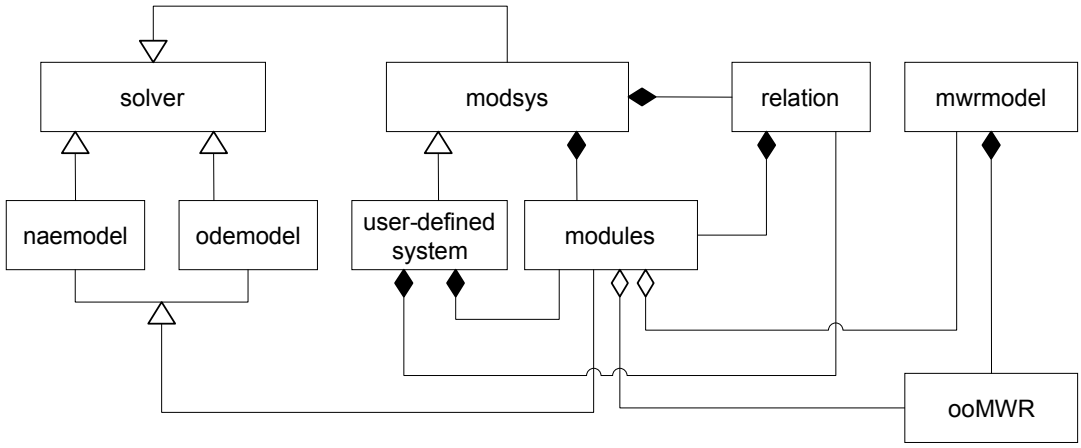

Figure 2.2: The class diagram of the simulator framework

it safer and easier to modify a model component, and in some instances facilitating distributed or parallel simulation [31].

How to decompose a large system into smaller modules and later reassemble them without loss of essential information about the original system is the primary challenge in developing simulation modules. Meyer [78] proposed the "open-closed principle" for software design where the modules, classes, methods, etc. should be open for extension, but closed for modification to avoid introducing errors. Cota and Sargent [31] suggest that modularity should have the properties of locality and encapsulation: locality means that all design decision information related to the system being modeled should be stored in the same place, and encapsulation keeps the state variables and behavior of one component isolated from other components. Using these properties, the modeler can make internal changes in one component without considering other parts of the model as long as the exposed interface remains

the same. Zeigler [109, 110] argues that a module should be like "black box", i.e. its internal state and behavior must make no references to analogous information of any other module, and the modules must communicate with others only through their input and output.

Based on the properties of modularity, a standalone module class shown in Figure 2.3 is formulated in terms of the following methods: a constructor method, a *residual* or *evalvar* method, and plotting or display methods. All process design, equipment geometry, and related information are stored in the constructor method, where the data fields are categorized into two types: *var* representing variables and *param* representing parameters. The data type of *var* and *param* is defined as an object of the class *assocaarray*. The class *assocarray* acts as a hash table, storing both variable names and values. Helper methods such as *delete*, *sort*, *insert*, and more were written to facilitate manipulation of *assocarray* objects. In the module class, if the variables to be found can be expressed by the modeling equations in terms of parameters explicitly, i.e., in the form of $x = g(p)$, then these equations are placed in the *evalvar* method; if only implicit expressions define the model residual, i.e., $f(x, p) = 0$, they should be used to define the *residual* method. The number of variables must be same as or less than that of the modeling equations. Modules can be solved individually to test the convergence behavior of each.

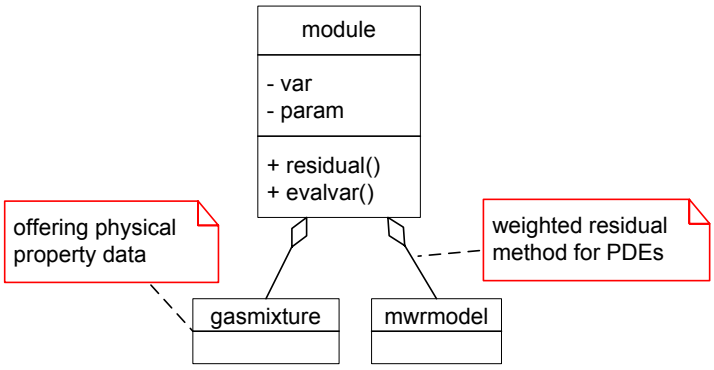

Figure 2.3: The class diagram of a standalone model module

2.2.2 The mwrmodel Class

If the modeling equations to be solved consist of partial differential equations (PDEs), a spectral projection or other discretization method implemented in ooMWR tools can be used to solve these systems. ooMWR tools was developed for solving boundary value problems (BVPs) in relatively simple geometries using global trial function expansions and weighted residual methods (MWR) [1, 2, 70]. Objects of *mwrmodel* class are created to discretize the PDEs into algebraic equations (AEs), or semidiscretize equations into ordinary differential equations (ODEs), or algebraic-differential equations (DAEs). The resulting systems then can be solved using the tools developed in *solver* classes which will be introduced in the following sections.

To integrate the ooMWR tools into the current framework, the facade pattern is applied to develop the *mwrmodel* class, which provides an interface to the ooMWR class subsystem, as depicted in Figure 2.4. The facade pattern wraps a

complex subsystem together and provides the user a simplified interface to access the functionality of the wrapped subsystem.

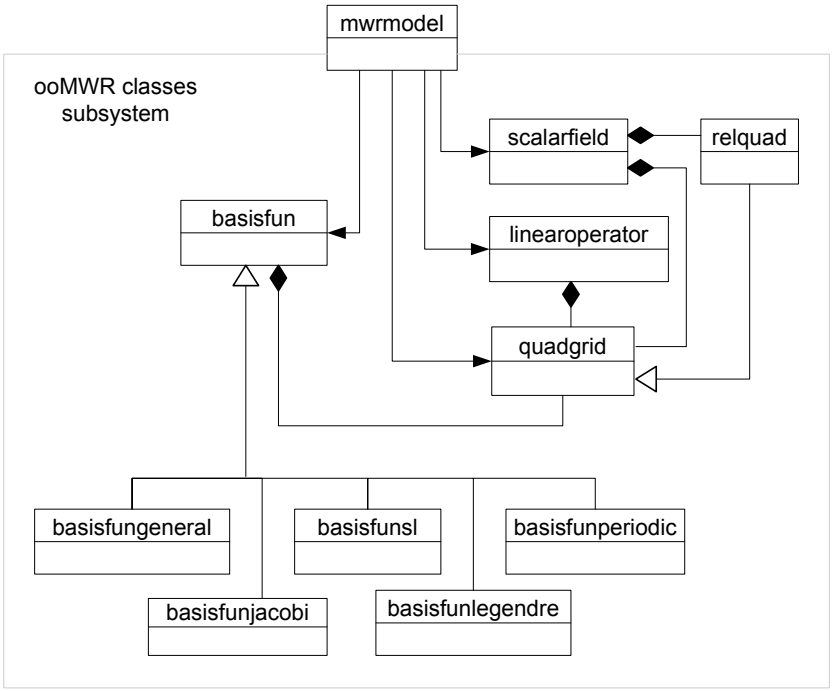

Figure 2.4: The *mwrmodel* interface for wrapping the ooMWR class subsystem

By creating an object of *mwrmodel* class, the user automatically obtains all ooMWR objects needed to discretize PDE systems without the need for any knowledge regarding its underlying classes, reducing the number of objects that the user must create, simplifying the discretization process. Also, the *mwrmodel* class decouples the subsystem of ooMWR from the user or other subsystems, promoting subsystem independence and portability. Any changes to the ooMWR classes will not affect the user or other subsystems provided the *mwrmodel* interface does not change, minimizing future module modification.

In most applications, the *mwrmodel* class interface provides all the functionality needed to implement numerical MWR. However, this pattern does not prevent users from accessing the underlying classes and methods directly, indeed, a sophisticated user preferring more controllability and flexibility may choose to instantiate objects from ooMWR classes directly.

2.2.3 Physical Property Data

The current database package provides methods for determining properties of non-polar and polar gases and organic and inorganic gases with emphasis on supporting common semiconductor process gases such as silane, tungsten hexafluoride and trimethylgallium. The thermo-physical properties include viscosity, thermal conductivity, binary diffusivity, heat capacity, molecular weight and density of pure gases and mixtures at an ideal gas state. Properties of 62 pure gases and their mixtures may be obtained through the interface. This package is implemented in JAVA for both web applications and local simulations. An interface wrapper class, *gasmixture*, was developed in MATLAB to facilitate data retrieval from the JAVA physical property objects.

2.3 The modsys and relation Classes

With a computational methodology for creating standalone model modules in hand, several issues must now be addressed. For example, how does one assemble several modules together to construct more complex modules or a complete chemical

process simulator? Modules developed by different research groups or developed for different applications may use different names for the same variables or same names for different variables, and so how do we relate each to the others when solving the integrated system? Finally, how is information exchanged among objects during computations involving assemblies of modules?

Direct communication among objects would result in a tight coupling of modules and loss of modularity. To link the modules together and still preserve the loose coupling and flexibility, we create a class called *relation*. The variables or parameters with same or different names in different modules are related through the information users provide when instantiating the objects of the *relation* class.

Although we have methods for updating data fields of related objects in the *relation* class, they are not intended to be called explicitly. Communication among objects is administrated through the methods of the *modsys* class, which is developed using the mediator pattern. The mediator pattern defines the only object that knows the state of all others and coordinates information exchange among the other objects in the system. These classes communicate with or through the mediator without referring each other explicitly. Building simulators in this modular fashion is easy to understand, maintain and extend. The *modsys* approach replaces many-to-many relationships with one-to-many relations between the module systems and module objects. It also decouples the module objects with plug and play functionality.

As depicted in Figure 2.5, the user may create a modular modeling system using one of two approaches. The first is to create an object of the *modsys* class directly by inputting the objects of individual module classes and the objects of the

relation class describing the relationship between variables and parameters among those module objects. Alternatively, a user can wrap all information relating to module objects and their relationships into a user-defined system, which is defined as a subclass of *modsys* class. In this way, users need only to instantiate the user-defined system in the main program, which makes the simulation programs simple to follow and hides unnecessary information from the clients.

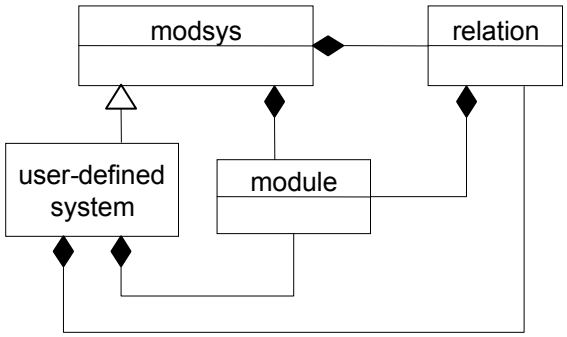

Figure 2.5: The class diagram of *modsys* and *relation* classes and their relationships with user-defined system and standalone modules

As mentioned earlier, each module class has a method called *residual* to store the modeling equations. Like the module class, the *modsys* class has a *residual* method as well, but its functionality is different: instead of combining all modeling equations from the individual module classes, the *residual* method in the *modsys* class acts as a coordinator for distributing, exchanging and receiving information among different objects.

During computation, solvers only access and update the system's variables. In the *residual* method, these updated variable values first are distributed to each cor-

responding module object, and then the *residual* method for each object is invoked to update the data field *resid* and the values of pseudo-constant parameters that are function of variables defined in that model. According to the *relation* objects, information in related module objects is updated by calling the method *updatefields* in the *relation* class. Finally, the computational results are collected and saved in the data field, *resid*, in the modsys object for the next calculation.

To demonstrate how this process works, we use a highly simplified model of heat transfer in a single wafer CVD system. The wafer is heated by a lamp bank located outside the reactor chamber (Figure 2.6). T_w is wafer temperature and T_q is quartz window temperature. Lamp radiation reaches wafer through the quartz reactor window; the window itself absorbs a fraction of the lamp radiation. The equations are coupled through the nonlinear radiative heat transfer terms between the wafer and window. The parameters used for simulation is listed in the Table 2.1 and the steady state modeling equations are given below.

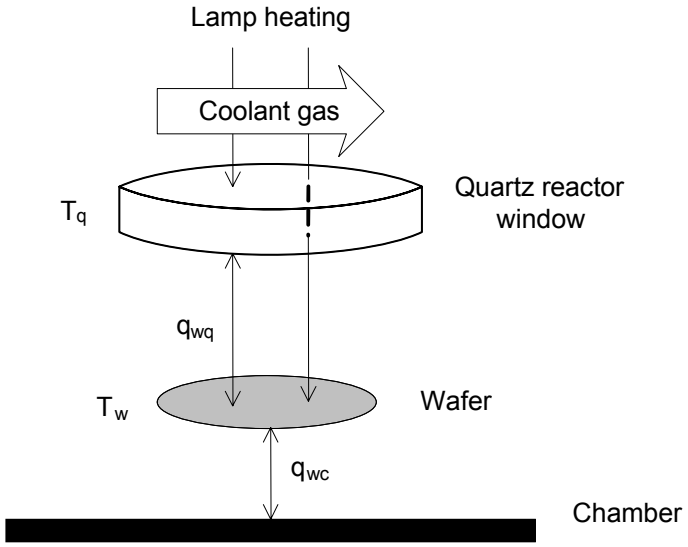

Figure 2.6: The diagram of a wafer heated by a lamp through a quartz window

$$
\begin{aligned}
f_1(T_w, T_q) \quad & \textit{wafer energy balance} \\
= \quad & \sigma\epsilon_w\epsilon_q T_a^4(T_q^4 - T_w^4) \quad \textit{wafer/window radiation } (q_{wq}) \\
& +\sigma\epsilon_q\epsilon_s T_a^4(1 - T_w^4) \quad \textit{wafer bottom/chamber heat transfer } (q_{wc}) \\
& +(1-a)Q \quad \textit{lamp heating of wafer}
\end{aligned}
$$

$$
\begin{aligned}
f_2(T_w, T_q) \quad & \textit{quartz window energy balance} \\
= \quad & \sigma\epsilon_w\epsilon_q T_a^4(T_w^4 - T_q^4) \quad \textit{window/wafer radiation} \\
& +hT_a(1 - T_q) \quad \textit{window coolant gas} \\
& +aQ \quad \textit{lamp heating of window}
\end{aligned}
$$

Table 2.1: Physical constants and RTP furnace design parameters.

Parameter	*Value*	*Description*
ϵ_w	0.7	Si emissivity
ϵ_q	0.5	quartz emissivity (high temperature)
ϵ_s	0.07	steel emissivity (room temperature)
a	0.01	quartz absorptivity
σ	5.670×10^{-8}J/(K^4 m^2 s)	Stefan-Boltzmann constant
T_a	300 K	ambient temperature
h	300 W/m^2	quartz window/cooling gas heat transfer coeff
Q	5000 W/m^2	lamp radiation flux

The iterative approach to solving for the coupled set of nonlinear algebraic equations can be written as:

$$\begin{pmatrix} T_w \\ T_q \end{pmatrix}^{(n+1)} = \begin{pmatrix} T_w \\ T_q \end{pmatrix}^{(n)} + \begin{pmatrix} \delta_w \\ \delta_q \end{pmatrix}^{(n)}$$

and the update vector δ can be obtained by:

$$\delta = -J^{-1} f$$

where J is the Jacobian matrix of the linearized system, and f is the vector of functions $f_1(T_w, T_q)$ and $f_2(T_w, T_q)$. The Jacobian matrix is defined as follows:

$$J = \begin{pmatrix} \frac{\partial f_1}{\partial T_w} & \frac{\partial f_1}{\partial T_q} \\ \\ \frac{\partial f_2}{\partial T_w} & \frac{\partial f_2}{\partial T_q} \end{pmatrix}$$

Now we solve this system using our modular approach. First, two individual modules called *wafertherm* and *windowtherm* are defined as:

wafertherm class:

var : T_w

param : σ, ϵ_w, ϵ_q, ϵ_s, T_a, T_q, a, Q

residual function : $f_1(T_w, T_q)$

windowtherm class:

var : T_q

param : σ, ϵ_w, ϵ_q, T_a, T_w, h, a, Q

residual function : $f_2(T_w, T_q)$

where the data field *var* includes the information on all variable names and initial solution estimates (or initial conditions for a time dependent-problem), and the date field *param* wraps all parameter names and values into an *assocarray* object. The *residual* method stores all linear or nonlinear algebraic equations whose residuals are to be driven to zero to define the variable values which constitute a steady-state solution. The number of equations must be equal to or greater than the number of variables. The complete class definitions in MATLAB is attached in the Appendix.

Each module can be solved independently: an object of each class is defined and the Newton's method is applied to each. For example, T_q is defined as a constant $600K$in the *wafertherm* module, the MATLAB script for solving the *wafertherm* module is as follows:

```
A = wafertherm;     % create instance of wafertherm class
```

```
A = newton(A);      % solve function f1
unpack(A)           % extract converged solution
Wafertemp = Tw*Ta
```

The convergence of the Newton procedure and resulting steady state wafer temperature are shown below:

```
Update/residual norm = 0.58589 / 368.2513
Update/residual norm = 0.031479 / 9.7532
Update/residual norm = 0.00086492 / 0.44817
Update/residual norm = 3.9704e-005 / 0.020345
Update/residual norm = 1.8024e-006 / 0.00092408
Update/residual norm = 8.1869e-008 / 4.1972e-005
Update/residual norm = 3.7185e-009 / 1.9063e-006
Update/residual norm = 1.6889e-010 / 8.6587e-008
Update/residual norm = 7.6711e-012 / 3.9345e-009
Update/residual norm = 3.4857e-013 / 1.7917e-010

Wafertemp = 766.5719
```

Note that because finite differences are used to compute Jacobian array elements, the convergence is less than quadratic; additional computational details follow.

Solving for *windowtherm* module individually, T_w is defined as a constant $600K$. The computational procedure is as follow:

```
B = windowtherm;

B = newton(B);

unpack(B)

Windowtemp = Tq*Ta
```

The updates during solution procedure and the resulting steady state quartz window temperature are:

```
Update/residual norm = 0.98835 / 1405.161

Update/residual norm = 0.016202 / 64.0842

Update/residual norm = 0.00073866 / 2.9101

Update/residual norm = 3.3544e-005 / 0.13218

Update/residual norm = 1.5236e-006 / 0.0060036

Update/residual norm = 6.92e-008 / 0.00027268

Update/residual norm = 3.1431e-009 / 1.2385e-005

Update/residual norm = 1.4276e-010 / 5.6253e-007

Update/residual norm = 6.4841e-012 / 2.5558e-008

Update/residual norm = 2.946e-013 / 1.1655e-009


Windowtemp = 308.1433
```

To solve for the two unknowns (T_w and T_q) simultaneously, we must define an integrated system. The relationship of the two independent modules are described as objects of the *relation* class, i.e., the parameter T_q in *wafertherm* class is equivalent

to the variable T_q in *windowtherm* class, and the variable T_w in *wafertherm* class is the parameter T_w in *windowtherm* class.

The procedure for assembling the integrated system begins by instantiating an object of *modsys* class using its constructor method, with the list of modeling objects and relationships as input parameters. A table of variables is created by mapping system variables to their corresponding module objects. Instead of pulling all equations from the individual modules' *residual* method and putting them into that of *modsys* class, the *residual* method of this class is used to manage each module class data updates and retrieve computed residuals from these modules, returning them as the residuals of system equations.

To illustrate how this works, consider computing of the first column of the Jacobian matrix by perturbing the variable T_w with ε. Then we calculate $f_1(T_w, T_q)$ and $f_1(T_w + \varepsilon, T_q)$ from the *residual* method of the *wafertherm* class, and $f_2(T_w, T_q)$ and $f_2(T_w + \varepsilon, T_q)$ from the *residual* method of the *windowtherm* class. To evaluate $f_1(T_w, T_q)$ and $f_2(T_w, T_q)$, the parameter T_q in the *wafertherm* class object must be updated by the value of variable T_q in the *windowtherm* class object, similarly, the parameter T_w in the *windowtherm* class must be updated by the value of variable T_w in the *wafertherm* class. Hence, when the *residual* method of *modsys* class is called, it updates each module object's variables with the latest values of the system variables through the mapping table; then it calls the *updateall* method of the *relation* class to exchange data information between the two module objects according to their relationship stated in the objects of the *relation* class; lastly it calls the *residual* method of each individual module object to retrieve the value of function residuals

and return them as $f_1(T_w, T_q)$ and $f_2(T_w, T_q)$.

Computation of $f_1(T_w + \varepsilon, T_q)$ and $f_2(T_w + \varepsilon, T_q)$ is, in principle, the same as that of $f_1(T_w, T_q)$ and $f_2(T_w, T_q)$. However, because only one system variable is perturbed, it is not necessary to update all objects' variables and to perform the relationship check of which parameters must be updated. Therefore, each time the *residual* method of the *modsys* class is called, it will check and compare the latest system variables with the previous ones to determine which has changed, and then only update the corresponding variables of the appropriate module object. The relationship of other objects associated with this object is determined and stored for further usage. Then only those parameters connected with this variable are modified according to the saved new relationships. In this example, T_w is perturbed and T_q remains unchanged, so only the variable in the object of *wafertherm* class and the parameter T_w in the object of *windowtherm* class are updated with $T_w + \varepsilon$, respectively. Finally the individual object's *residual* method is called to collect the function residuals, and they are returned in the *residual* method of *modsys* class as the values of $f_1(T_w + \varepsilon, T_q)$ and $f_2(T_w + \varepsilon, T_q)$.

Finally, the system solutions obtained by calling the *newton* method defined in the *naemodel* class are retrieved through the *getobj* method in *modsys* class. The following is the actual code in MATLAB for the implementation of this example.

```
% define relationship between A and B
R = relation( A,B,{ 'Tw' 'Tw' 'Tq' 'Tq' } );
S = modsys(R,A,B);     % create modular system
```

```
S = newton(S);          % use Newton method to solve

D = getobj(S);          % extract converged module objects

unpack(D); unpack(A,'var'); unpack(B,'var');

Wafertemp = Tw*Ta

Windowtemp = Tq*Ta
```

The steady state solutions of wafer and quartz window temperatures are obtained as follows:

```
Update/residual norm = 0.21692 / 138.182

Update/residual norm = 0.015952 / 5.2968

Update/residual norm = 0.00056957 / 0.24148

Update/residual norm = 2.6063e-005 / 0.010966

Update/residual norm = 1.1834e-006 / 0.00049808

Update/residual norm = 5.375e-008 / 2.2623e-005

Update/residual norm = 2.4413e-009 / 1.0275e-006

Update/residual norm = 1.1088e-010 / 4.6668e-008

Update/residual norm = 5.0366e-012 / 2.1164e-009

Update/residual norm = 2.286e-013 / 9.7218e-011


Wafertemp = 697.3332

Windowtemp = 315.1561
```

The sequence diagram in Figure 2.7 illustrates the interaction of these objects and each function call procedure. The *modsys* class is mainly used for solving coupled

modules simultaneously, while independent modules can be solved sequentially. The current computational framework supports both approaches.

2.4 The Solver Package

Generally, the modeling systems we wish to solve consists of algebraic equations (AEs), ordinary differential equations (ODEs) or partial differential equations (PDEs), and differential-algebraic equations (DAEs). As discussed in the previous section, PDE systems normally are (semi)discretized using a global spectral projection method. Alternatively, we can use the *quadgrid*, *linearoperator* and *scalarfield* classes in combination to discretize a PDE system into a semi-discretized ODE system by collocation; this approach normally results in the boundary conditions becoming algebraic equations. In our framework, ODE model systems assembled with objects having *residual* and *evalvar* methods become DAE systems automatically because the *evalvar* method stores algebraic equations defining variables explicitly.

Therefore, in the *solver* package, we must provide computational tools for solving AE, ODE or DAE systems. The class diagram of this package is illustrated in Figure 2.8. In this package, we have a base class, *solver*, which defines data fields such as *var*, *param*, *resid*, *solverset*, *currtime*, and more for subclass usage, and offers utility methods such as *unpack*, *display*, *get*, *set*, *columnnae*, *residual* and *evalvar*, to facilitate derived class operations. The *residual* method and *evalvar* method are both abstract methods and, as we have seen, must be overloaded in subclasses of the *solver* class. If only the *evalvar* method is overloaded, the abstract *residual* method

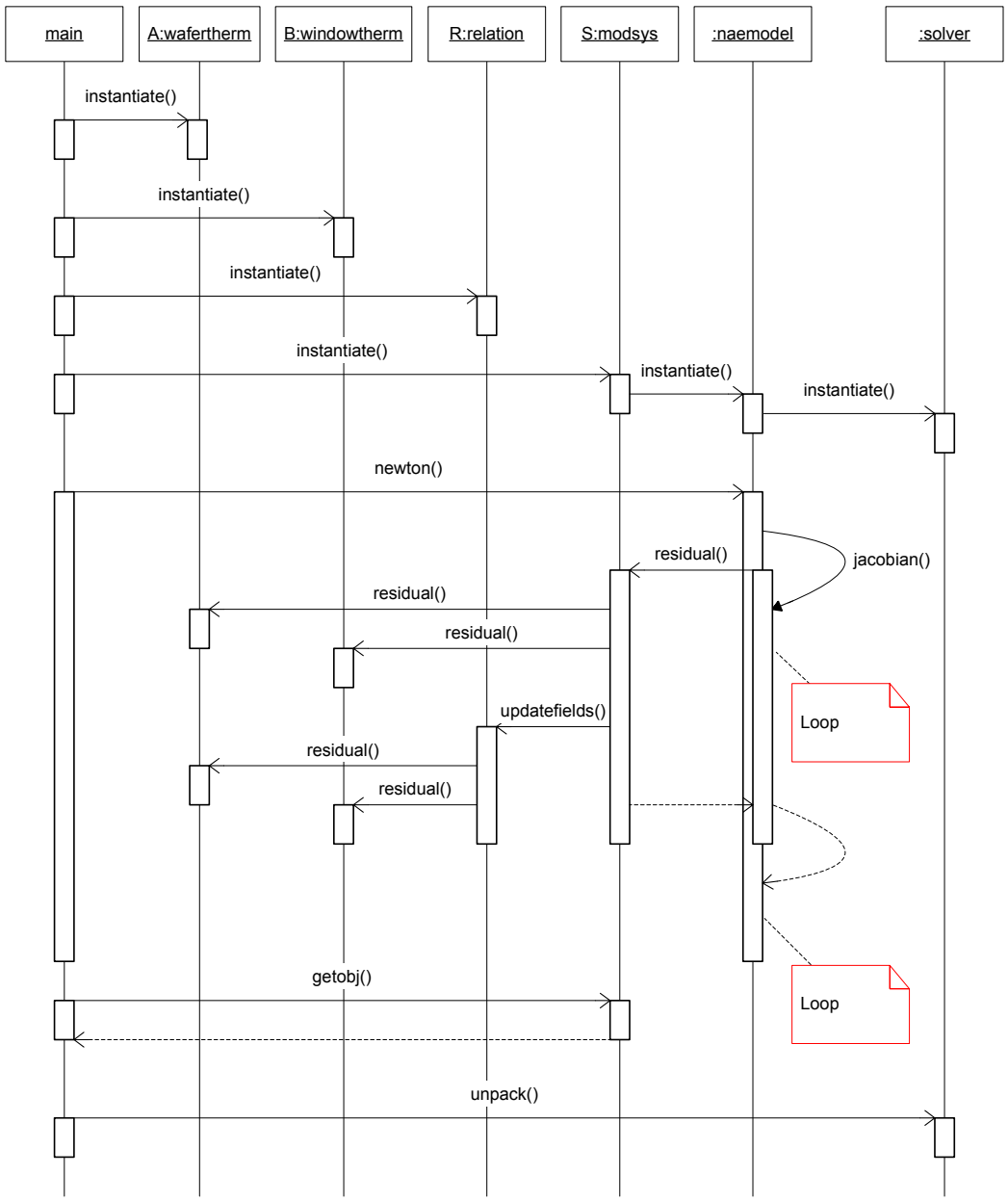

Figure 2.7: The sequence diagram for solving an assembled system with two modular objects

is used to handle several tasks of the *evalvar* method, such as setting the *dxdtcoeff* field to indicate equations in the *evalvar* method that are algebraic equations, and updating the *resid* field for the *evalvar* method where only *var* field is updated during each computation procedure.

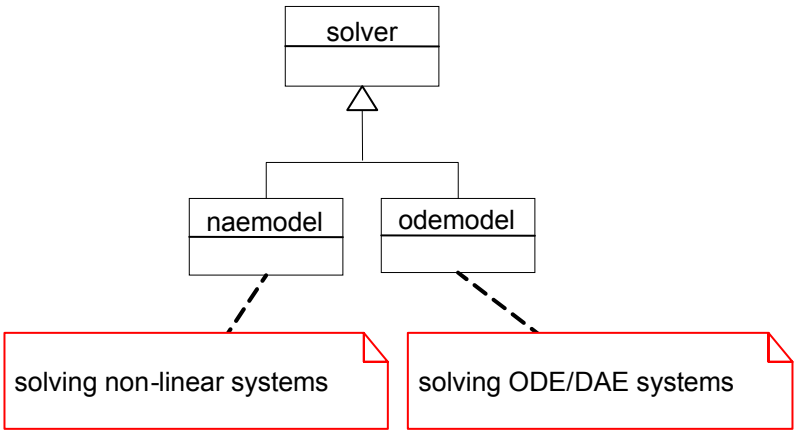

Figure 2.8: The class diagram of the *solver* package

Solving linear/nonlinear systems is a common operation in scientific computing. In addition to the Newton-Raphson procedure described, generic methods used for finding solutions to nonlinear systems include the Quasi-Newton, bisection, and successive substitution methods, and for linear systems, iterative methods such as Krylov subspace method. To group a number of different algorithms together, the class *naemodel* was developed using the strategy pattern. The strategy pattern supports a family of algorithms, which conceptually do the same thing but have different implementations. Different numerical algorithms are implemented as member methods of the *naemodel* class, and the user can invoke different numerical algorithms

by calling these methods directly. Individual module classes or the *modsys* class are defined as a subclass of this *solver* class.

MATLAB supports multi-inheritance. To be able to use the different methods of the *odemodel* class for solving ODE/DAE systems, the module class or *modsys* class also must be a subclass of the *odemodel* class. But the design of the *odemodel* class is totally different from that of *naemodel* class. There are many numerical computation packages developed in MATLAB for solving ODEs and DAEs with different algorithms. To make use of these existing solvers, we designed an interface, class *odemodel*, using the adapter pattern, to allow integration of MATLAB built-in ODE/DAE solver packages into our simulation framework. The adapter pattern, also known as a wrapper, converts the interface of a class into another interface to be compatible with other classes. In MATLAB, a representative interface of the ODE and DAE solvers is:

```
[t,y] = ode45(odefun, tspan,y0,options, p1,p2 );
```

and the returned time t and solution y has the form of column vector and array, respectively. The function reference and variable data type are not compatible with our design because in the module class, the data type of *var* and *param* is *assocarray*. Variables wrapped in *var* may have other format data types, such as double, array or *scalarfield* which is defined to wrap the quadrature grid and state variable values for discretized states. Recall also modeling equations are defined in the module class *residual* method with the form of $A = residual(A)$ instead of the user supplied functions with the form of $dydt = odefun(t, y)$.

Therefore, a method called *odesolver* in the *odemodel* class was developed to perform multiple functions: (1) to be responsible for data type conversion, i.e., change the user defined data type to one compatible with that recognized by the MATLAB ode solvers and recover the results in the user-defined type; (2) redirect the function reference: instead of directly referring the *residual* or *evalvar* method of the module class, the function handler (*odefun*) in the standard ode solver is referred to a function defined within the *odesolver* method, where the module or *modsys* object is passed as a parameter and the *residual* method of the object is called to compute and return the time-derivative values; and (3) to generate a mass matrix for DAE systems automatically. By evaluating the data field *dxdtcoeff*, which is set in the *residual* method of module class, it will be determined whether this is an ODE system or a DAE system. For a DAE system, the size of the dependent variables is evaluated and a sparse mass matrix is generated. The default DAE solver is called to solve the system if users do not supply one. The *odesolver* method has the interface:

```
Bv = odesolver(Aobj, tspan, options, solver);
```

where *Aobj* is the object of *odemodel* class, *tspan* and *options* are same as that defined in standard MATLAB ODE solvers, *solver* is a string to allow the user to choose different solvers in MATLAB, for example, 'ode45', 'ode23', and so on. *It is important to note that the output Bv is a vector of odemodel objects corresponding to the state variable at each point in time.*

Chapter 3

Modeling and Simulation of Tungsten CVD Reactor System

To demonstrate the use of the modular simulation framework, we now consider the problem of simulating a tungsten CVD deposition process both at steady state and dynamically over a true processing cycle. While single wafer CVD systems are true dynamic processes, steady state simulations are useful for determining the maximum temperature a wafer can reach during a processing cycle. A schematic diagram of the tungsten CVD reactor system under consideration is shown in Figure 3.1. We will consider the H_2 reduction of WF_6 as the deposition process for this example.

Hydrogen enters the chamber through the transparent showerhead mounted in the top of the chamber, and tungsten hexafluoride is injected through a slit on one side of the chamber wall. A tungsten-halogen lamp ring is located outside the reactor chamber and is used to heat the wafer through the transparent quartz showerhead window. The precursor gases mix in the chamber and react at the wafer surface, forming thin film layer of tungsten on the wafer. The 4-inch diameter wafer is supported by a slowly rotating quartz susceptor and its outer edge is covered by a quartz guard ring. More details on the model derivation, heat transfer parameter selection and reactor geometry can be found in the paper [3].

In the following sections we will discuss modeling gas composition change dur-

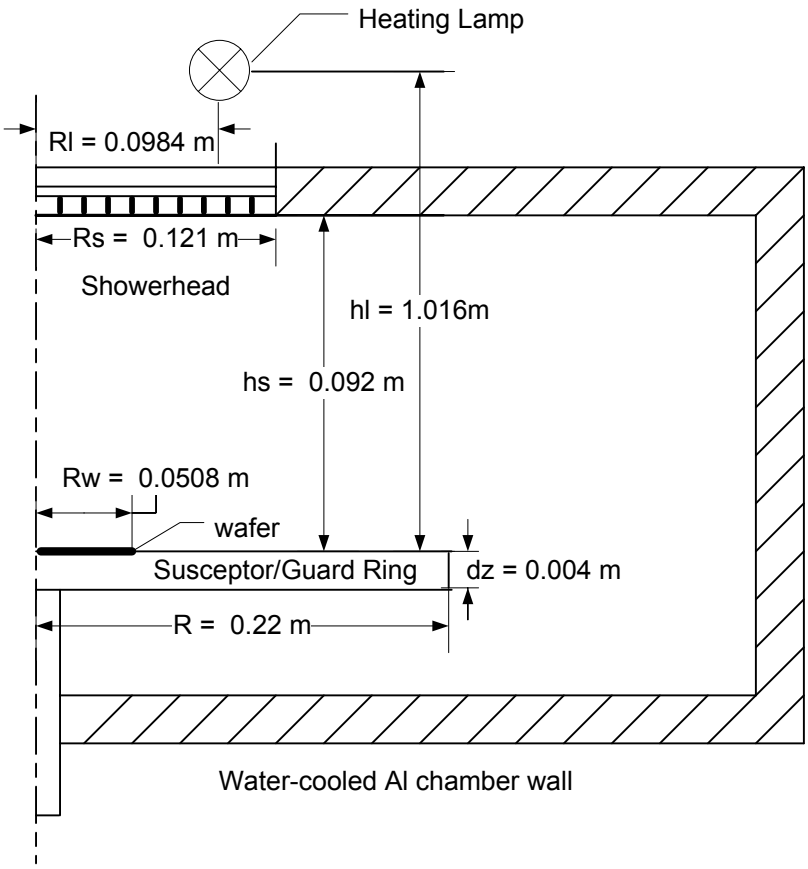

Figure 3.1: The schematic diagram of the CVD reactor chamber

ing the deposition process and temperature distribution on the surface of wafer/guard ring assembly. The formulation of individual modular components, solution of the individual modules to test each model components, and the solutions of the combined modeling system are discussed in turn. Dynamic models of the type ultimately developed are useful for deposition process recipe development.

3.1 Thermal Model of the Wafer/Ring Assembly

To describe the one-dimensional thermal dynamics of the wafer, susceptor and guard ring assembly, several assumptions are made:

- The wafer is thin, and there is no temperature gradient across the wafer thickness;
- We neglect any wafer assembly temperature gradients in azimuthal direction;
- Heat transfer through the wafer assembly vertical support rod is not included.

Under these assumptions, the energy balance of the wafer assembly can be written as:

$$\triangle z\rho\frac{\partial(C_p(T,r)T)}{\partial t} = \frac{\triangle z}{r}\frac{\partial}{\partial r}\left[k(T,r)\,r\frac{\partial T}{\partial r}\right] - \sigma\varepsilon(r)T^4 + Q_{other} \tag{3.1}$$

with boundary conditions and initial condition,

$$r = 0, \quad \frac{\partial T}{\partial r} = 0; \quad r = R, \quad k\frac{\partial T}{\partial r} = -\sigma\varepsilon(R)T^4 + q_c; \quad t = 0, \quad T(r) = T_0$$

The first term on the right hand side of (1) accounts for the conductive heat transfer through the wafer assembly in the radial direction. The second is the radiation from wafer surface to the wafer's environment whose temperature will be initially set to zero. The term Q_{other} represents the net heat flux to the wafer including that exchanged by radiative with other components. The thermal properties of the assembly are functions of temperature and radial position, with a jump discontinuity where the wafer and guard ring meet. The symbol R represents for the guard ring radius, and R_w is the wafer radius. For $r < R_w$,

$$k = 0.1k_w(T) + 0.9k_{gr}(T)$$

$$C_p = 0.1C_{p_w}(T) + 0.9C_{p_{gr}}(T)$$

$$\varepsilon = \varepsilon_w$$

for $r > R_w$,

$$k = k_{gr}; \quad C_p = C_{p_{gr}}; \quad \varepsilon = \varepsilon_q$$

The heat transfer parameters can be found in the paper [3]. For this particular case, Q_{other} is the total energy flux from the heating lamp Q_l, heat exchange between wafer and reactor chamber Q_c, and heat transfer between wafer and gas phase Q_g, i.e.,

$$Q_{other} = \alpha(r)u(t)Q_l(r) + Q_c(T, r) + Q_g(T, x)$$

where $\alpha = \alpha_w$ for $r < R_w$; and $\alpha = \alpha_{gr}$ for $r > Rw$. The time-dependent lamp power controller $u(t)$ is defined as $0 \leq u(t) \leq 1$.

Instead of putting all equations into a single large module to solve this complex system, several smaller and simpler modules are created to describe different heat transfer phenomena. The advantages of doing so include:

1. Module reusability: different CVD reactor systems have different materials of construction and heating systems; for example, instead of using a heating lamp, the wafer could be heated with substrate heater. Separating modules defining the heat sources and other components from the wafer module as much as possible improves the flexibility of model building because reactor systems can be defined simply by combining the required modules from a model library.

2. Convergence problems are more easily debugged by starting with the solution

to a single module, and then adding modules in an incremental fashion. In this way, we can monitor how new heat transfer terms affect wafer temperature profiles and track down the source of solution divergence when it occurs.

3. The approach avoids repeated computations: for example, the heating lamp flux Q_l is not the function of wafer temperature. Solving it once separately from the remainder of the system improves computational efficiency.

Four independent module classes are defined to describe heat transfer in this system: *wafer*, *lampflux*, *chamber* and *recipe*. To demonstrate how the framework is applied for solving systems assembled from separate modules, we will solve the individual *wafer* module first, then add one module at each step and solve the subsystems, finishing with the complete system solved simultaneously. We intentionally use differing and equivalent variable and parameter names to represent temperature or fluxes in different modules to demonstrate how the framework handles these potential sources of problems using the *relation* class.

3.1.1 Wafer Module Solution

The *wafer* class is derived as subclass of *naemodel* and *odemodel* classes. Equation (1) is an implicit function of T, and the discretized version of this equation is stored in the *residual* method. In the constructor method, the heat flux of heating lamp Q_l, radiation between chamber and wafer Q_c, and conduction between gas and wafer Q_g, are defined as parameters. The data type of each is *scalarfield*. The default state of objects of this model assume constant values for each Q. One data

field is defined as an object of *recipe* class, which is used to control the power of the heating lamp; this allows the standalone solution of those objects when no other heat transfer information is present. The sample script is as follows. The source code of the *wafer* class constructor and *residual* methods can be found in the appendix.

```
% create wafer object discretized with N collocation points
A = wafer(N);
% process recipe; lamp power u changes within time interval ti
B = recipe(ti,u);
A = set(A,'param',B,'ru');    % update process recipe in object A
A_s = newton(A);              % solve for steady state solution
A_t = odesolver(A,t);         % solve for dynamic solution
```

To solve partial differential equation (1), the roots of a Jacobi polynomial generated from the *quadgrid* class are used as collocation points; together with the differential operator objects of *linearoperator* class, the PDE and boundary conditions are discretized with this collocation method. The interior residuals (within the defined physical domain) are calculated using the discretized equation (1), and the residuals at the two end points are defined by the boundary conditions. The resulting system becomes a differential-algebraic-equation system for the dynamic case or a non-linear algebraic equation system for the steady state case. To obtain the dynamic solution of wafer temperature, we assume the initial temperature of the wafer surface is 300K, and the power of heating lamp is set to $5000W/m^2$. Figure 3.2 depicts the dynamic wafer temperature distribution when wafer is heated for 10

mins, and the steady state solution of temperature distribution on the wafer surface is shown on Figure 3.4; in both cases the full lamp power is used, i.e.,$u(t) = 1$.

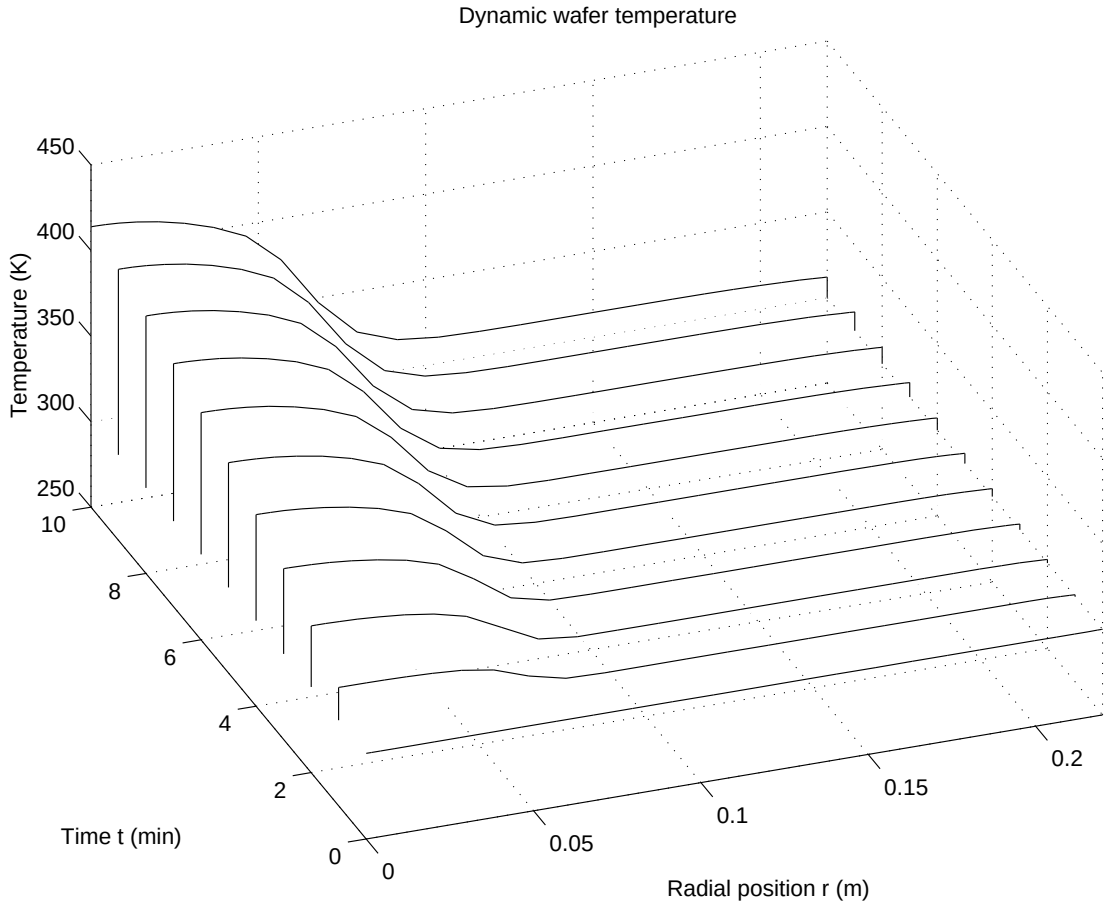

Figure 3.2: Solution of the *wafer* module with heating lamp flux fixed at $5000W/m^2$: The dynamic wafer temperature distribution corresponds to heating the wafer for 10 mins with full lamp power

The results show the temperature on the wafer surface increases with constant heating flux. Because of the larger absorptivity of the wafer relative to the susceptor/guard ring and the contribution of radiant energy exchange of the guard ring edge with the environment, the temperature reaches the maximum at the center of

assembly and gradually decreases to the outer edge.

3.1.2 Solving the wafer and lampflux Modules

The heat flux from the heating lamp to wafer, $Q_l(r)$, which is named as Q in the *lampflux* module, can be computed by (2), for a single lamp ring of radius R_l at a distance h_l from the wafer surface [3]:

$$Q(r) = 0.7\frac{h_l(6,000\text{W})}{(4\pi)(2\pi)} \int_{-\pi}^{\pi} \left[h_l^2 + r^2 + R_l^2 - 2rR_l\cos\theta_l\right]^{-3/2} d\theta_l \qquad (3.2)$$

Because the heat flux from heating lamp is the function of radius only and can be obtained by calling *evalvar* method directly, when constructing the *lampflux* module, the *evalvar* method is overloaded to wrap the equation and the class is derived as subclass of the *solver* class only.

To obtain the wafer temperature distribution using the heat flux from the heating lamp instead of the assumed value, $5000W/m^2$, the *wafer* and *lampflux* modules are solved together. However, creating the object of *modsys* class is not necessary. Because the heat flux of lamp Q is not a function of wafer temperature, we can solve the *lampflux* module first to obtain the flux distribution along wafer radial direction, then solve the wafer temperature by substituting the parameter Q_l in the *wafer* class by Q. The simulation script is as follows:

```
C = lampflux(A);          % create lampflux object based on the wafer model
[A,C] = exchange(A,C,{'Ql','Q'});     % substitute Ql in wafer object by Q
AC_s = newton(A);         % steady state wafer temperature
AC_t = odesolver(A,t);    % dynamic wafer temperature profiles
```

The radiant heat flux of the lamp and wafer steady state and dynamic temperature distribution computed from the more accurate Q are shown in Figure 3.3, Figure 3.4 and Figure 3.8, respectively. The heating lamp flux distribution corresponds to full lamp power. Compared to the solution obtained by setting Q_l constant ($5000W/m^2$), we can see the wafer temperature is higher and less spatially uniform because of larger heat flux Q_l.

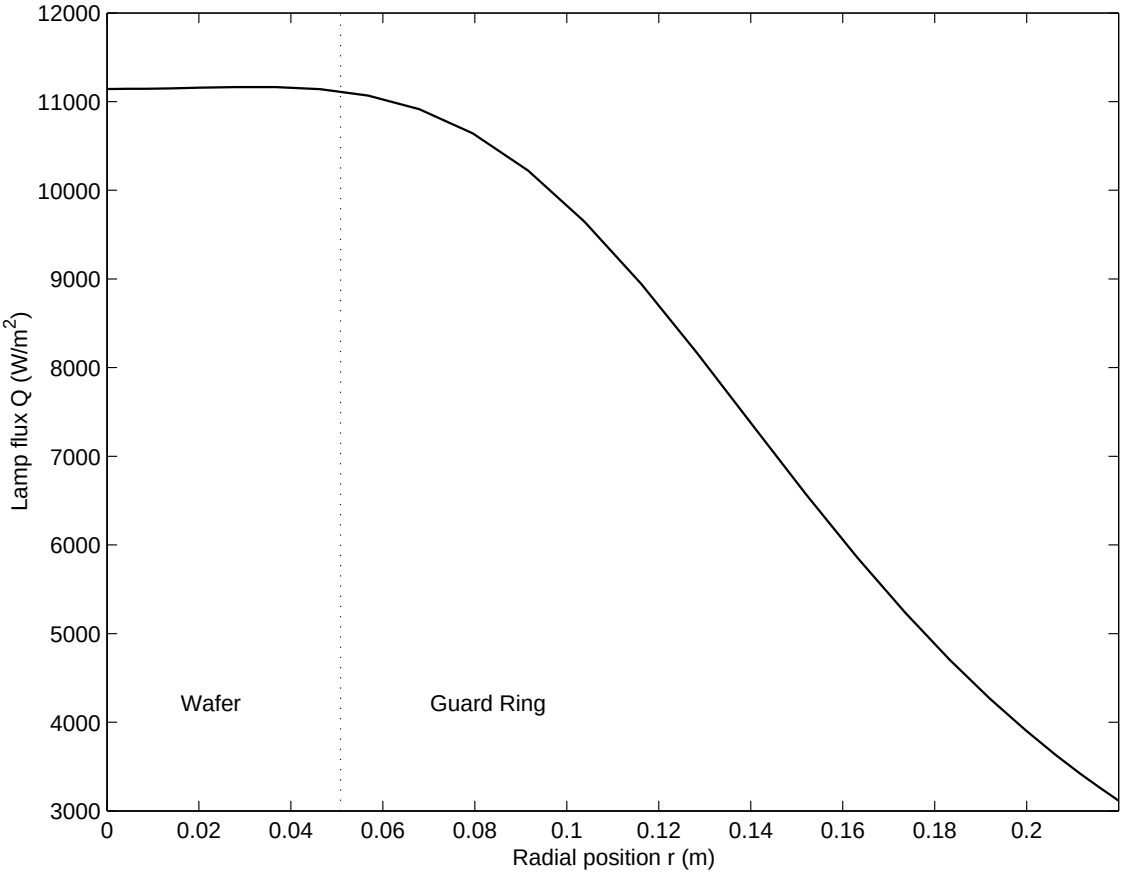

Figure 3.3: Heating lamp radiant flux distribution at the wafer assembly surface; note how it is relatively uniform over the wafer surface

3.1.3 Solving the wafer + lampflux + chamber Subsystem

The radiative heat exchange from the wafer to chamber, Q_c, which is assumed zero in the previous cases, is expressed as Q in the *chamber* module:

$$Q = \sigma\varepsilon(r)\varepsilon_s F_a(r)(T_{sh}^4 - T^4) + \sigma\varepsilon(r)\varepsilon_c(T_c^4 - T^4) + \sigma\varepsilon(r)T^4 \tag{3.3}$$

where ε_s and ε_c are emissivity for the showerhead and aluminum chamber wall. The geometry shape factor $F_a(r)$ is computed using

$$F_a(r) = \frac{1}{2}\left\{1 - \frac{(r/h_s)^2 + 1 - (R_s/h_s)^2}{\sqrt{(r/h_s)^4 + 2[1-(R_s/h_s)^2](r/h_s)^2 + [1+(R_s/h_s)^2]^2}}\right\}$$

The radiative heat flux q_c in the boundary conditions of equation (1) which is assumed zero previously can be obtained by:

$$q = \sigma\varepsilon(R)T^4 + \sigma\varepsilon(R)\varepsilon_c(T_c^4 - T^4)$$

Because Q and q are explicit functions of the state variable, temperature T, they can be computed directly, therefore only *evalvar* method is overloaded and the *chamber* module class is derived as the subclass of *solver* class. To substitute these models into *wafer* module and solve them simultaneously, an instance of a *relation* class is used to indicate the relationship of *var* and *param* in different modules, and an object of *modsys* class is created to define the system to be solved. The script below continues that of the previous sections:

```
D = chamber(A);          % create chamber thermal model object
E = relation(A,D,{'T','Tw','qc','q','Qc','Q'});
F = modsys(E,A,D);       % modular system: wafer + chamber
```

```
F_s = newton(F);          % steady state solution

F_t = odesolver(F,t);     % dynamic solution
```

The simulation results are depicted in Figure 3.4 and Figure 3.8 for the steady and dynamic states, respectively. As shown in the figures, the steady state and dynamic temperatures are higher than that of corresponding cases when solving only *wafer* or *wafer/lampflux* modules. This is because the radiation term of the standalone wafer module assumes no radiation is returned to the wafer. By putting the wafer inside the reactor chamber, the radiative heat flux of the wafer/showerhead and wafer/chamber wall are considered. Similarly, for the q_c in the boundary conditions, heat exchange between assembly outer edge and chamber wall is accounted for when solving combined modules.

3.2 Gas Species Material Balance

When we solve the thermal model of wafer, we assume no gases fill the chamber and the heat transfer between wafer and gas, Q_g, is zero. To consider a dynamic deposition process, we add a material balance model of the reactant and product gases into the previous subsystem. To simplify the complexity of the transport modeling in the gas phase, a CSTR model is used and several assumptions are made for this tungsten chemical vapor deposition process:

- Gases are well mixed in the reactor chamber;
- Gases can be treated as ideal gases for these operating conditions;

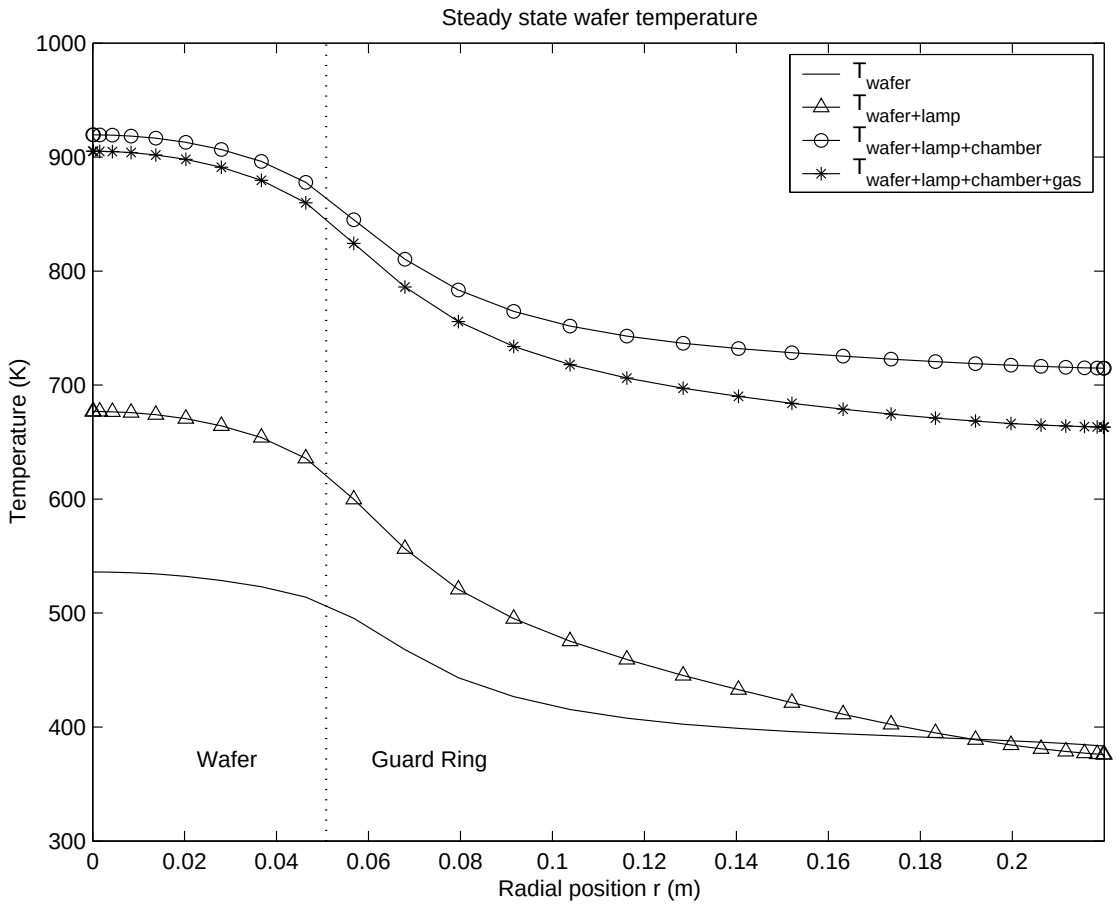

Figure 3.4: Comparison of steady state wafer temperature for different simulation cases with heating lamp set at full power: (a) Solving the *wafer* module only (W); (b) Solving the combined *wafer* and *lampflux* modules (WL); (c) Solving the combined *wafer*, *lampflux*, and *chamber* modules (WLC); (d) Solving the combined mass and thermal models (WLCG)

- No significant temperature or pressure gradients exist within the chamber;
- Gas phase variables are functions of time only, not spatial position;

Based on the gas flow rate, pressure and temperature conditions used in the CVD process, it is reasonable to say the above assumptions are accurate. The reaction occurring on the wafer surface is described as:

$$WF_6 \ + \ 3H_2 \ \rightarrow \ 6HF \ + \ W(s)$$

The empirical expression for reaction rate is chosen as:

$$R_{rxn} = k_0 P_{H_2}^m P_{WF_6}^n \exp\left(-\frac{E_a}{RT_w}\right) \tag{3.4}$$

where the reaction rate order m is usually taken to be $1/2$ and n depends on the ratio of $H_2 : WF_6$, where for high ratio values, n is 0, and for low ratio value, n is $1/6$ [63]. The lumped model for the molar balance of the reaction gases can be written as:

$$\frac{d(Nx_{H_2})}{dt} = F_{in}x_{H_{2,in}} - F_{out}x_{H_2} - 3A_w R_{rxn} \tag{3.5}$$

$$\frac{d(Nx_{WF_6})}{dt} = F_{in}x_{WF_{6,in}} - F_{out}x_{WF_6} - A_w R_{rxn}$$

$$\frac{d(Nx_{HF})}{dt} = -F_{out}x_{HF} + 6A_w R_{rxn}$$

where N is the total number of moles inside the reactor, A_w is wafer area, and F_{in} and F_{out} are molar flow rates calculated by:

$$N = \frac{PV}{RT_g} \tag{3.6}$$

$$T_g = \frac{T_w + T_a}{2}$$

$$F_{in} = F_{out} + 2A_w R_{rxn}$$

where, T_a is the ambient environment temperature. The conductive heat transfer between gas phase and wafer, Q_g, can be approximated by:

$$Q_g(T_w, r, x) = \frac{k_{wg}(x, T_g, P)}{L_{ww}}(T_g - T_w) \tag{3.7}$$

The gas thermal conductivity k_{wg} is a function of gas composition, temperature, and pressure; L_{ww} represents the length scale of the gas thermal boundary layer. For a given wafer temperature, the steady state and dynamic solutions of gas composition can be obtained by solving above nonlinear algebraic or differential-algebraic equations.

A module class called *rxngas* is constructed to wrap the mass balance model. The modeling equations from (3.4) to (3.7) are stored in the class method *residual*. Other utility methods such as *rxnrate*, *growthrate*, and *residtime* are created to facilitate the computation and simulation results retrieval. A sample script for solving *rxngas* module with constant wafer temperature is shown below:

```
G = gascomp(A,D);          % create gas composition model object
G_s = newton(G);           % steady state solution
G_d = odesolver(G,t);      % dynamic solution
```

Figure 3.5 depicts the gas composition profiles for solving a *rxngas* module under the operation conditions: $P = 0.5torr$, $T_a = 300K$, $T_w = 673K$, $q_{tot} = 50sccm$ (H_2:WF_6 = 4:1). During the first 10 mins, only H_2 is introduced to the chamber, after which the gas flow of WF_6 is turned on. The deposition process lasts 10 mins,

after which the WF_6 gas is turned off and only the H_2 gas is used to flush the chamber. At the steady state, for $m = 1/2$, $n = 1/6$, $x_{H_2} = 0.7570$, $x_{WF_6} = 0.1869$, $x_{HF} = 0.0560$, the tungsten growth rate is 25.0 nm/min. This simulation result is close to the average deposition rate of 21.6nm/min, experimental results produced under the same operation conditions except the wafer temperature setting was $773K$ during experiments [108]. It is reasonable to set the wafer temperature to $673K$ in the simulator because the true wafer temperature normally is at least 100K lower than the controller setting in the particular reactor studied [25].

3.3 Combined Thermal and Mass Balance Model Solution

During the deposition process, the gas phase concentration changes due to reaction and the process recipe, gas phase thermal conductivity changes with composition, and the reaction rate is a function of wafer temperature. Therefore, the one-dimensional thermal model and lumped mass balance model become coupled through gas composition x and wafer temperature T. To solve this combined modeling system, the total reaction rate is obtained through the integration of wafer temperature along the radial direction, i.e.,

$$R_{tot} = \int_0^{R_w} R_{rxn} 2\pi r dr$$

where R_{tot} represents the product of $A_w R_{rxn}$. The script for the main program is as follows:

```
H = recipe(ti,xfeed);          % process recipe object
G  = set(G,'param',H,'Rf');    % set process recipe in chamber model object
```

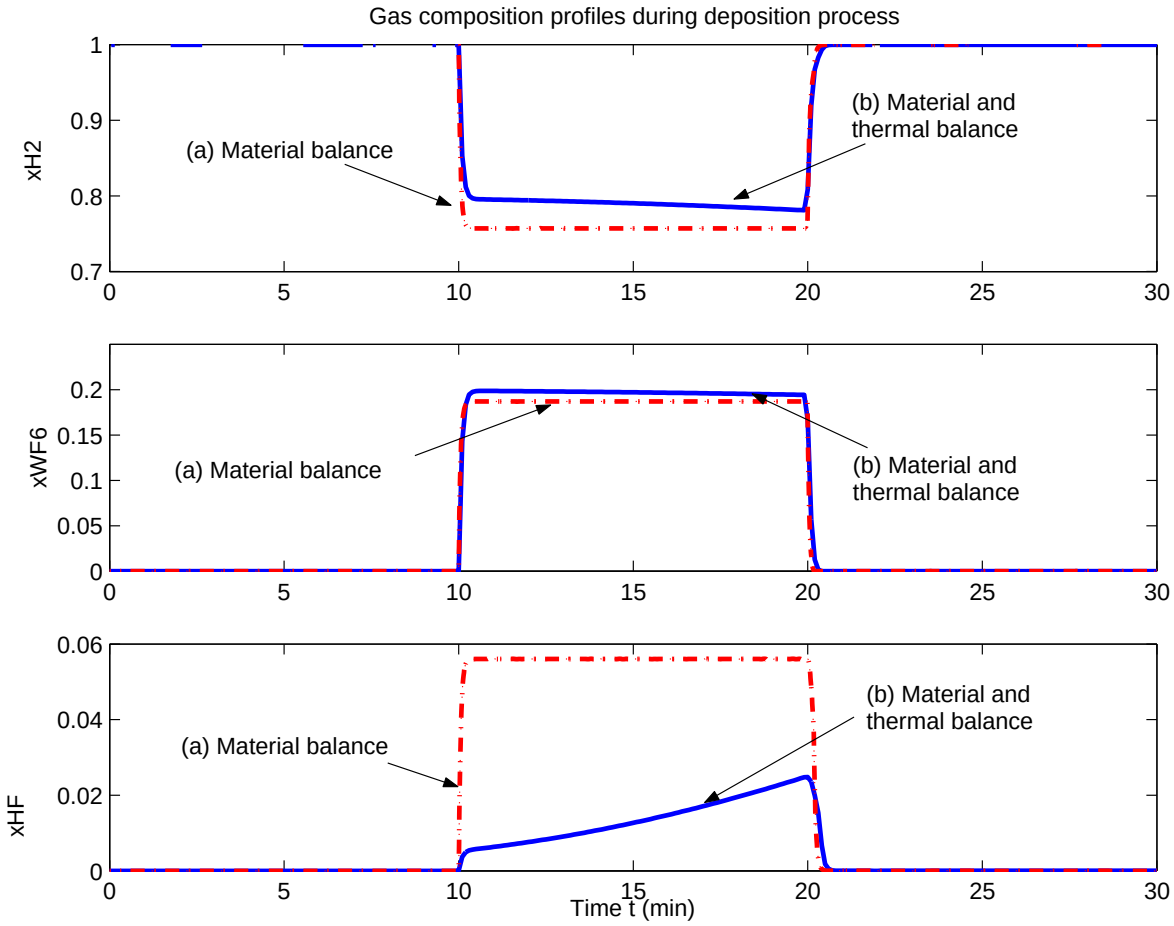

Figure 3.5: Comparison of gas composition profiles during deposition process: (a) Solving the *rxngas* module at constant wafer temperature, $T_w = 673K$. For the first 10 mins, only H_2 is induced to the chamber; then WF_6 is introduced to the chamber during 10 mins deposition process after which the WF_6 is shut off; (b) Solving the combined mass and thermal models where the wafer is heated with full lamp power for 10 mins while only H_2 filled the chamber, followed by introducing WF_6 for a 10 mins deposition reaction with 0.7 heating lamp power, ending with the shut off of WF_6 and chamber purged with H_2, cooling the wafer 10 mins

```
E(2) = relation(A,G,{'T','Tw','Qg','Qg')});
I = modsys(E,A,D,G);          % complete CVD system model
I_s = newton(I);              % steady state solution
I_d = odesolver(I,t);         % dynamic simulation
```

Figure 3.6 illustrates the class diagram of this assembled modular system. To test the integrated modeling system, a complete deposition process is simulated. The process cycle consists of:

1. Load the wafer and introduce precursor gas H_2 to purge the reactor, preheating the wafer for 10 minutes with full lamp power;

2. Introduce the gas WF_6 to the chamber with the heating lamp controller maintained at 0.7. The reaction starts and deposition process lasts 10 minutes;

3. Terminate the WF_6 flow, turn off the lamp, and introduce a large flow of H_2 to cool down the wafer for 10 minutes prior to removing it.

The transient phases of gas composition and dynamic and steady state wafer temperature distribution are shown in Figure 3.5, Figure 3.7 and Figure 3.4, respectively. The steady state gas compositions are: $x_{H_2} = 0.4356$, $x_{WF_6} = 0.0891$ and $x_{HF} = 0.4753$, where $m = 1/2$ and $n = 1/6$.

Comparing the transient gas profiles when the *rxngas* module is solved alone with the combined mass and thermal system, we observe that the conversion rate of WF_6 is higher and the gas phase steady state is reached quickly for the first situation because the wafer temperature is constant at $673K$. For the second case,

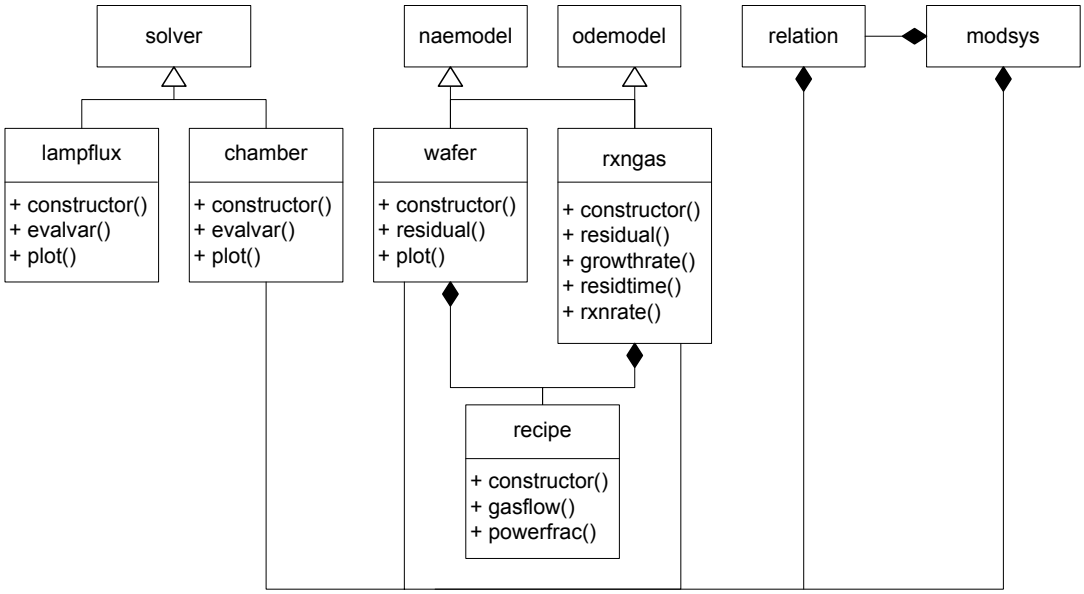

Figure 3.6: The class diagram of the CVD simulation system assembled from modular objects of *wafer*, *lampflux*, *chamber*, *recipe*, and *rxngas* classes

at the time the precursor gas WF_6 is introduced into the chamber to start the deposition process, the wafer surface temperature is only about $580K$ after 10 mins of preheating. Although the generation rate of HF climbs with increasing wafer temperature, the deposition rate is still much lower relative to the first case because at the end of deposition the wafer temperature can not reach $673K$ and only is $660K$.

The steady state wafer temperature and snapshots of transient wafer temperature corresponding to the four different simulation cases are plotted in Figure 3.4 and Figure 3.8, respectively. They show the wafer temperature changes with the heating power and environment. In the first simulation case, the wafer temperature

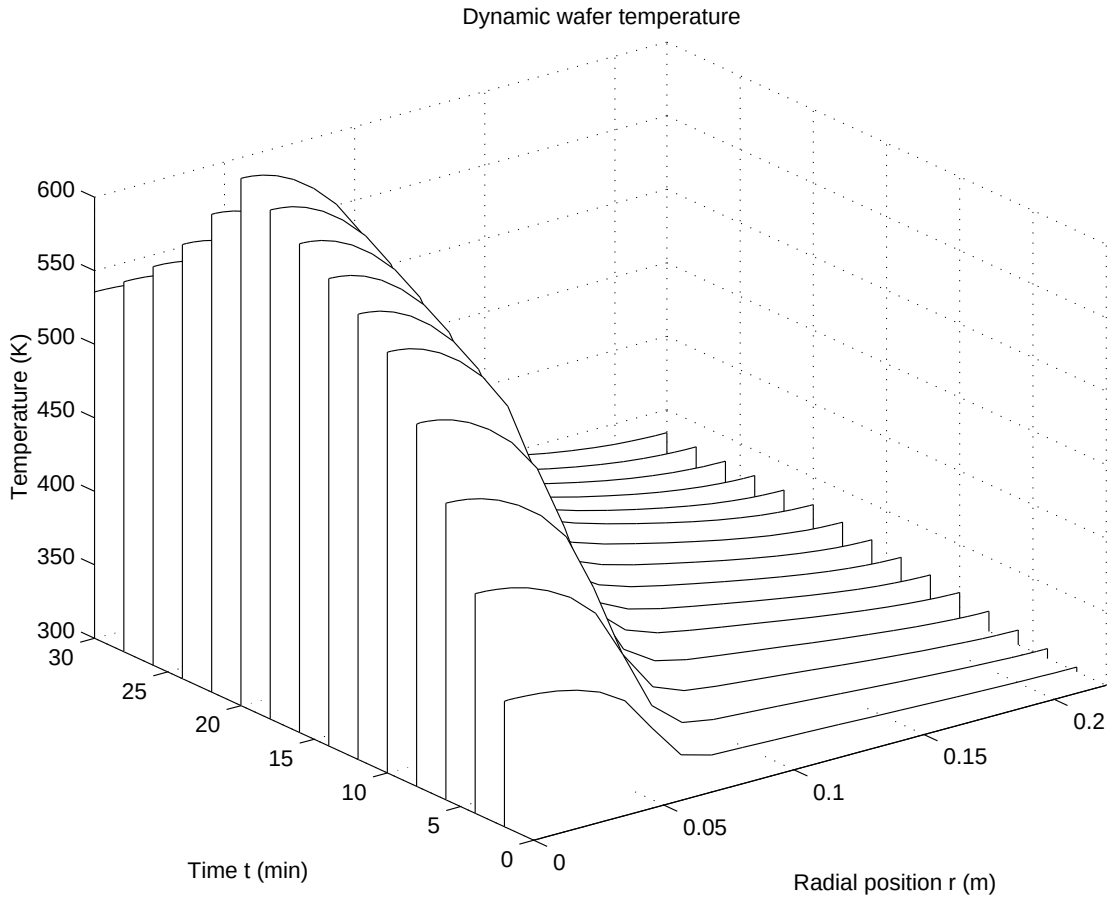

Figure 3.7: Solving the combined mass and thermal models: Dynamic wafer temperature distribution of the wafer heated with full lamp power for 10 mins while only H_2 filled the chamber, followed by introducing WF_6 for a 10 mins deposition reaction with 0.7 heating lamp power, ending with the shut off of WF_6 and chamber purged with H_2, cooling the wafer for 10 mins

is lower than that found by combining the *lampflux* and *wafer* modules because we assume the heating lamp power is constant $5000W/m^2$, which is much lower than the true value distributed over the wafer surface, ranging from $11000W/m^2$ to $3000W/m^2$. After adding the *chamber* module, the wafer temperature increases because the heat lost in the *wafer* module is partly compensated by the chamber wall radiation. Finally, the wafer temperature slightly decreases when the *rxngas* module is added because of wafer cooling by conduction through the gas phase.

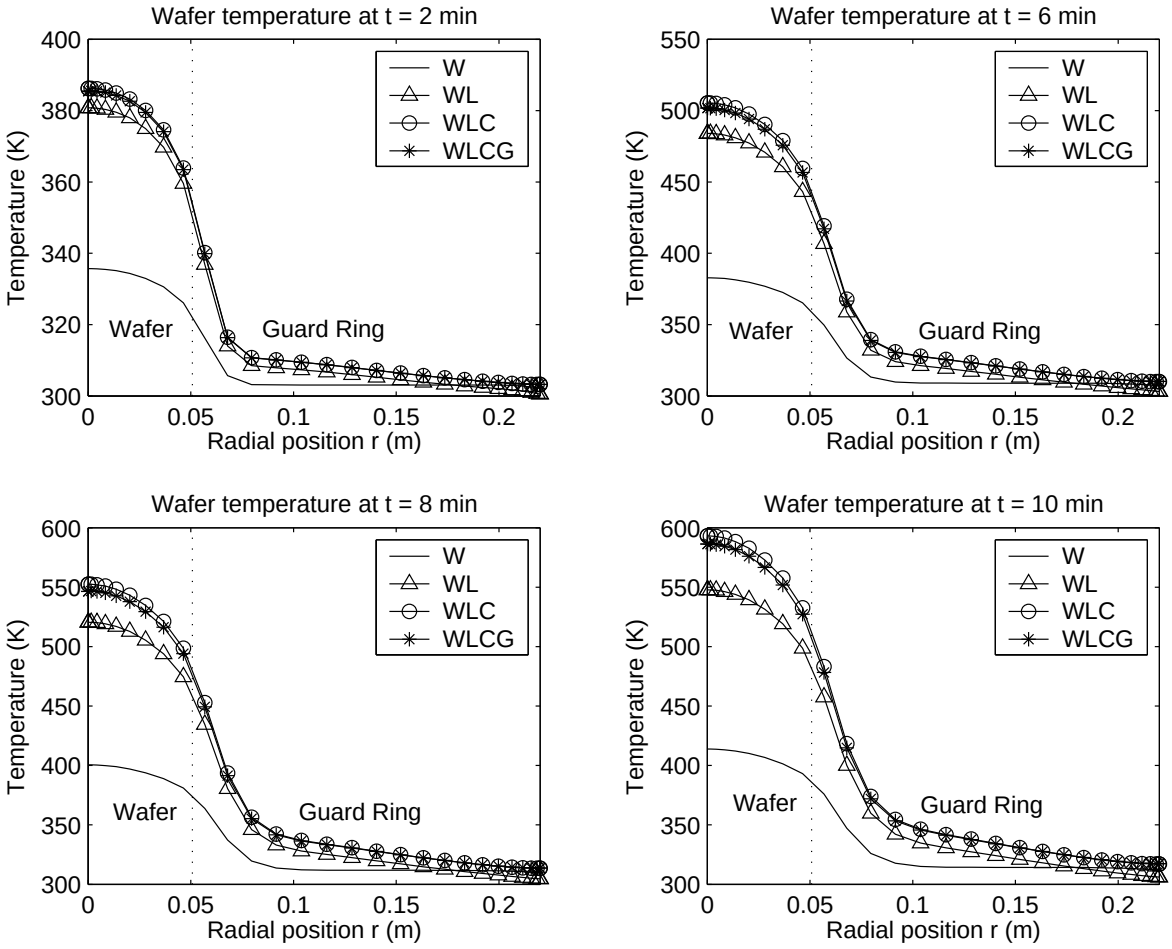

Figure 3.8: Snapshot of wafer temperature at different times (t=2,6,8,10 min) for the four simulation cases with heating lamp set at full power: (a) Solving the *wafer* module only; (b) Solving the combined *wafer* and *lampflux* modules; (c) Solving the combined *wafer*, *lampflux*, and *chamber* modules; (d) Solving the combined mass and thermal models

Chapter 4

Simulation-based Design and Analysis of the Programmable CVD System

This chapter discusses the development of a simulator describing the reaction gas transport in the multi-segmented showerhead for better understanding and control of gas composition across a wafer surface in a spatially controllable or programmable CVD reactor system using the modular simulation approach. The experimental data obtained from the three-segment prototype system are used to validate the simulation models and assumptions.

4.1 Introduction of the Programmable CVD System

The Programmable CVD reactor (PCVD) shown in Figure 4.1, a spatially controllable CVD system [27], was developed at the University of Maryland. Compared to conventional CVD reactors, this new reactor system features spatially-tunable process recipes. Equipped with a distributed sensing system, the design of programmable CVD system aims to achieve several goals, including: (1) across-wafer uniformity and desired product performance of deposited thin film; (2) controlled nonuniformity across the wafer for combinatorial study of complex materials and processes; (3) advanced process control through in-situ sensors.

The main components of the programmable CVD system include: (1) a feed

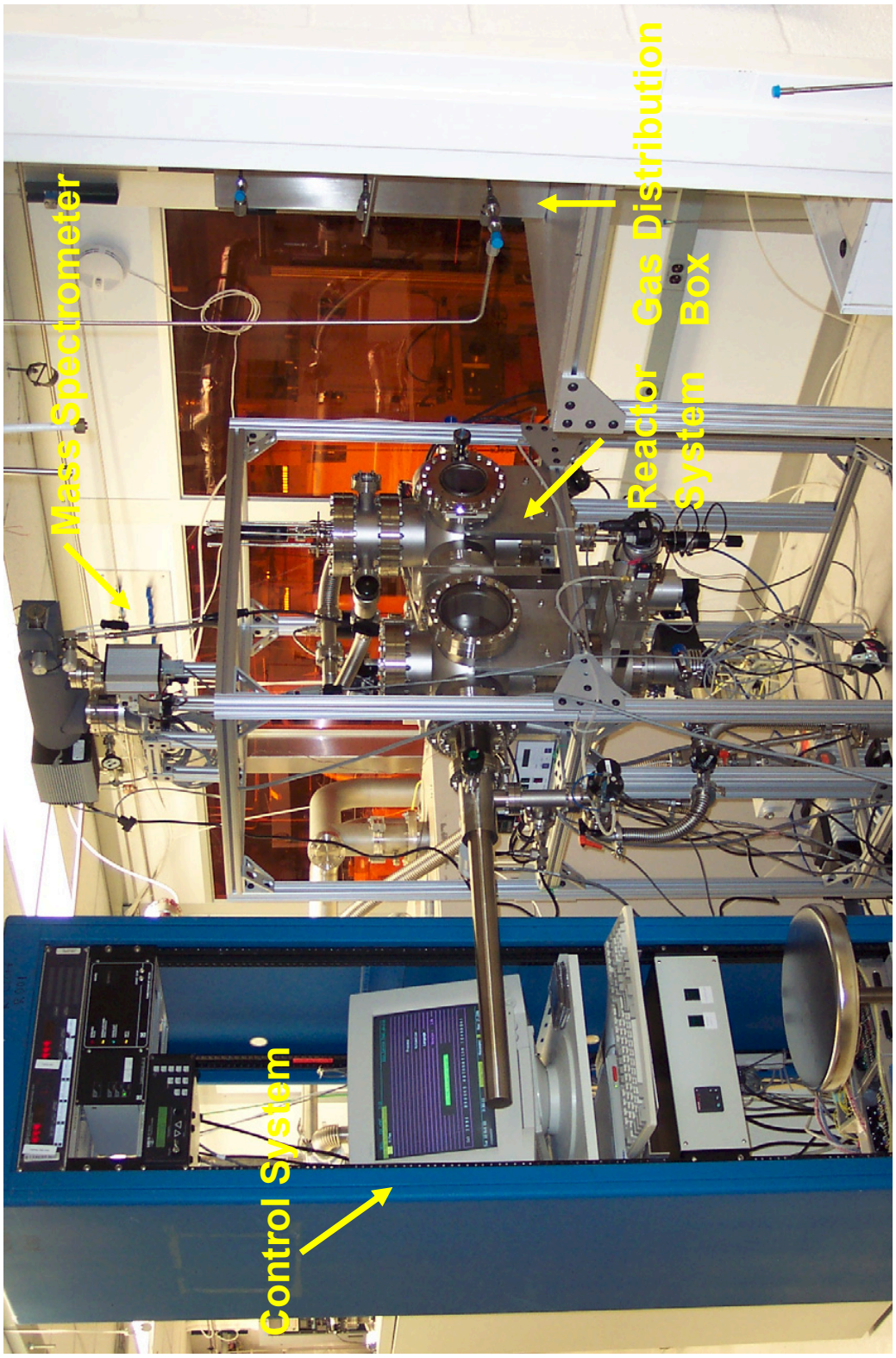

Figure 4.1: The programmable chemical vapor deposition reactor system

gas distribution box; (2) a process control system; (3) a mass spectrometer system for sensing gas composition; (4) and a deposition system including one load lock chamber and one reactor chamber. The key design feature of this system is its segmented showerhead which makes possible true 2-dimensional control of gas phase composition across the wafer surface. As shown in Figure 4.2, each segment of the showerhead includes individually controllable gas feeds, and exhaust gases are recirculated through the showerhead itself to the common exhaust zone. This construction enables control of spatial distribution of gas flow rate and minimizes the interaction among the segments.

To proceed with a deposition process, a process recipe including pressure, temperature, precursor gas flow rates, deposition time, etc. is entered as input to the control system. The precursor gases are introduced into the segmented showerhead through feed tubes, and the residual gases are redirected and pumped out through each segment. The deposition is performed on the wafer surface which is heated by the substrate heater. The gas composition in each segment is monitored with respective mass spectrometry sampling tube during operation.

To gain the insight into the operation of this CVD system and quantify the relative importance of different precursor transport and reaction mechanisms inside this reactor system, a set of simulation tools is necessary to supplement experimental methods.

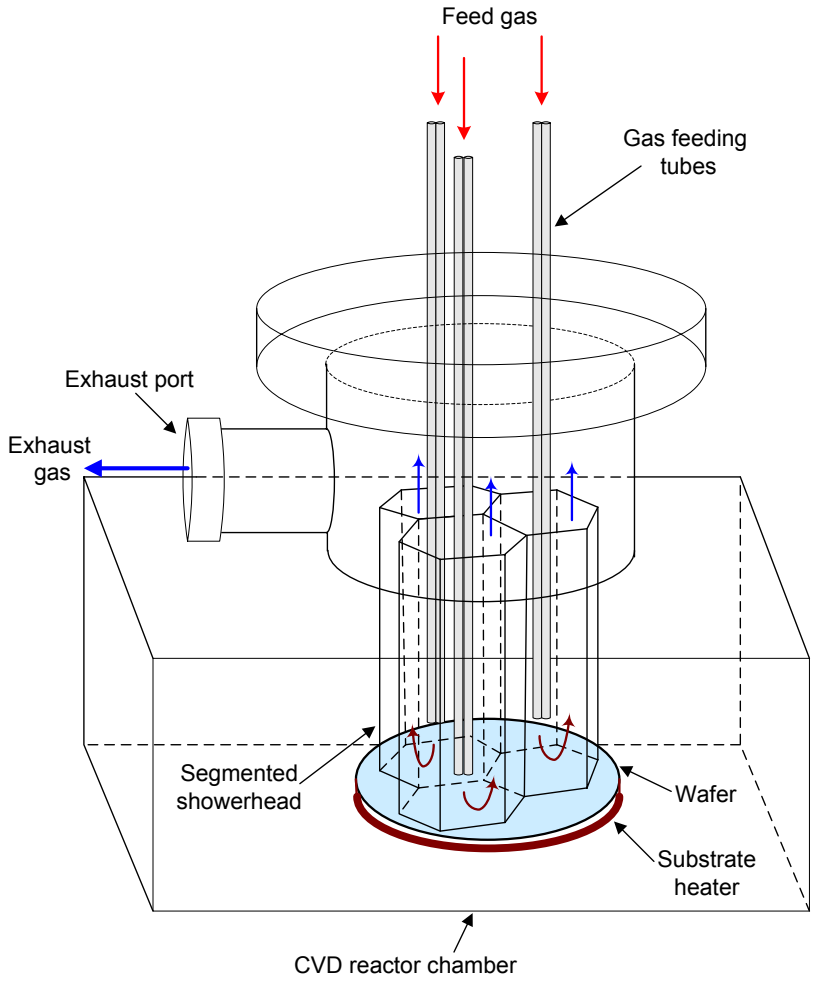

Figure 4.2: The schematic diagram of three-segment programmable CVD system

4.2 Modeling and Simulation of the Multi-segment CVD System

Although the simulation will not completely replace the need for experiments, simulators can result in very substantial cost savings in developing new generation CVD reactor systems and in solving manufacturing problems [86]. Because of the complexity of this system, conventional CVD simulation tools may not be appropriate to design and analyze the programmable CVD system.

To model gas species transport within each individual segment, the Maxwell-Stefan equation is employed to describe multicomponent gas species transport:

$$\nabla x_i^k = \sum_{j=1}^{n} \left[\frac{1}{CD_{ij}} (x_i^k N_j^k - x_j^k N_i^k) + \frac{x_i^k x_j^k}{D_{ij}} \left(\frac{D_j^T}{\rho_j} - \frac{D_i^T}{\rho_i} \right) \nabla lnT \right], \quad 0 < z < L \quad (4.1)$$

subject to boundary conditions

$$x_i^k(L) = x_i^{exit} \quad (4.2)$$

where, x_i^k is the mole fraction of each gas species in each segment. The subscript i refers gas H_2, WF_6, and Ar, and superscript k is the segment number. To find the one-dimensional gas compositions distribution along segment length, $n_g * n$ coupled nonlinear ODEs must be solved simultaneously, where n is the number of segments, and n_g represents the number of precursor gases. While current three-segment design is intended to demonstrate the programmable CVD concept and prove its feasibility, multi-segment construction is required for manufacturing level implementation.

4.2.1 Challenges in Building a Multi-segment CVD Simulator

To develop a flexible simulator for understanding gas transport within the multi-segment showerhead, the challenges we face include:

- A wide range in the number of segments: the number of variables and corresponding equations depends on the number of segments. The work necessary to modify the simulator due to adding or removing segments should be minimized.

- Diverse patterns of segment arrangements (see Figure 4.3): spatially controllable process recipes across wafer surface leads to the intersegment diffusion because of gas concentration gradients. To optimize the deposition performance, it may be necessary to control the impact of neighboring segments. Assuming different patterns of arrangement of segments through the simulator will provide an easy and low-cost solution. One issue arising from dynamic arrangement of segments is how to determine the outside segments; when we consider the interaction of gas diffusion between gap region and chamber, only the gases in the gap region corresponding to the very outside segments will be influenced from chamber, and those segments surrounded by other segments need only consider the gas impacts from neighboring segments.

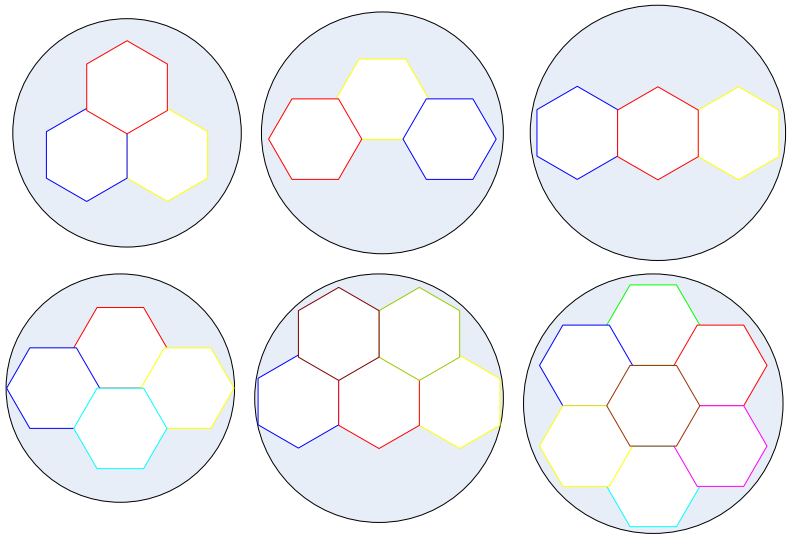

Figure 4.3: Representative patterns of segment arrangements

- Model validation: a number of assumptions such as ideal gas state, well-mixed

gases in the exhaust region and chamber, linear temperature distribution along segment length, reaction mechanism on the wafer surface, etc., made to simplify the current model and simulation, will be validated through the experimental results. Thus, this requires that the simulator should have such capabilities to simplify testing of assumptions by making it easy to incorporate new models or assumptions.

- Simulator reusability: the newly designed programmable CVD system is undergoing concept test, feasibility assessment, equipment upgrade, etc. to improve and optimize the system performance. Keeping simulation components reusable to the greatest extent possible will shorten the design process and lower future development costs.

4.2.2 Construction of Simulator Modules

The design of the programmable CVD reactor lends itself very naturally to a modular approach because of the segmented showerhead design - while each segment is a fairly complex component, with individually controllable reactant gas supply and residual gas exhaust, the design of each segment is identical.

Using the modular simulation framework we developed earlier, a new simulation can be constructed relatively easy through the integration of user selected modular components. Several small and simple modules have been constructed in replace of developing a complicated simulation program to wrap all modeling equations. Key advantages of this approach are: (1) the redundant work of rewriting

similar equations with different variables can be avoided; (2) it allows the rapid replacement of individual simulator elements, making it possible to easily assess modeling assumptions and the relative importance of the elements that make up a complete simulation.

In the following, we introduce the model development along with the construction of individual modules. The building blocks of the modeling system are depicted in Figure 4.4, they are: a *segment* module describing steady-state one-dimensional transport of gases in an individual segment; a *topmix* module assuming well-mixed gases in common exhaust zone; a *gap* module describing the inter-segment gas diffusion in the gap region between the wafer and segmented showerhead, and gas diffusion between gap and reaction chamber; a *chamber* module depicting gas state and compositions in the reaction chamber.

The *segment* module class: This module includes information on describing intra-segment transport, in which the multicomponent gas species transport can be expressed by the Maxwell-Stefan equation 4.1. Rearranging the equations and defining the flux as the combination of ordinary diffusion and thermal diffusion, the equation 4.1 can be presented in a simplified form:

$$\nabla x_i^k = \sum_{j=1}^{n} \frac{1}{CD_{ij}} (x_i^k \bar{N}_j^k - x_j^k \bar{N}_i^k) \tag{4.3}$$

where

$$\bar{N}_i^k = N_i^k + \frac{D_i^T}{M_i} \nabla lnT$$

In the constructor method of this class, gas compositions x_i^k are defined as variables, all others are classified as parameters, who are either constant (e.g. seg-

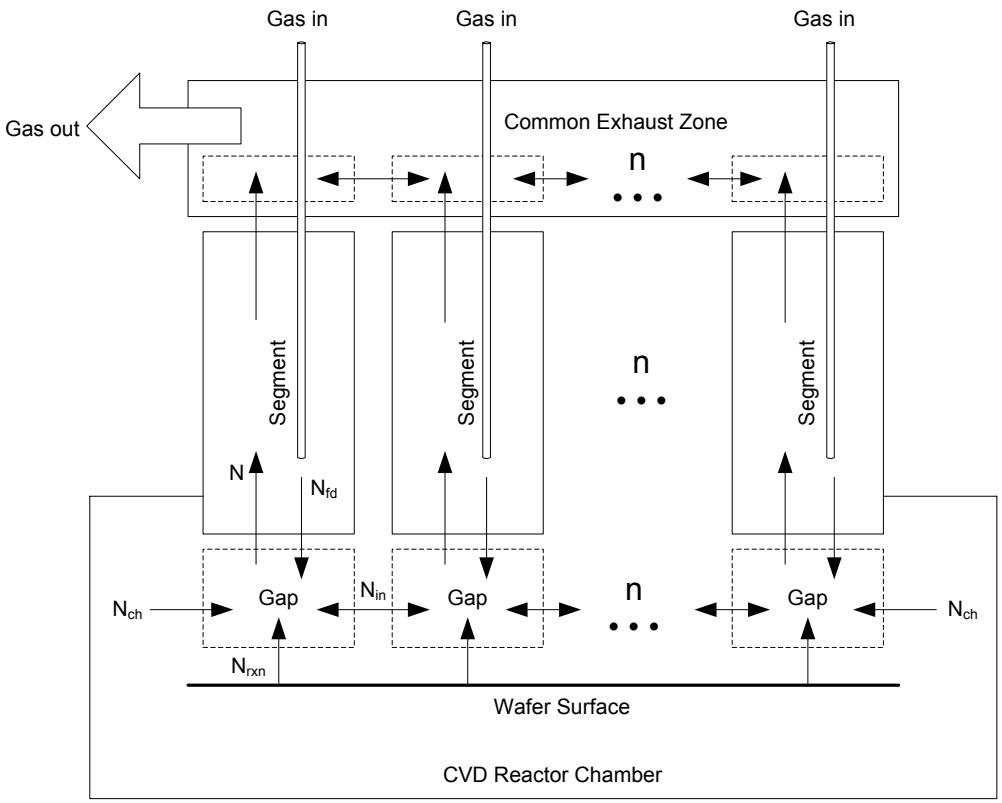

Figure 4.4: The building blocks of programmable CVD modeling system

ment and feeding tube geometry information) or have hidden relationships with state variables, or must be updated by exchanging information with parameters or variables in other modules; see Table 4.1.

In the *residual* method of *segment* class, the equation 4.3 is defined, in which the parameters who are functions of state variables are updated during each computational step. The geometry diagram of one segment is depicted in Figure 4.5.

The *gap* module class: The gap region is defined as the area between the segment bottom to the wafer surface, whose distance could be zero millimeter to several millimeter depending on the experimental settings. Because of various oper-

Table 4.1: List of part of parameters in the *segment* class

Parameter	*Description*	*Value*
L_{seg}	segment length	constant
L_{ftw}	distance from feeding tube bottom to wafer	constant
A_s	segment area	constant
A_f	feeding tube bundle area	constant
D_i^T	multicomponent thermal diffusivity for species i	$D_i^T = f(x_i, T, P)$
D_{ij}	binary diffusivity	$D_{ij} = f(x_i, T, P)$
C	total concentration of gas	$C = f(x_i, T, P)$
N_i^k	ordinary diffusion flux	from *gap* class
x_i^{exit}	mole fraction of species i in exhaust zone	from *topmix* class

ating conditions for each segment and the feature of reversed residual gas flow, we divided the gap region into several small virtual hexagon blocks corresponding to each segments (Figure 4.4).

Models developed in this module are used to compute the total flux contribution from gap area to the segmented showerhead. Using a CSTR-type model for gas transport, the lumped mass balance in each block can be written in the form of:

$$N_i^k = \frac{hW}{A_s}(\sum_{m=1}^{n} N_{i,in}^{k-m} + N_{i,ch}^{k-c}) + N_{i,fd}^k + N_{i,rxn}^k \tag{4.4}$$

where, k is the segment number, i refers to the gas species, n is the total number of segments, h is gap distance between wafer surface and segment bottom, and W is segment side length.

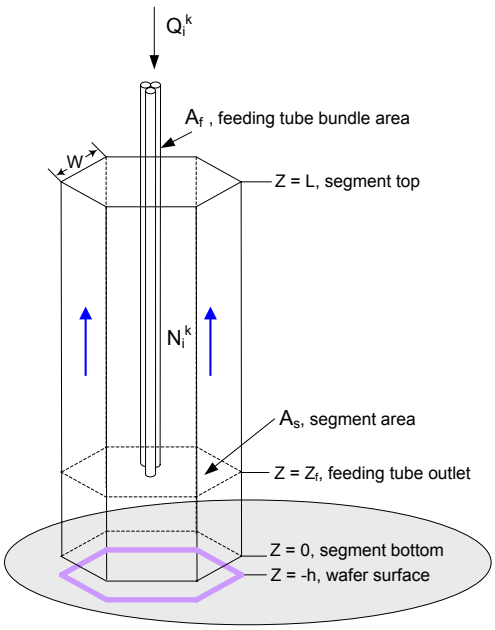

Figure 4.5: The geometry diagram of one individual segment

$N_{i,in}^{k-m}$ represents the inter-segment diffusion flux at the bottom of segments, and it can be written as:

$$N_{i,in}^{k-m} = CD_{i,j} f_d \frac{x_i^m(0) - x_i^k(0)}{2Wcos(\pi/6)}$$

where, f_d is the gap factor to correct intersegment flux, and it can be estimated from experimental data. For the current simulation study, we set it to unity.

Likewise, the diffusion flux from the chamber to the gap region, $N_{i,ch}^{k-c}$, can be defined as:

$$N_{i,ch}^{k-c} = CD_{i,j} \frac{x_i^c(ch) - x_i^k(0)}{2Wcos(\pi/6)}$$

$N_{i,ch}^{k-c}$ is zero for the segments where all six sides are surrounded by other segments. $N_{i,fd}^{k}$ is the flux input from the segment feeding tubes. It is zero for

$z < z_f$, and for $z \geq z_f$, it is defined by:

$$N_{i,fd}^k = \frac{Q_i^k \rho_i}{MW_i(A_s - A_f)}$$

where, Q_i^k is species i flow rate at segment k, and ρ_i and MW_i are species i density and molecular weight, respectively.

$N_{i,rxn}^k$ is the flux from the deposition reaction on the wafer surface. It can be obtained from the deposition reaction rate:

$$N_{i,rxn}^k = R_{rxn}$$

In the constructor method of the *gap* class, the total flux N_i^k is defined as a variable. The values of constant parameters such as wafer temperature, operating pressure, segment geometry and pattern, etc., are acquired when creating instances of the class. The dynamic parameter values can be obtained through the information exchange between modules; see Table 4.2

Table 4.2: List of part of parameters in the *gap* class

Parameter	*Description*	*Value*
x_{ch}	mole fraction of species in chamber	from *chamber* class
x	mole fraction of species at the bottom of segments	from *segment* class

Because the variable N_i^k can be expressed by an explicit function, the equation 4.4 is put into the method of *evalvar* instead of *residual*. To analyze the effects of chamber gas diffusing into the gap region and deposition reaction, the resulting flux $N_{i,ch}^{k-c}$ and $N_{i,rxn}^k$ can be turned on/off in the *evalvar* method. The method *growthrate*

defined in this class is used to calculate deposition rate and test different reaction mechanism, which can be easily switched on/off.

The *topmix* module class: It is used for studying the gas composition x_i^{exit} at the top common exhaust area, and it provides boundary conditions for the ODEs in the *segment* class. For current simulations, the common exhaust volume is treated as perfectly mixed. The exhaust volume composition x_i^{exit} is computed as the average of the feed compositions to each segment because of relatively low depletion rate of reactions. Therefore, in the constructor method, it is identified as an parameter whose value can be obtained when creating objects of this class, and no variables are defined here. Consequently, no *residual* or *evalvar* methods are required in this class. One can either place an empty *residual* method in the class definition or it will inherit one from its base class, *solver*.

The *chamber* module class: this class is for investigating gas composition in the reaction chamber. It provides information to *gap* class for computing diffusion flux from the chamber to the gap region. The flow rate of gas transport to the chamber is determined by the gap size. The assumption of perfect gas mixing is made in the chamber. Likewise, because of the relatively low deposition rate under the tested operating conditions, the gas composition x_i^{cham} is taken as the average of the feed compositions. This assumption will be validated later. Similarly, in the constructor method, only parameters x_i^{cham} whose values can be acquired through the input of creating the class objects are defined, and no variables are needed. Therefore, definition of the *residual* method is not required.

It may appear unnecessary to construct *topmix* and *chamber* modules for the

current simplified modeling, and we could simply hard-code those information in the classes requiring them. The benefit for defing the modules is that it simplifies susequent modifications of respective models and facilitates the testing of assumptions for the common mixing region and reaction chamber without disturbing anything in other modules.

The *prgcvd* module class: This class is optional. It has been developed mainly for hiding the relationship among the objects of the module classes described above, which can be described in the main program. By doing so, only the object of *prgcvd* class must be instantiated, which simplifies the main program implementation. Several utility methods are developed in this class, such as *plot* for plotting solutions of gas composition profiles along segment length and the segment arrangement patterns, *segdeprate* outputs information on each segment deposition rate, *setpattern* is for setting segment patterns, and so on.

The class diagram in Figure 4.6 shows the relationship of constructed module classes. The simulation framework developed can be used for n-segmented showerhead CVD reactor simulation with any desired alignment pattern, which greatly facilitates the test of different reactor design ideas and saves substantial cost, labor and time of experiments.

4.2.3 Solving the Multi-Segment CVD System

To obtain the gas profiles within each segment of the programmable CVD reactor, the following steps in the main program will be taken:

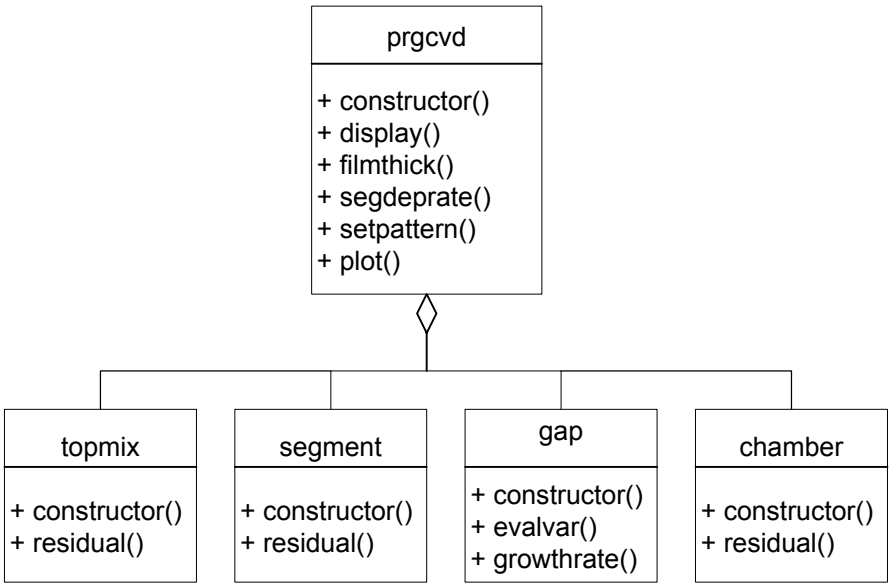

Figure 4.6: The class diagram of the programmable CVD system

- Set operating conditions, such as pressure, wafer temperature, gap distance, gas flow rates, etc., from user input or to be retrieved from data file in XML format;

- Create objects of *segment* class. The number of objects is equal to the number of showerhead segments;

- Define the showerhead pattern by identifying the relative positions of each segment;

- Create an object of *prgcvd* class by providing the *segment* objects and their pattern, gap distance between showerhead and wafer surface, etc., as inputs;

- Solve the modeling system defined by the *prgcvd* object using the Newton's method of *naemodel* class;

- Plot the gas composition profiles.

As mentioned earlier, the $n_g * n$ nonlinear coupled ODEs (Equation 4.1) must be solved simultaneously subject to boundary conditions (Equation 4.2). The collocation method is used to discretize the ODE system, which results in $n_g * n * n_{col}$ nonlinear algebraic equations, where n_{col} is the number of collocation points located within the domain of interest. Combined with two interval endpoints at which the residuals are defined by the boundary conditions, the nonlinear AE system is solved using Newton's method by driving all residuals to zero. The information exchange among the modules during the computation is similar to that described in Chapter 2.

For demonstration purposes, a three-segment alignment pattern is created, as depicted in Figure 4.7 and Figure 4.8. The operating conditions for this system are: total pressure $P = 0.5torr$, wafer temperature $T_w = 673K$, gap distance $h = 5mm$, flow rate in segment 1: $H_2 = 0$; $WF_6 = 0$; $Ar = 50sccm$, flow rate in segment 2: $H_2 = 0$; $WF_6 = 50sccm$; $Ar = 0$, flow rate in segment 3: $H_2 = 50sccm$; $WF_6 = 0$; $Ar = 0$. The number of collocation points is 20.

As shown in Figure 4.8, all three gas concentrations converged to the same values at the gas exits of each segment because the complete mixing assumption is made, that in the exhaust zone, the gas composition is the average of all feed gas compositions. Although in each segment, the dominant gas is the feed species, the other two gases entered the segments through the inter-segment diffusion in the gap and back diffusion from the top of the segments. The back diffusion effect is reduced

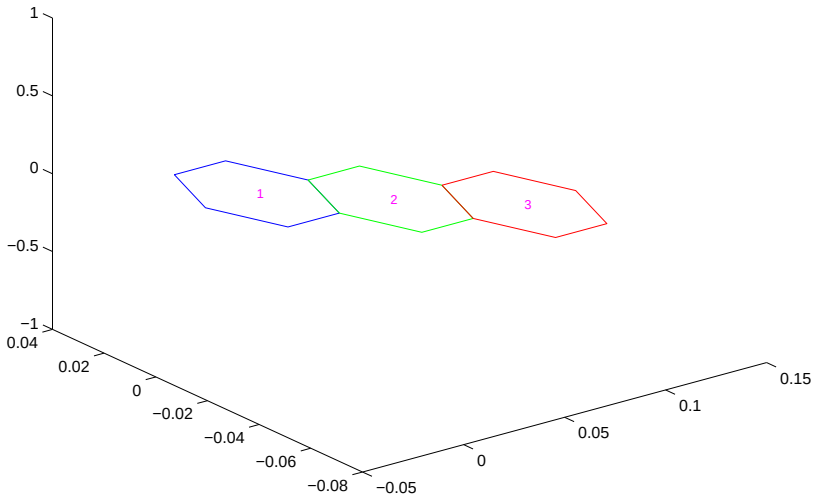

Figure 4.7: The alignment pattern of three-segment design

at higher feed flow rates [27]. Because of the larger value of diffusivity of H_2, as shown in the figure, H_2 diffuses more extensively relative to the other two gases which results in H_2 having a higher concentration than that of WF_6 in segment 1 and Ar in segment 2. The gap distance from the segment bottom to wafer surface is $5mm$, which results in considerable inter-segment diffusion and diffusion between the reactor chamber and the gap region. The contributions from reaction depletion is small compared to other transport effects.

4.3 Model Validation in the Three-segment Programmable CVD System

The three-segment prototype of the programmable CVD system had been built and deposition experiments have been performed to demonstrate the programmable

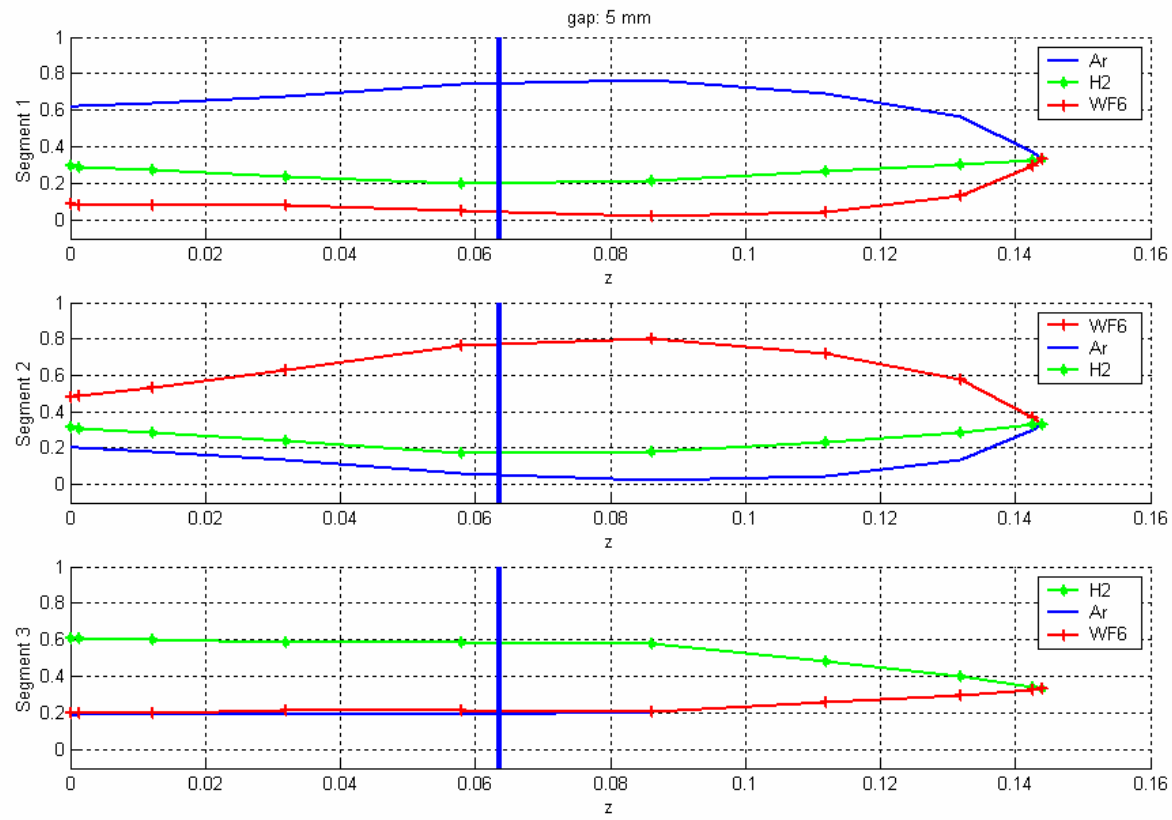

Figure 4.8: Gas composition profiles as the function of position within each segment

concept feasibility [29]. The experimental results, including the metrology data of deposited tungsten (W) film thickness measured by a four point probe sensor, the true W thickness measured by scanning electron microscope (SEM), and gas distribution in each segment obtained from mass spectrometry signals, are used to validate model's assumptions and accuracy.

4.3.1 Gas Concentration Profiles along Vertical Segments

As seen in Figure 4.8, the simulator predicts the gas concentration profiles along segment height based on the specified operating conditions. In the experi-

ments, a mass spectrometer was used to monitor the electrical signal as it is related to the partial pressure of the measured gas in a segment. One experiment was conducted to measure gas distribution at the different vertical positions within one segment, and the mass spectrometry data was used to evaluate the model's predictions.

Because of the safety concerns during movement of the sampling tube related to potential leaks at the O-ring seals, only inert gases Ar and H_2 were used for this study [21]. The experiment recipes are shown in Table 4.3.

Table 4.3: Experimental recipe for gas concentration profile measurement: Gap = 1mm, Pressure = 1 torr, Temperature = room temperature.

Feed gas / Flow rate (sccm)	*Seg1*	*Seg2*	*Seg3*
Ar	60	30	60
H_2	0	30	0

The feed tube outlet is 2.25" away from the segment bottom and the sampling tube was moved up from 0.5" to 4.5", as measured from the segment bottom. The mass spectrometry signal of current was recorded in segment 2 and converted into the mole fraction. The comparison of the simulation profiles and experimental data is depicted in Figure 4.9.

Due to the small gap size (1mm), the magnitude of the inter-segment diffusion is small, but not insignificant. As we can see from the Figure 4.9, at the bottom of segment 2, the Ar concentration is a bit higher than at the feed tube outlet because

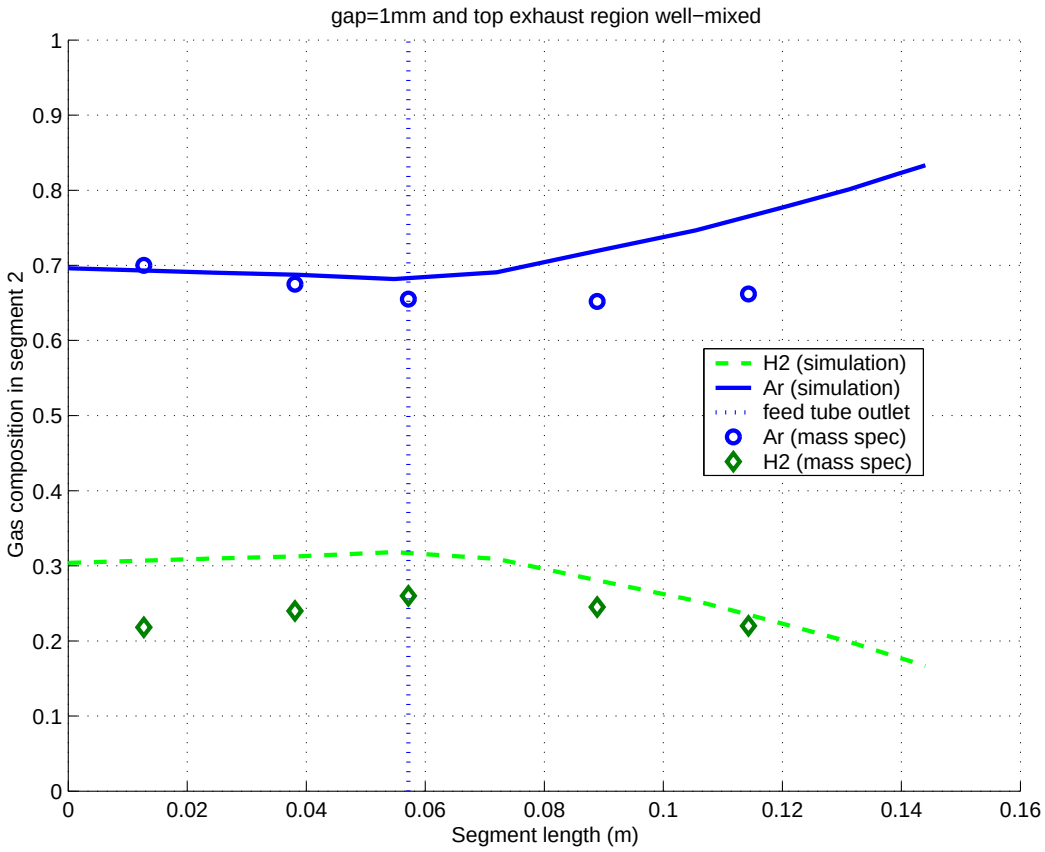

Figure 4.9: The simulated gas composition profiles vs. experimental measurements by mass spectrometry (one object created for the exhaust volume area). Sum of squre error (SSE) = 0.0156 for Ar simulation

of the diffusion from segment 1 and segment 3 where Ar is the dominant gas, while H_2 composition at the bottom of segment 2 is a slightly lower than at the feed tube position as a result of the diffusion of H_2 to segment 1 and segment 3 where initially no H_2 was introduced. Based on the perfectly mixed assumption for the top exhaust region, one object of *topmix* module was created for the simulation, and the average value of the feed compositions was taken as the exhaust gas composition. Although the comparison of experimental data and simulation results shows the

same trend in the gas concentration profiles along the vertical position in segment 2, the obvious difference of Ar composition at the top of the segment calls into questions the validity of the well-mixed assumption.

As a result of this discrepancy, the simulator was modified by assuming that in the exhaust region, gases become well-mixed after a certain height from the exit of the segment, but in the area immediately adjacent to the exit of the segments, gases are not completely mixed and the gas compositions are influenced by the inter-segment diffusion on the top area. Three objects of *topmix* module were created, and the exhaust volume compositions of Ar and H_2 were approximated as 0.75 and 0.25 for segment 2, 0.95 and 0.05 for segment 1 and segment 3, respectively. The comparison of new simulation results vs. mass spectrometry data is shown in Figure 4.10. It shows great improvements in matching simulation results with experimental data. The sum squared error (SSE) of Ar simulation vs. mass spectrometry measurement data is 0.0156 and 0.0042 for the well- and not well-mixed assumptions, respectively. In conclusion, the exhaust gases appear to be not well-mixed in the region immediately next to the segment exits.

Although both Ar and H_2 mass spectrometry measurements give same trend as simulation profiles, we tested our model assumptions based on the match between the Ar experimental data and simulation results. This is because the sum of the mole fractions of H_2 and Ar experimental data are not unity, we believe Ar's signal is more dependable because of its concentration higher resulting in a stronger current signal relative to $H2$, resulting in less relative measurement errors.

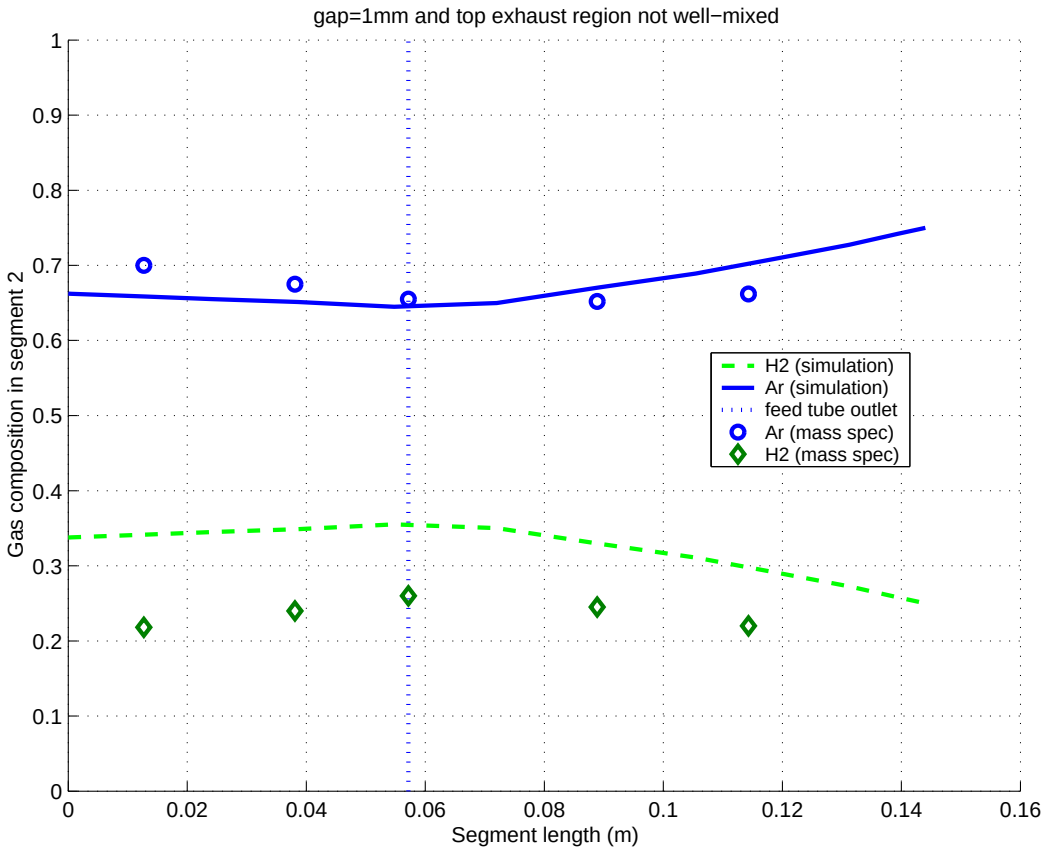

Figure 4.10: The improved simulation gas composition profiles vs. experimental measurements by mass spectrometry (three objects created for the exhaust volume area). Sum of square error (SSE) = 0.0042 for Ar simulation

4.3.2 Kinetic Rate Mechanism Validation through Uniform Deposition Experiments

The programmable CVD system is designed to be able to spatially control uniformity and nonuniformity of thin films across the wafer surface. A uniform deposition experiment was conducted to measure the segment-to-segment uniformity. The process recipe can be found in Table 4.4 [28].

Table 4.4: Experimental recipe for uniform tungsten deposition: Pressure = 1 torr, Heater temperature = 673 K, Gap = 1mm.

Feed gas / Flow rate (sccm)	*Seg1*	*Seg2*	*Seg3*
Ar	30	30	30
H_2	24	24	24
WF_6	6	6	6

W film was generated by H_2 reduction of WF_6, and the reaction is:

$$WF_6 \ + \ 3H_2 \ \rightarrow \ 6HF \ + \ W(s)$$

Numerous studies have been performed to investigate the reaction rate mechanisms of hydrogen reduction of tungsten hexafluoride. It has been argued that under the operating conditions of LPCVD (Low Pressure CVD), the reaction rate is primarily a function of temperature and H_2 concentration [15, 74], however the overall rate can not be independent of the WF_6 pressure at very low WF_6 concentration because the supply and mass transfer become reaction limiting steps [81, 61], so both H_2 and WF_6 effects on the reaction rate should be considered [22, 75]. Representative models are summarized as follows. The reaction rate is in nm/s, pressure in $torr$ (except for the first group of rate constants in the equation 4.5, in P_a), temperature in K, and activation energy of the reaction in J/mol.

- An empirical rate expression form, with the coefficients listed in the Table 4.5:

$$R_{rxn} = k_0 P_{H_2}^m P_{WF_6}^n \exp\left(-\frac{E_a}{RT}\right) \tag{4.5}$$

Table 4.5: Rate constants for the empirical rate expression

Ref.	*m*	*n*	*k0*	*Ea*
1 [74]	1/2	0	6.8×10^4	73,000
2 [15]	1/2	0	1.16×10^7	68,500
3 [81]	1/2	1/6	3500±400	64,000

- A modified rate from which decreases to zero as the partial pressure of WF_6 approaches zero [22]:

$$R_{rxn} = 1.2\exp\left(-8800/T\right)\frac{P_{WF_6}P_{H_2}}{1+150P_{WF_6}} \tag{4.6}$$

- A rate expression indicating WF_6 adsorption is the rate limiting step in the mass transfer controlled regime and HF desorption is the one when the surface is saturated with WF_6 [75].

$$R_{rxn} = [\frac{1}{k_1 P_{WF_6}} + \frac{1}{AP_{WF_6}^{1/6}P_{H_2}^{1/2}\exp\left(-E_a/RT\right)}]^{-1} \tag{4.7}$$

  where, $A = (7.17 \pm 0.20) \times 10^6$, and $k_1 = (2.61 \pm 0.23) \times 10^5$.

These reaction rate models are included in the *growthrate* method of *gap* module. And each model was tested by turning off other models during computation. The film thickness in each segment are calculated by the *filmthick* method in the *prgcvd* module. Based on the previous experiments, the wafer temperature in the simulator is set to 355^oC corresponding to 400^oC setpoint of heater temperature [28]. The gases in the reactor chamber are assumed well-mixed and the gas compositions are taken as the average of the total flow rate to the three segments.

This assumption is valid because of the equipment preconditioning. The purpose of preconditioning is to reduce contaminants inside reactor chamber and gas delivery system and allow the mass spectrometry signal to reach stable status. The parameter settings (P, T, gas flow rate ratio, etc.) for the conditioning process are the same as that for deposition process and one or two dummy wafers are deposited. The conditioning procedure usually takes about two to three hours during which the inside walls of the reactor chamber are saturated with reactant and byproduct gases [21].

The deposition time for all processe is $10min$. The average film thickness measured by four point probe (4pp), the true film thickness range estimated from the scanning electron microscope (SEM) measurement of films produced in nonuniformity experiments, and the simulated thickness using different reaction models at different wafer temperatures are listed in Table 4.6.

As shown in the table, we used model 4.5-(2) to analyze the effect of chamber gas diffusion on the gap region within segmented showerhead and wafer surface by turning on and off the *chamber* module in the simulator. The little difference on the deposited film thickness shows the gas interaction between gap and reactor chamber is small. The small gap size, $1mm$, is the main reason.

The average thickness is computed from 4pp measurements. The output of 4pp is the sheet resistance, and it can be converted to film thickness by dividing film resistivity with sheet resistance. Since the film resistivity is not available, we use bulk resistivity of W ($5.6\mu\Omega \cdot cm$) as an estimation to determine the film thickness. The true thickness can be obtained by adjusting the 4pp results by multiplying a factor

Table 4.6: Comparison of uniform deposition experimental data of W film thickness measured by four-point probe (4pp) with simulation results: Gap = 1mm, Pressure = 1 torr, Wafer temperature is to be determined, chamber gas diffusion is counted (i.e., ch:on).

Average thickness (nm)	*Seg1 (355^oC/380^oC)*	*Seg2 (355^oC/380^oC)*	*Seg3 (355^oC/380^oC)*
4pp measurement	154.5	162.6	158.8
True thickness range	420 - 580	420 - 580	420 - 580
Model 4.5, (1)	279.9/480.0	279.9/480.0	279.9/480.0
Model 4.5, (1)	460.4 (378^oC)	460.4 (378^oC)	460.4 (378^oC)
Model 4.5, (2)	163.3/271.1	163.3/271.1	163.3/271.1
Model 4.5, (2) (ch: off)	165.1/274.3	165.1/274.3	165.1/274.3
Model 4.5, (3)	125.3/199.9	125.3/199.9	125.3/199.9
Model 4.6	234.8/400.5	234.8/400.5	234.8/400.5
Model 4.7	146.9/234.1	146.9/234.1	146.9/234.1

of 2 to 5, which is the range of the ratio of true film resistivity/bulk resistivity. The true thickness range listed in the table is assessed based on the SEM measurement for nonuniform deposition experiment results [28].

According to the Choo's paper [28], he suggested that the actual wafer temperature is approximately 355^oC for the 400^oC set point of heater temperature. We evaluated different reaction rate models using this temperature, and all results showed low film thickness. The wafer temperature approximation of Choo may

be inaccurate because it was measured using H_2, N_2, or Ar under non-deposition conditions. Because gas heat conductivity, gap size, gas composition, and thermal conductivity of mixture gases will all cause significant temperature differences, one important application of the simulator is to find the true wafer temperature by comparing simulation results with experimental data. Therefore, we also estimated deposition rates at 378^oC and 380^oC wafer temperatures. As a result, we found model 4.5 with rate constants (1) gives the closest approximation to the true thickness at the 378^oC and 380^oC wafer temperatures. We use this model to predict film thickness for subsequent non-uniform experiments.

4.3.3 Model Predictions for Nonuniform Deposition Experiments

To demonstrate the deposition programmability of the programmable CVD system, an intentional nonuniform deposition experiment was performed. The process recipe is listed on Table 4.7 [28].

Table 4.7: Experimental recipe for nonuniform tungsten deposition: Pressure = 1 torr, Heater temperature = 673 K, Gap = 1mm.

Feed gas / Flow rate (sccm)	*Seg1*	*Seg2*	*Seg3*
Ar	60	30	0
H_2	0	24	48
WF_6	0	6	12

As indicated in the previous section, the simulator gives a better prediction

of deposition rate using model 4.5 with rate constants (1). Hence, we choose this model for current simulation. Under the nonuniform experimental conditions, we computed film thickness at different wafer temperatures, ranging from 355^oC to 380^oC, and also evaluated the well-mixed or not well-mixed assumption for the top exhaust area. The comparison of simulation and experimental measurements are listed in the Table 4.8.

Table 4.8: Comparison of nonuniform deposition experimental results of W film thickness [28] with simulation results: Gap = 1mm, Pressure = 1 torr, Wafer temperature is to be determined, chamber gas diffusion is counted (i.e., ch:on), rate expression: Model 4.5-(1).

Average thickness (nm)	*Seg1*	*Seg2*	*Seg3*
4pp measurement	56.2	146.1	180.7
SEM	270	420	580
T=355°C, 3 topmix objects	139.8	278.7	350.0
T=375°C, 3 topmix objects	217.3	430.4	539.4
T=378°C, 3 topmix objects	231.5	458.3	574.3
T=379°C, 3 topmix objects	236.5	468.0	586.3
T=380°C, 3 topmix objects	241.5	477.8	598.5
T=380°C, 1 topmix object	375.1	480.2	570.2

It can be found from the comparison of true thickness values with simulated values that the 378^oC wafer temperature setting and not well-mixed top area as-

sumption gave the closest estimation of thickness to the true value. For not well-mixed assumption, we created three objects of *topmix* class, and set the gas compositions in each object is equal to the feed composition on each segment. This is not an accurate approximation, the results will be greatly improved if we can adjust the gas compositions at the top area by considering inter-segment diffusion. Overall, this experiment confirmed the rate model we chose, the true wafer temperature corresponding to 400^oC heater setting, and the assumptions we made for chamber and common exhaust volume. Again, this simulator demonstrates its capability to predict the deposition results and to describe the transport phenomena during deposition process.

4.3.4 Conclusions

By comparing three different deposition experiments conducted in three segment prototype system, the simulation model has been validated. Several comments on the model are to be made:

- An appropriate deposition reaction mechanism has been determined for the programmable CVD system under current operating conditions.
- The well-mixed assumption in the reactor chamber is valid based on the preconditioning process.
- The assumption of perfectly mixed gases in the top area of segments can be held only from some certain level height in the common exhaust volume. The higher the total gas flow rate fed into each segment, the thicker the not

well-mixed layer. This parameter estimation can be acquired by fitting the experimental data to the model. A simple binary flux model can be used to describe the inter-segment diffusion in this top area and will improve the model's precision.

- Wafer temperature is determined through the model validation by experimental results. The true wafer temperature is approximately 378^oC for heater setting 400^oC.

In general, this object-oriented modular simulator developed for the programmable CVD system demonstrates its capability of accurately predicting deposition rate, film thickness and concentration profiles along the segment height. The simulation results can be used to guide the selection of experimental operating conditions for run-to-run control. It also improves the optimization efficiency and reduces the experimental cost and development cycle of new system design. The simulator is easy to maintain and adjust to adapt to the equipment modifications.

Chapter 5

XML-based Information System for the Programmable CVD System

This chapter discusses how the Java and XML-based information techniques are applied for data management for the programmable CVD system. First, we will consider the motivation for this research, followed by an overview of JAVA and XML applications. Then the three-tier information framework is presented. The framework functionalities are demonstrated with applications to a three-segment prototype of the programmable CVD system.

5.1 Research Motivation

When working on the design and analysis of the programmable CVD system, one major concern came up repeatedly, i.e., the management and sharing of different format data from different sources (Figure 5.1). Specifically, several challenges come to light:

- Different data sources: How to store data from different sources while recording all related information to the data for post-processing becomes challenging. For example, for the prototype I of the programmable CVD system, (1) the film thickness had been measured in selected seven points for each segment, which requires that the coordinate of each point should be recorded and mapped with that point thickness; (2) the experimental conditions should be associated with

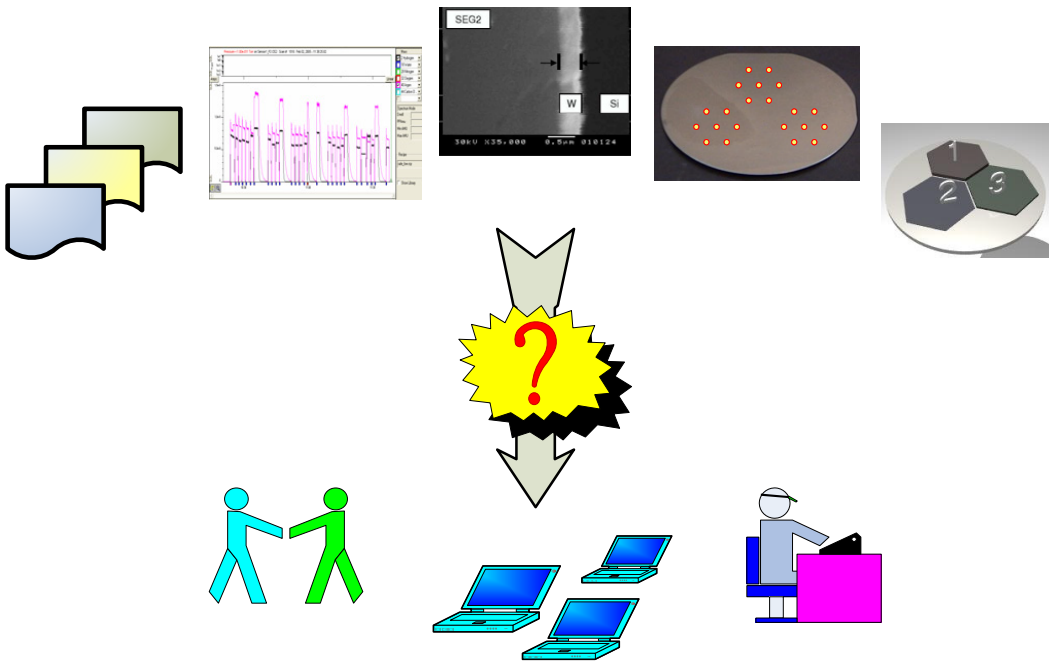

Figure 5.1: The challenges of different format data management and sharing within the research groups

the deposited wafer photo. For the prototype II design, (3) the film thickness had been measured in 30 by 30 matrix format; (4) the raw mass spectrometry data requires preprocessing before further use. Figure 5.2 is a snapshot of a typical mass spectrometry data file.

It is seen from the file that hundreds of items of data are saved for each running process recipe, and the process recipe was recorded in a separate place manually. The measurement data information were marked at the end of the file, such as when data were taken, under preconditioning or deposition process, where they were taken, i.e., in exhaust area or inside a specific segment, if the data is invalid because of other operations during process, and so on.

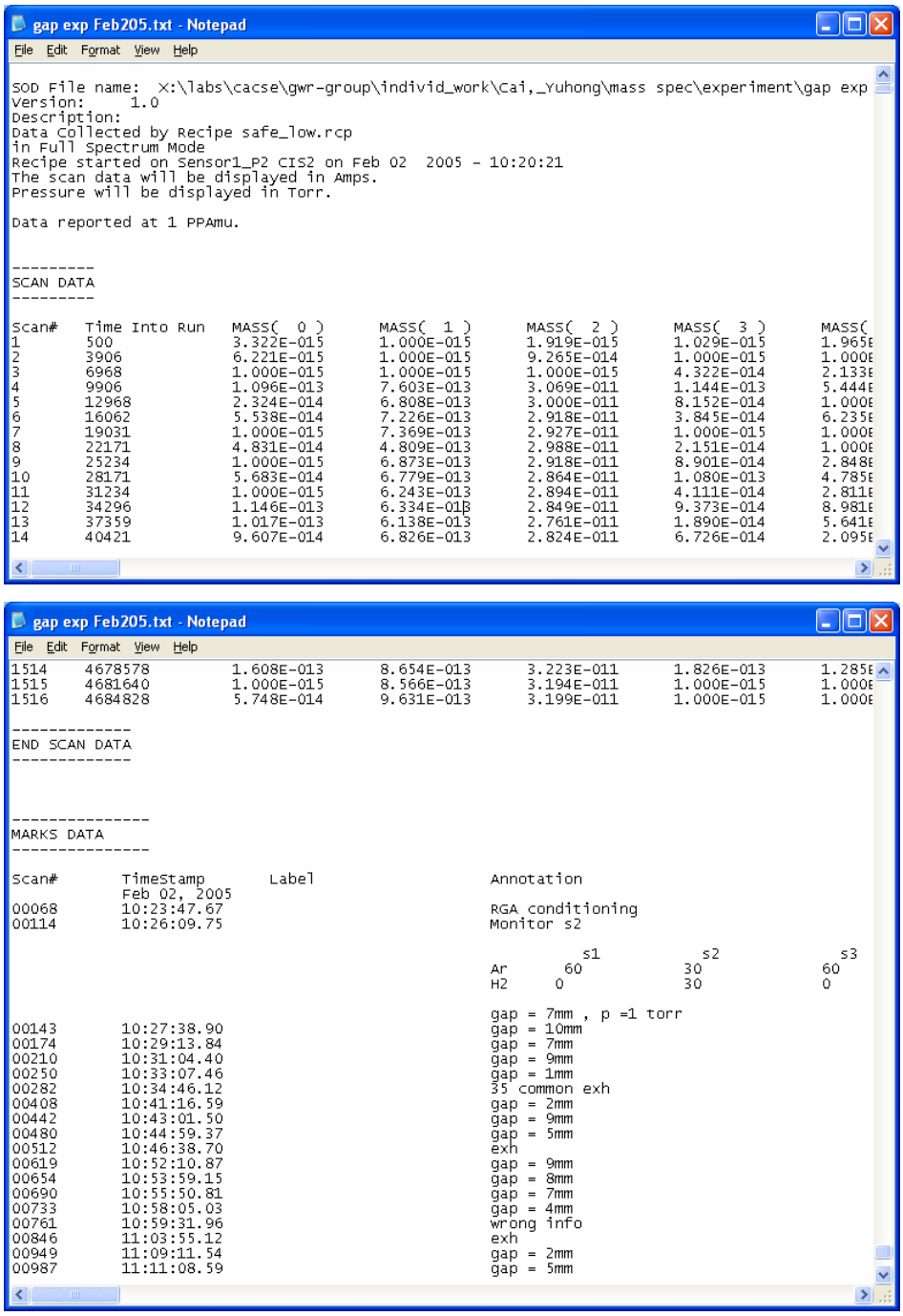

Figure 5.2: Snapshot of mass spectrometry data file

According to these annotations, one must choose the related data information manually from hundreds of data points and regroup them for further processing. For a batch production of daily deposition experiments, thousands of data items were generated during the process, which is labor and time consuming work.

- Different data formats: A desired solution to data storage is to group associated information together. For example, to study a deposition process, the related information includes: equipment settings, operation conditions, measurements during experiments, measurements after experiments, deposited wafer photo, etc. Data, text of remarks and images are mixed together and must be saved together.

- Data sharing and different applications: The research has been conducted with collaboration among different groups. Data coming from various sources is in dissimilar format and must be archived in an efficient way for sharing locally and remotely, i.e., one would like to easily retrieve data in the desired format and use preferred software such as Excel and MATLAB to process them.

Motivated the needs discussed above and the limitations of current data management, we developed a Java and XML-based integrated three-tier information system framework to facilitate data archiving, management, analysis and distribution for the CVD system design. The information system renders several user-friendly interfaces and can be easily integrated into other large data management systems. The advantages of the system include: (1) Reusable design in system architecture and

data storage; (2) Integration of data archiving and distributed simulation; (3) Storage and browsing of large volumes of heterogeneous data produced in semiconductor manufacturing processes (4) Data sharing among different group for dissimilar applications.

5.2 Literature Review

With the soaring of the Internet in recent years, people have begun to couple the promising information technology with the current manufacturing technologies to reduce the overall manufacturing costs. E-Manufacturing, e-Diagnostics and e-Business have become "buzzwords" in the semiconductor industry.

The "e" indicates information to be gathered, shared and manipulated through secured web-based networks for manufacturing, diagnosis and business. e-Manufacturing synchronizes the planning, procurement, and operations of a factory with its support functions at significantly faster speeds with greatly reduced costs, which is achieved through the efficient information flows among the factory, its suppliers, its customers and its internal support groups [12]. e-Diagnostics provides the field-service engineers the ability to access the IC maker's facility remotely via secured network and monitor, diagnose and configure the equipment rapidly or even remotely [100]. e-Business enables companies such as IC makers, equipment and material suppliers, business partners and customers to collaborate more closely [9].

Although some companies recently announced they have been developing huge semiconductor-specific commercial management software [4, 89, 96, 103], these de-

velopments primarily focus on data management for equipment productivity, supply plan, yield management and business-to-business data exchange, etc. Little attention has been paid to the information system development for the process data analysis and manipulation.

In the semiconductor industry, the manufacturers have a large amount data with various formats to maintain for manufacturing and business transactions. However, different data formats limit the data exchange among applications. To enable exchange of these data, an industry consortium group, the Pinnacles Group, designed an industry-specific Standard Generalized Markup Language (SGML). Their idea was that the data wrapped by industry-specific SGML would enable intelligent applications not only to display semiconductor data sheets as readable documents but also to drive ULSI design processes [14]. The problem of this approach is that SGML is so complicated that almost no software has ever implemented it completely. Programs that implemented or relied on different subsets of SGML were often incompatible with each other [45].

XML is a descendant of SGML, but with much less complexity of SGML [73]. With a simple, well-documented data format and rich data structures, it aims the interchange of metadata. XML documents are plain text and can be read and edited with any text-editor. The most common applications of XML today involve the storage and transmission of information among different software applications and systems. "XML is often referred to as the technology of the future" [76]. Many large database management systems offer XML interface to generate and take XML format data for further delivery.

Java is a high-level programming language, which is object-oriented, platform-neutral, distributed, robust, secure, multithreaded, easy to maintain, etc and has many library packages supporting Internet applications (Servlets, JSP, JavaBeans, etc.) [73, 23]. It is not only a fully functioning programming language just like C, C++, but also it can be regarded as an appropriate programming language for the World Wide Web [37].

XML is a perfect match to Java for web applications, because Java code is portable and XML makes the data portable. More details on these two technologies can be found from the many reference books widely available, which also demonstrate their popularity.

The three-tier system architecture has been applied widely to develop enterprise solutions in various industries including the semiconductor industry for years, because of its scalable, reusable and manageable properties [12, 100, 52, 98, 32]. While the solutions for different businesses may appear different, the essential framework structure is actually identical, i.e., the front-end layer and the backend layer are communicated through the middle layer which takes orders from the front layer, extracts inquired information from back layer server, wraps them in the desired format, and then presents them to the front layer. There are several different methods to implement a three-tier system architecture solution.

5.3 Information System Framework

As shown in Figure 5.3, the integrated information system developed is based on the three-tier infrastructure design and provides us with a scalable and flexible solution with a great security structure to share data locally and remotely.

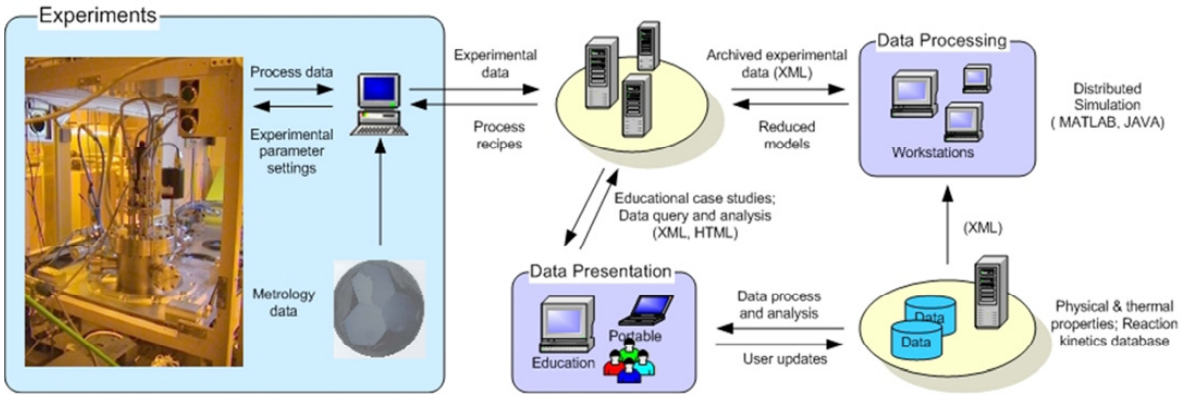

Figure 5.3: The Information System Framework for the Programmable CVD System

The backend tier is the data server in which processed experimental data that includes equipment information, experimental conditions and experimental results are imported and different types of data are wrapped in XML format for further delivery or applications. The middle tier takes orders from the front-end users, extracts the inquired information from data servers, wraps data in the desired formats for presentation, and then sends them back to the users. The front-end clients may query and display experimental information in HTML format by retrieving data from XML documents. For the analysis and simulation work on the client side, the data (such as experimental information, physical and thermal properties, chemical kinetics data) wrapped in XML format will be parsed by a parser written by users

and integrated into applications for computing and plotting in MATLAB. Simulation results can be compared to experimental data to validate process models, or experimental data can be used for parameter identification in the model, and the optimized simulation models and control parameters could be used as feedback at the experiment sites to improve experiment results. The information system exhibits these features:

- User friendliness: The potential users of the information system could be engineers, researchers, students or people from any areas. User-friendly intuitive interfaces make it much easier to navigate through a complex system. Web-browser interfaces would always be good choices if possible, which eliminate any complicated training.

- Flexibility: It can be easily deployed and run on different platforms or servers. For the web-based applications, compatibility with different web-browsers is also be assured.

- Scalability: What has to be done to the system if more or fewer clients are involved? A good system design always takes care of the scalability concern. Three-tier system architecture can manage and distribute workload among servers intelligently and scale up or down with little effort depending on targeted workload.

Data managed in this information system may include process data, experimental results data and other relevant data, such as species physical property data,

related reaction kinetics data or data from other archives. These data are likely to have heterogeneous formats. Process data may include real-time data collecting from deposition process; and experimental results may contain those data describing the image or properties of deposited films; the relevant data may come from various sources, such as literature, suppliers, or other fabrication process. The application of JAVA and XML technique to the information system makes data management, archiving and distribution successful and efficient.

5.3.1 Data Store and Archive

Because data type, locations or sources, amount, future growth patterns and operations applied to data may be different, different storage strategies have been developed. For example, Ignatius and Simas [52] proposed a distributed data store and management model to leave the data where they were and refer to them from the central location, because the size of data stored is large, the users do not want to give up their data, and there is no need to duplicate the data in the central server. This is one of the reasonable solutions for large scale manufacturing data management, especially when data sharing is across corporations or organizations. To a fabrication process, such as a CVD process, the centralized data store and management model may be more appropriate, which stores all data on the central data warehouse for quick access and easy management.

XML is a universal data storage format and it can be tailored to store and organize any kind of information of user interests [45, 94]. With its simple syntax

and hierarchical tree-like data structure described by a document model, document type definitions (DTDs), XML is easy to read and parse. It also separates data representation and presentation to allow different views of the same set of data. Combined with style-sheets, XML can create formatted documents in any style. For small data application, XML may be used as a self-contained data store at zero cost compared with commercial database management systems.

The strategy taken for developing the information system is to use XML to wrap and mark data from different sources and centralize them together in a specific location for distributed applications.

5.3.2 Data Access and Retrieval

The middle tier also called business-handling layer consists of two layers: the business logic layer and the server side presentation logic layer. The business logic layer interprets user's request to determine what data service is requested, how to invoke the requested data services and how that information should be extracted or generated from the data service module. As indicated by its name, the server side presentation logic layer handles the formatting and presentation of user-interested data.

The business-handling module, hosted on the web server, is implemented with Java Servlet and JSP (Java Server Pages) technology. Java Servlet is supported directly or by a plug-in virtually on every major web server. Java Servlet is easier to use, more efficient, powerful and portable, safer and cheaper than traditional

CGI and many alternative CGI-like technologies. Servlet also has an extensive infrastructure for automatically parsing and decoding HTML form data, reading and setting HTTP headers, handling cookies, tracking sessions, and many other high-level utilities [44].

All clients communicate with the web server through HTTP protocols to invoke the business logic layer's Servlets using a URL (Uniform Resource Locator), which in turn calls the data service module to request the corresponding services. Some of these Servlets are merely utility modules, which are exclusively used by other Servlets and not offer any service directly to users.

Well-defined XML documents are highly structured and easy to parse. There are two types of parser model: Document Object Model (DOM) and Simple API for XML (SAX). A DOM-based parser has following significant features: the entire document is parsed and stored in memory to create the hierarchical data structure, and it is a simple approach applicable to the repeated, random access to different sections of the document. The SAX-based parser is an event-driven model, where no tree structure is built and data is passed to the application when it is found, so less memory space is needed than DOM.

Since it is easier to traverse and edit an XML document with a DOM tree structure, a DOM-based parser has been developed in Java to extract the data wrapped in XML format for analysis and simulation purpose in the MATLAB environment on the client side. To retrieve data needed from the XML data files, user may download the parser and related Java classes, and then create an object of the parser class to obtain the data.

5.3.3 Data Presentation and Applications

XML separates data representation from its presentation and facilitates different views of the same set of data. An XSL (eXtensible Stylesheet Language) stylesheet that marks up the data with formatting objects can be used to transform an XML document into a variety of formats, such as PDF, Excel, etc. To present an XML document on the web in HTML format, generally a CSS (Cascading Style Sheets) stylesheet or an XSLT (XSL Transformations) stylesheet can be used. The alternative option of rendering XML data on the web is to instantiate a DOM object. Combing it with HTML and JavaScript, we can obtain the capabilities of generating Dynamic HTML (DHTML) on the client side [46].

The client-server architecture supports two application models: thin client model and thick client model. The thin client model characterizes all application software needed to generate a presentation report reside on the server side and is the recommended model for client-server network architecture [37, 52], while in the thick client model the server merely provides data to the clients and the clients run their own software for specific applications. The thin client model has many advantages for clients: no need to take care of software updates, no special training or computer knowledge is necessary, no expensive or large hard drives required, intention to "network computing instead of personal computing" [37]. Web-browsers are good examples of the thin client model. The thin client model would be a preferable structure from cost saving view point and for general applications. However, if clients have already acquired some complex analysis systems (such as MATLAB,

FLUENT, or FORTRAN) and also prefer running large computing or simulation applications in a more controllable way, the thick client model will be the desirable solution.

We utilize the full power of client-server architecture and apply both the thin client model and the thick client model to develop our proposed information system, i.e., the system will support user queries against data collected and generate reports on demand from the server as in the thin client model, and may also render the data for client-side applications as in the thick client model. The computation work may include simulator validation, parameter identification with experimental data, experiment optimization with simulation results, etc. This system also gives users more flexibility to manipulate experimental data. Different users could use data for different purposes, such as plotting and analyzing experimental results for case study or educational purposes.

5.4 Demonstration of the Framework Functionalities

The framework functionalities have been illustrated using the three-segment prototype design of programmable CVD system. A great number of experiments had been performed on prototype I and II systems. The implementation of the data management strategies are different for each.

For prototype I, all experimental results are put into one single XML data file because of the small amount of measurement data. A parser written in JAVA is used to parse and extract data from the data file to MATLAB simulation environment.

This parser can also be used for publishing data with web applications.

For prototype II, because of vastly greater amount of measurement data such as 4pp and mass spectrometry data associated with one single wafer, each experiment run is wrapped in one single XML file. With the new version MATLAB (7.0 and higher version) supporting to process XML file, a parser was written in MATLAB to handle data retrieval from XML data files.

The following sections illustrate the information system developed through the data management for the prototype I reactor system. The differences in data structure in XML between reactor type I and type II also is discussed.

5.4.1 Data Management for Prototype I CVD System

The prototype system I was constructed as a modification to an Ulvac ERA1000CVD cluster tool. A large amount of heterogeneous data generated from the experiments must be archived, including equipment information, experimental operating conditions measurement results and wafer images.

- Experimental Data Description and Archiving

  The experimental data file (PRdata.xml) is illustrated in Figure 5.4, viewed using Internet Explorer web browser. It shows how we create our own XML-based language and use these fabricated element names to encode experimental data. The experiment data file only stores raw data. The root element of this file is *ExperimentalData*, all other elements are the child nodes of this root, and each child element can have attributes which provide additional information

on this element.

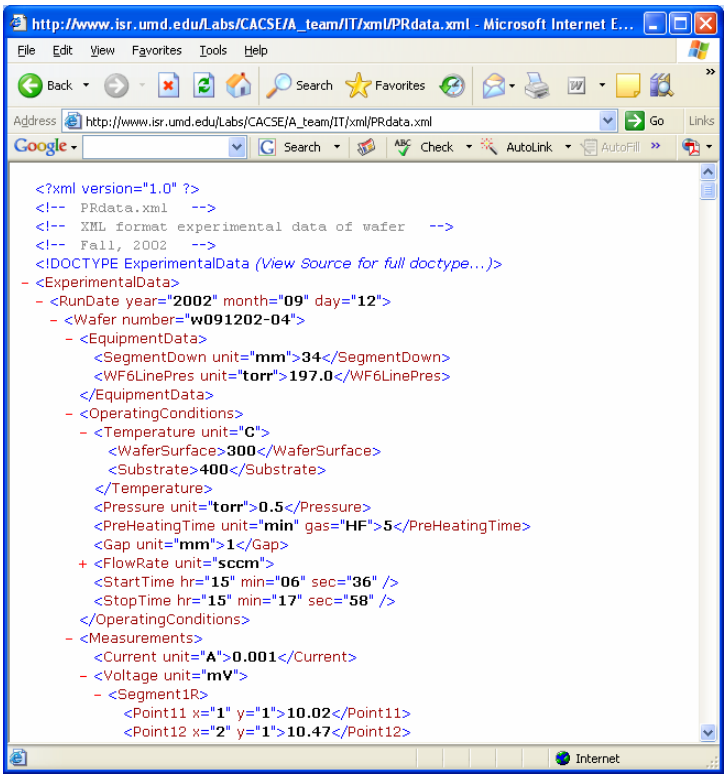

Figure 5.4: The XML data file viewed using Internet Explorer web browser

Figure 5.5 shows the hierarchical tree-like data structure of the experimental data XML file, which is defined in the experimental data Document Type Definition (DTD) document. The root element *ExperimentalData* contains child elements *RunDate*. The *RunDate* elements do not contain content, but have associated attributes *Year*, *Month* and *Day* to record the experiment's date. For any day, no limits are set for the number of wafers that can be processed. Wafers are sorted by their key ID - "Wafer number". The detailed information on each wafer is divided into three categories: the Equipment data (process diagnosis), the Operating conditions data (simulator input), and the

Measurements data (analysis, simulator validation).

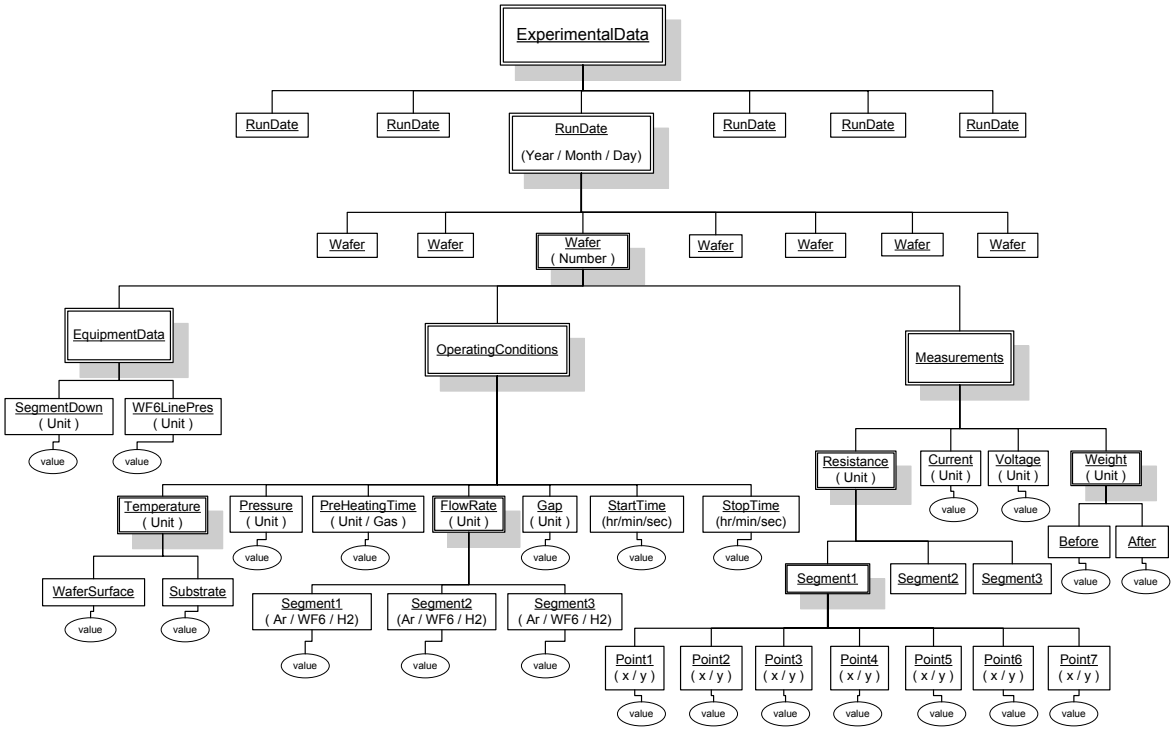

Figure 5.5: Hierarchical data structure of the XML data file for prototype I

The Equipment data include the showerhead position, wafer position, gas line gauge pressure, etc; the Operating conditions data include all information about experimental conditions, such as operating temperature, pressure, segment gas flow rates, wafer/segment spacing, deposition time, and so on; the Measurements data include experimental results, such as wafer mass before and after deposition, wafer image files, voltages and coordinates of selected points on the wafer surface for calculating film thickness. Generally, we use a descriptive word for the XML tag name to describe their data content. Most of these elements have *Unit* as their associated attribute to mark the data.

Some elements have more than one attribute, such as *hour*, *min*, *second*, *gas*, etc, while some does not have any attributes. All elements may have content and child elements. Their content usually is called the leaf of a tree. They may include the real data or null value.

- Parse and Retrieve Experimental Data

  The data flow from an XML file to a MATLAB simulation client is shown in the Figure 5.6. The parser consists of four Java classes: *EquipmentData*, *OperatingCond*, *Measurements* and *DOMPRdata*.

  The three utility classes, *EquipmentData*, *OperatingCond*, *Measurements*, define data structures to store extracted raw data and methods for post-processing. The class *EquipmentData* encapsulates equipment information on experiments. Its data members are defined to store equipment information and no member service is needed to process the raw data beyond queries. The class *OperatingCond* encapsulates information on experimental operating conditions, such as temperature, pressure, flow rate and so on. Its data members are used to store experimental operating conditions and member methods provide services for further processing the raw data, such as computing deposition time from start and stop time. The class *Measurements* encapsulates information on experimental and measurement results, such as wafer image file's name, coordinates of measurement points, wafer weight before and after deposition, etc. Its data member variables are declared to store experimental and measurement results and member functions are used to computer wafer mass gain

and convert voltage to thickness at measured points.

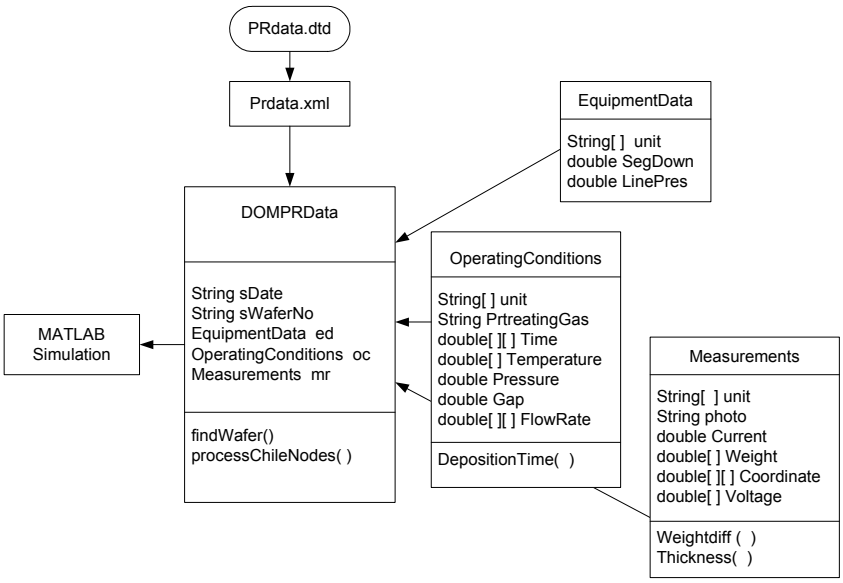

Figure 5.6: The data flow from an XML file to a MATLAB application

The class *DOMPRdata* is the parser drive class, which instantiates the three utility classes. This class has two constructors allowing users to specify a local or remote XML file for parsing. Both of the constructors take two parameters: 1) the name of the local or remote XML file (i.e., PRdata.xml) and 2) the wafer number, the key to accessing all experimental data associated with a particular wafer. Below is an example of the calling conventions in MATLAB to load PRdata.xml file from a local or remote location.

```
% load PRdata.xml locally
WJO = DOMPRdata('PRdata.xml', 'w091202-04');
% load PRdata.xml from a remote location
```

```
url=java.net.URL('http://www.isr.umd.edu/Labs/CACSE/...
A_team/IT/xml/PRdata.xml');
WJO = DOMPRdata(url, 'w100802-02');
```

The DOM-based XML parser first validates the user specified XML document (PRdata.xml). If it is a valid XML document, the parser will parse the entire document and create the hierarchical data structure. Then the parser traverses it to locate all experimental information of the wafer identified by the querying wafer number through the member function findWafer.

The *findWafer* method parses the XML document and compares the querying wafer number against all retrieved wafer numbers. If there is a match, the member function processNode is called to process all information related to the wafer. Then the member function *assignValue* is called to organize and store these information into the corresponding objects of these three utility classes. Otherwise, the message "wafer not found" is returned.

The *processNode* method takes one argument, wafer node, and outputs information (node's name and value) about the node and its child elements to a 2-dimensional array. This method uses a switch structure to determine the node type. If an *ELEMENT* node is matched, the element's attributes are output and then its child nodes are processed in *processChildNodes* member method which uses recursive method to retrieve a node's child nodes by calling *processNode* method. In the cases of *CDATA* or *TEXT* node are matched, the node's text content is output.

The parser we developed is very generic and well designed. If a user modifies the experimental XML file, such as deleting or adding data under any wafer nodes, there is no need to rewrite the parser drive class. Only a few minor changes in the corresponding utility classes are necessary, making it convenient for maintenance and customized usage. Users can download and put these JAVA classes under their working directory. By instantiating an object of the parser in their application programs developed with MATLAB or JAVA, all XML wrapped data would be ready to use.

- Experimental Results Query

  A web application of downloading or browsing data from the XML file is shown in Figure 5.7. An HTML file is generated with the data extracted from XML file of experimental data and formatted by XSLT. By using Java Servlet and JSP technology, and instantiating the parser we have written, an interactive Internet webpage could be created easily. Because authorization is required to deploy servlets in the university's server, as an alternative to creating parser object, we used JavaScript and XSLT to create a dynamic webpage.

- Demonstration of data applications

  Figure 5.7 exhibits different applications that use XML wrapped experimental data. A user can query wafer information according to its ID from the database (now it is an XML data file, PRdata.xml) and use that part of the data needed for simulation or analysis, for example, the operating conditions can be taken as simulator inputs. The simulation outputs will be compared

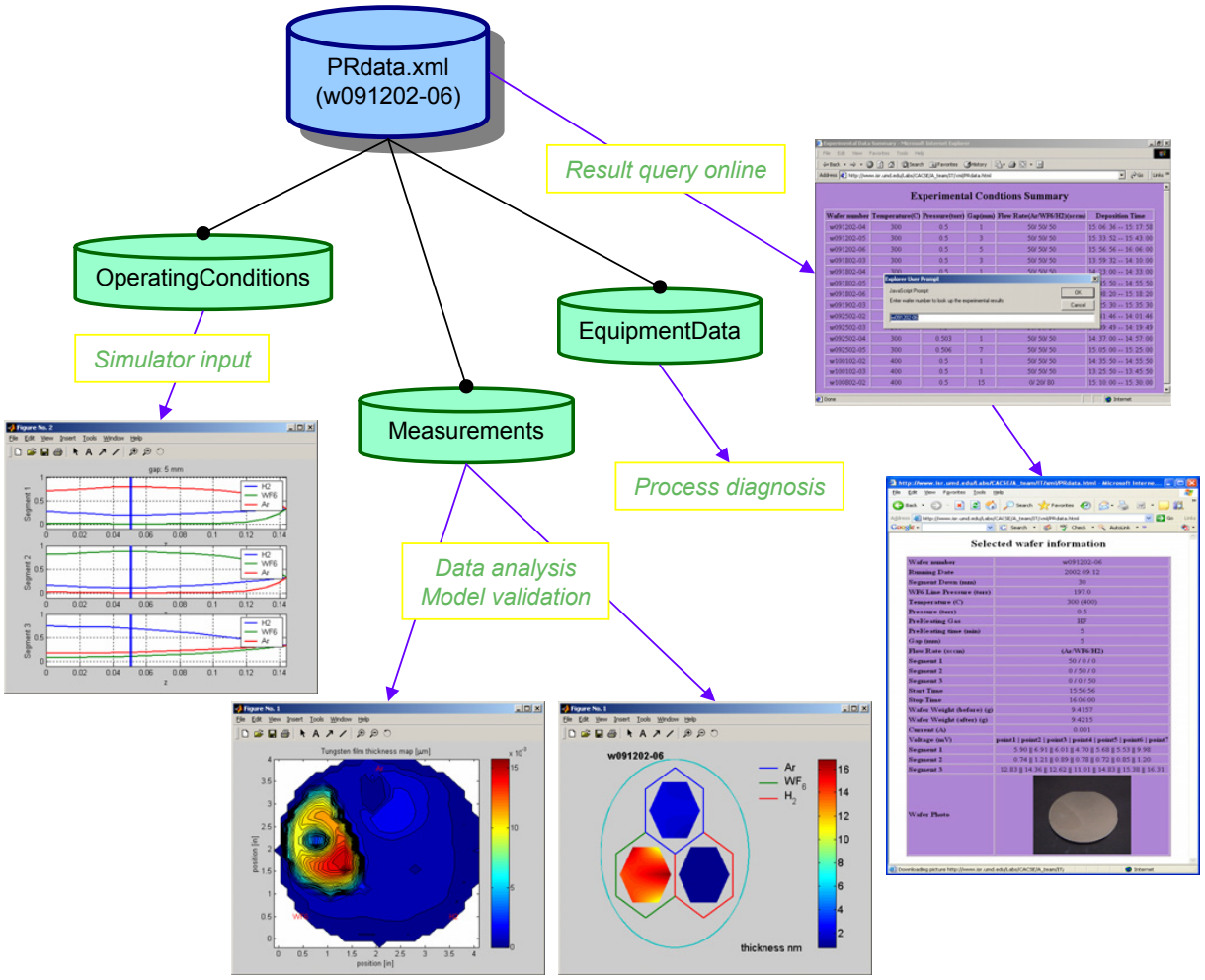

Figure 5.7: Applications of data retrieval from an XML data file

with experimental results saved as measurement data. The thickness measurements can be extracted and visualized in the MATLAB figure. For web applications, the wafer film image and corresponding experimental conditions can be presented in the web browser based on the user query.

5.4.2 Data Management for Prototype II CVD System

The prototype II CVD system includes modifications such as the addition of a gas distribution box, an advanced control system, and a mass spectrometry instrument for real-time gas composition measurements. Also, a four point probe method is employed to measure the deposited film thickness, generating a 30 by 30 matrix data format.

As mentioned earlier, for each deposition experiment, one data file of mass spectrometry includes hundreds of data points. Therefore, to include all experimental information in a single large XML file becomes infeasible and impractical, because each time when we query the database, we only need one item of wafer information. Moreover, parsing a large data file and storing all data temporarily in memory might cause the system to run out of memory or will slow down the program running speed. Therefore, a small single XML file is created to store all information connected to one single wafer deposition. The data structure for XML file is depicted in Figure 5.8.

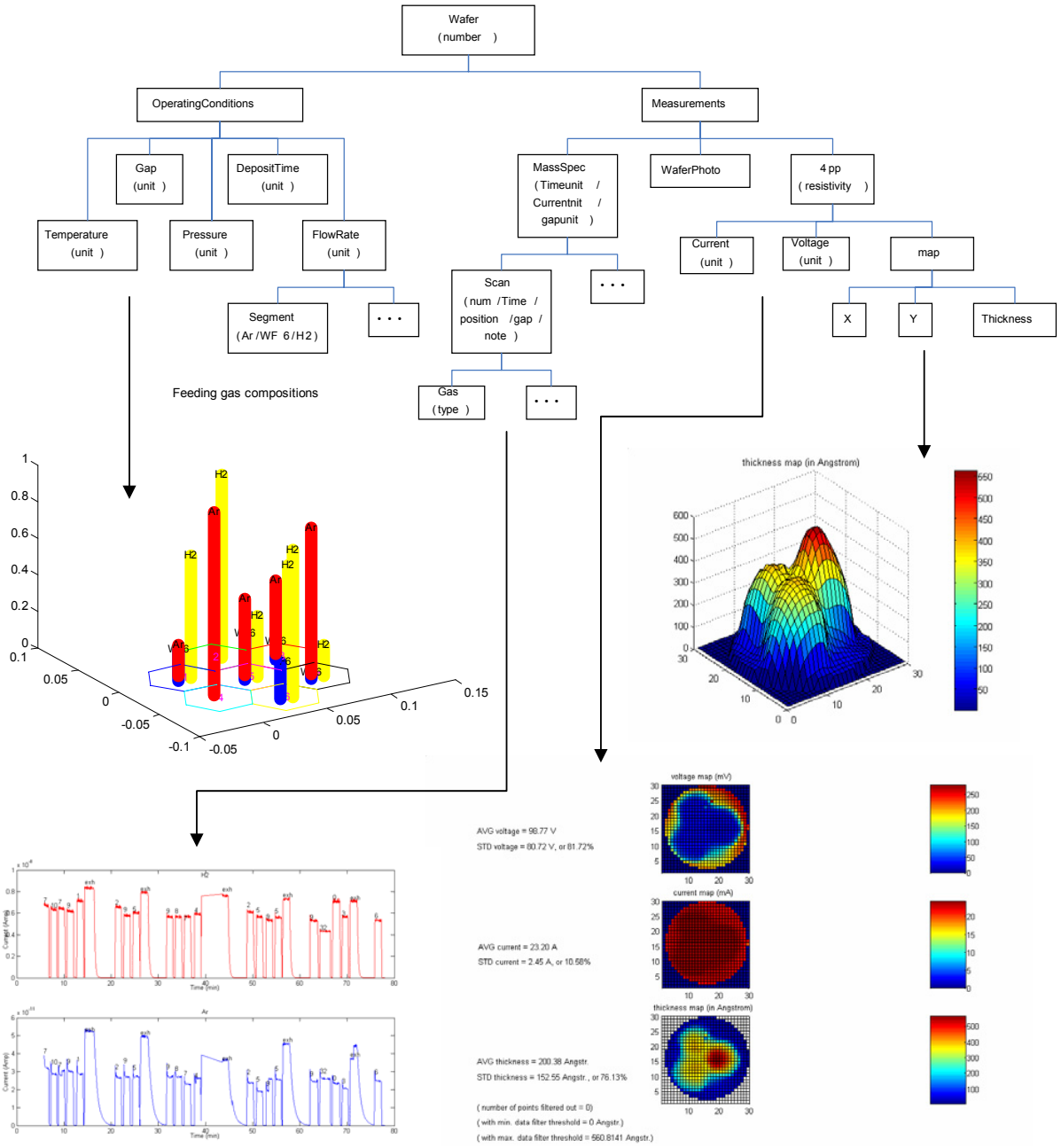

Figure 5.8: Hierarchical data structure of the XML data file for prototype II

Chapter 6

Conclusions and Perspectives

6.1 Concluding Remarks

Object-oriented design concepts and a modular approach were used to develop a flexible framework for chemical process simulation. Created in the context of semiconductor process simulation applications, the modular modeling approach was found to be effective in describing equipment hardware elements, and well as reaction mechanisms and simulator elements that did not necessarily correspond to physical equipment components. The ability to solve and test individual modules together with the ease with which modules can be combined, solved, analyzed, and swapped in and out was found to simplify simulator construction and debugging, as well as facilitate evaluation of model elements when the modeling equations or physical assumptions upon which models are based are under development in a simulation procedure. We find this modular framework facilitates an "evolutionary" approach to simulator development, starting with a simple process description and building model complexity and testing modeling hypotheses in a step-by-step manner.

In developing this simulation framework, the use of design patterns, which offers a high-level abstract structure to avoid dealing with the programming details in the early development phase, makes the design procedure more efficient, and helps us have a better understanding of object-oriented analysis and design. While

the framework currently is implemented using MATLAB, with the proposed design patterns, users can create their object-oriented modular system through object-oriented programming languages of their choice.

The simulation framework is capable of determining numerical solutions to model modules described by nonlinear algebraic equations, ordinary and partial differential equations, as well as any combination. Boundary-value problems are (semi)discretized with a built-in quadrature-based MWR procedure, and the framework interfaces to the full suite of MATLAB ODE/AE solvers. Current research focuses on improving the computational performance of the numerical algorithms, including the potential ability for parallel and distributed simulation, investigating the applicability of the framework for multiscale simulations, and making use of the flexibility of the simulation framework for the design and optimization of a range of advanced semiconductor device fabrication processes.

We have shown how simulation gave insight into the operation of the programmable CVD reactor and quantified the relative importance of different precursor transport and reaction mechanisms inside this reactor system. The modular-based approach offers great flexibility in rearranging the segment relative positions in the simulation, which facilitates the study of different reactor designs and validation of different models and reaction mechanisms. The effects of operating parameters and their susequent tuning can also be rapidly determined through the simulation.

Initial success has been achieved in developing an XML-based framework for on-line archiving and distributed simulation for the three-segment prototype I and II of programmable CVD system.

6.2 Future Work

To make the current computational framework more powerful and improve the ability to simulate programmable and other advanced CVD systems, the following are suggested as future reserach directions:

- The computational framework

  More sophisticated and complex modules could be added to expand system capabilities; for example, integrating more MATLAB built-in PDE, AE and ODE solvers into the class *Solver*. To solve for PDEs in the complex or irregular domains, the development of a spectral element method would be useful.

  Currently the Jacobian matrix is formed by using a finite difference method to compute every element in the matrix. Modifying the current approach with small matrix blocks generated for the entire sparse matrix, the computational efficiency is expected to get improved.

  Explore the application of the framework in these fields: multiscale modeling and simulation, distributed simulation and parallel simulation.

- The library of modular components

  Presently, the framework is mainly used for semiconductor applications. The development of reusable modular components such as showerhead, wafer, etc. will make the modeling work easier, shorten the development time and reduce cost of future solutions. For future usage, more general transport classes, such as energy module class, mass module class, etc. should be developed. For

specific applications, the user can write a simple subclass by inheriting some features from these base classes.

- The graphical user interface

  To introduce the simulation framework to more general audiences and have more groups to use it, a user-friendly graphical interface will make it easier for tutorial and learning. Like the popular commercial software, the interface can provide image icons for users to drag and drop components into the working space to assemble a modeling system, and click the selected solver to solve it.

- The programmable CVD system modeling and simulation

  Develop a two-dimensional model to describe gas composition changes within each segment. A thermal model for analyzing the temperature distribution on the wafer is needed. Detailed models to explain the gas mixture stated in the top exhaust zone and the gap region between the segment bottom and the wafer surface are preferred.

  To describe the complex deposition reactions precisely, and to predict different segment tungsten deposition rates and film thicknesses of the programmable CVD reactor, gaseous phase reactions will be considered and the surface reaction mechanism of tungsten deposition reaction will also be included into the modeling and simulation. The next generation design of a multi-segment programmable CVD system will benefit from the simulation results. It may also provide guidance for the programmable ALD (Atomic Layer Deposition) system design.

- The information system

  Currently the gas data used in the thermo-physical property estimator developed in Java is hard-coded. Each time when the data are updated or a new species is added, we must edit and recompile the source codes. To avoid touching the source code, we will store these data in an XML format file and offer a graphic user interface to facilitate property data editing and retrieval.

  With the evolving programmable CVD reactor design, more experimental data will be generated. Future work should consider a database design to appropriate to manage these individual XML data files. Also with the development of the modular simulator, reaction kinetics data need to be wrapped into the XML format file as well. Hand coding these data into XML file is a tedious, error-prone and time-consuming work. Automation tools with interactive graphic interfaces should be developed to facilitate various data encoding in the XML format.

  Experiment results and physical property data encoded in XML format make it easy to share and demonstrate the research results through Internet. Developing an online virtual CVD lab can help students learn the CVD process in a lively manner, e.g., run the simulation by retrieving data of experiments operating conditions, compare the predicted results with experiments, analyze and optimize the experimental parameters to see how these conditions affect the deposition performance.

Appendix A

Sample MATLAB codes

A.1 Definition of wafertherm class

A.1.1 wafertherm class: constructor method

```
function B = wafertherm(u)
emisw = 0.7;
emisq = 0.5;
emiss = 0.07;
a = 0.01;
sig = 5.670e-8;
Ta = 300;
Q = 5000;
Tw = 2;
Tq = 2;
var   = assocarray({'Tw' Tw});
param = assocarray({ 'emisw' emisw 'emisq' emisq 'emiss' emiss ...
                     'a' a 'sig' sig 'Ta' Ta 'Q' Q 'Tq' Tq ... });
A = naemodel('wafertherm',var,param);
B = struct([]);
B = class(B,'wafertherm',A);
```

A.1.2 wafertherm class: residual method

```
function A = residual(A)
unpack(A)
f1 = sig*emisw*emisq*Ta^4*(Tq^4-Tw^4) + ...
     sig*emisq*emiss*Ta^4*(1-Tw^4) + (1-a)*Q;
A = set(A,'resid',f1);
```

A.2 Definition of windowtherm class

A.2.1 windowtherm class: constructor method

```
function B = windowtherm(u)
emisw = 0.7;
emisq = 0.5;
a = 0.01;
sig = 5.670e-8;
Ta = 300;
h = 300;
Q = 5000;
Tw = 2;
Tq = 2;
var   = assocarray({'Tq' Tq});
param = assocarray({ 'emisw' emisw 'emisq' emisq 'a' a 'sig' sig ...
                     'Ta' Ta 'h' h 'Q' Q 'Tw' Tw });
A = naemodel('windowtherm',var,param);
B = struct([]);
B = class(B,'windowtherm',A);
```

A.2.2 windowtherm class: residual method

```
function A = residual(A)
unpack(A)
f2 = sig*emisw*emisq*Ta^4*(Tw^4-Tq^4) + h*Ta*(1-Tq) + a*Q;
A = set(A,'resid',f2);
```

A.3 Definition of wafer module class

A.3.1 wafer class: constructor method

```
function A = wafer(N)
Rw = 0.0508;                              % wafer radius, m
Rr = 0.22;                                % guard ring radius, m
delz = 0.004;                             % assembly thickness, m
rho = 2600;                               % assembly density, kg/m^3
sig = 5.677e-8;                           % Boltzmann constant, W/(m^2 K^4)
Ew = 0.7;                                 % wafer emissivity
Eq = 0.37;                                % guard ring (quartz) emissivity
a_w = 0.9;                                % wafer absorptivity
a_q = 0.1;                                % guard ring absorptivity
Ta = 300;                                 % ambient temperature, K
Ru = recipe([0 600 1200],[1 0.7 0]);  % u(t)
M = mwrmodel('cyln', N, 'r', [0 Rr]); % quadrature grid info
S = get(M,'R');                           % quadgrid object
Dr = get(M,'dr');                         % 1st-derivative operator
DDr = get(M,'ddr');                       % 2nd-derivative operator
wmask = mask(S,'lineseg',[0 Rw]); % scalarfield object to handle discontinuity
emiss = wmask*Ew + (1-wmask)*Eq;
alpha_w = wmask*a_w + (1-wmask)*a_q;
Ql = scalarfield(S,5000);                 % heat flux of heating lamp, W/m^2
Qg = scalarfield(S,0);                    % heat flux of gas, W/m^2
Qc = scalarfield(S,0);                    % heat flux of chamber, W/m^2
qc = scalarfield(S,0);                    % heat flux at r=Rr, W/m^2
T = scalarfield(S,Ta);                    % initial solution guess, K
var   = assocarray({'T' T});
param = assocarray({ 'Dr' Dr 'DDr' DDr 'S' S 'wmask' wmask 'delz' delz ...
        'sig' sig 'alpha_w' alpha_w 'emiss' emiss 'Ta' Ta 'Rw' Rw ...
        'rho' rho 'Ql' Ql 'Qg' Qg 'Qc' Qc 'qc' qc 'Ru' Ru });
B = naemodel('wafer',var,param);
C = odemodel('wafer',var,param);
A = class(struct([]),'wafer',B,C);
A = set(A,'dxdtcoeff',{[0 0]});
```

A.3.2 wafer class: residual method

```
function A = residual(A)
unpack(A)
kw = 269 - 0.585*T + 3.75e-4*T^2;   % wafer thermal conductivity, W/m/K
kg = 1.53 - 1.87e-3*T + 5.8e-6*T^2; % guard ring thermal conductivity, W/m/K
k = wmask*(0.1*kw + 0.9*kg) + (1-wmask)*kg; % assembly thermal conductivity
k = bpsf(k,1);                      % filter for discontinuity
Cw = 307 + 1.54*T - 1.06e-3*T^2;    % wafer heat capacity, J/kg/K
Cg = 330 + 1.77*T - 9.45e-4*T^2;    % guard ring heat capacity, J/kg/K
Cp = wmask*(0.1*Cw + 0.9*Cg) + (1-wmask)*Cg; % assembly heat capacity
dCpT = wmask*(0.1*(Cw - 307)+0.9*(Cg - 330)) + (1-wmask)*(Cg-330);
Cpt = Cp + dCpT;          % d(CpT)/dt
t = get(A,'currtime');    % current time, sec
u = powerfrac(Ru,t);      % heating lamp power
Rp = ( delz*( (Dr*k)*(Dr*T) + k*(DDr*T) ) - sig*emiss*T^4 + ...
    alpha_w*u*Ql + Qg + Qc ) / (delz*rho*Cpt);  % interior residual
Rp = setbval(Rp,Dr*T,'r','min');     % residual at r=0
Rp = setbval(Rp,k*(Dr*T)+sig*emiss*T^4-qc,'r','max'); % residual at r=Rr
A = set(A,'resid',Rp);
```

BIBLIOGRAPHY

[1] Adomaitis, R.A., Lin, Y.H., & Chang, H.Y. (2000). A computational framework for boundary-value problem based simulations. *Simulation, 74*(1), 28-38.

[2] Adomaitis, R.A. (2002). Objects for MWR. *Computers and Chemical Engineering, 26*(7-8), 981-998.

[3] Adomaitis, R.A. (2003). A reduced-basis discretization method for chemical vapor deposition reactor simulation. *Mathematical and Computer Modeling, 38*, 159-175.

[4] Agilent technologies debuts new graphics-based software solution to simplify parametric data management, extraction and analysis, July (2002). *http://www.agilent.com/about/newsroom/presrel/2002/17jul2002f.html*.

[5] Aling, H., Banerjee, S., Bangia, A.K., Cole, V., Ebert, J., Emami-Naeini, A., Jensen, K.F., Kevrekidis, I.G., & Shvartsman, S. (1997). Nonlinear model reduction for simulation and control of rapid thermal processing. *Proceedings of the American Control Conference, v 4*, 2233-2238.

[6] Badgwell, T.A., Edgar, T.F., & Trachtenberg, I. (1992). Modeling and scale-up of multiwafer LPCVD reactors. *AIChE Journal, 38*(6), 926-938.

[7] Badgwell, T.A., Breedijk, T., Bushman, S.G., Butler, S.W., ChatterJee, S., Edgar, T.F., Toprac, A.J., & Trachtenberg, I. (1995). Modeling and control of microelectronics materials processing. *Computers and Chemical Engineering, 19*(1), 1-41.

[8] Balakrishnan, K.S., & Edgar, T.F. (2000). Model-based control in rapid thermal processing. *Thin Solid Films, 365*, 322-333.

[9] Baliga, J. (2001). E-Business enters the semiconductor industry. *Semiconductor International*, March.

[10] Banerjee, S., Cole, J.V., & Jensen, K.F. (1998). Nonlinear model reduction strategies for rapid thermal processing systems. *IEEE Transactions on Semiconductor Manufacturing, 11*, 266-275.

[11] Banks, H.T., Beeler, S.C., Kepler, G.M., & Tran, H.T. (2002). Reduced order modeling and control of thin film growth in an HPCVD reactor. *SIAM Journal on Applied Mathematics, 62*(4), 1251-1280.

[12] Bloss, D., & Pillai, D. (2001). E-Manufacturing opportunities in semiconductor processing. *Semiconductor International*, July.

[13] Bode, C.A., Ko, B.S., & Edgar, T.F. (2004). Run-to-run control and performance monitoring of overlay in semiconductor manufacturing. *Control Engineering Practice, 12*, 893-900.

[14] Bosak, J. (1997). XML, Java, and the future of the Web. *http://www.ibiblio.org/pub/sun-info/standards/xml/why/xmlapps.htm*, March.

[15] Broadbent, E.K., & Ramiller, C.L. (1984). Selective low pressure chemical vapor deposition of tungsten. *Journal of the Electrochemical Society, 131*(6), 1427-1433.

[16] Butler, S.W., & Edgar, T.F. (1997). Case studies in equipment modeling and control in the microelectronics industry. *AIChE symposium series, 316*, 133-144.

[17] Braatz, R.D., Alkire, R.C., Seebauer, E., Rusli, E., Gunawan, R., Drews, T.O., Li, X., & He, Y. (2006). Perspectives on the design and control of multiscale systems. *Journal of Process Control, 16*(3), 193-204.

[18] Breiland, W.G., & Coltrin, M.E. (1990). Si deposition rates in a two-dimensional CVD reactor and comparisons with model calculations. *Journal of the Electrochemical Society, 137*(7), 2313-2319.

[19] Cai, X. (1998). An object-oriented model for developing parallel PDE software. *4*, Preprint at the Department of Informatics, University of Oslo.

[20] Cai, X., & Langtangen, H.P. (2002). Developing parallel object-oriented simulation codes in Diffpack. *WCCM V, Fifth World Congress on Computational Mechanics*. Mang, H.A., Rammerstorder, F.G., & Eberhardsteiner, J. eds., July 7-12, Vienna, Austria.

[21] Cai, Y. (2005). *Multiplexed Chemical Sensing and Thin Film Metrology in Programmable CVD Process*. Ph.D thesis, University of Maryland, College Park.

[22] Cale, T.S., Park, J.H., Gandy, T.H., Raupp, G.B., &Jain, M.K. (1993). Step coverage predictions using combined reactor scale and feature scale models for blanket tungsten LPCVD. *Chemical Engineering Communication, 119*, 197-220.

[23] , Campione, M., Walrath, K., & Huml, A. (2001). *The Java Tutorial*. 3rd Edition, Addison-Wesley.

[24] Chang, H.-Y., & Adomaitis, R.A. (1998). Model reduction for tungsten chemical vapor deposition. ISR Technical Report: TR 1998-28, University of Maryland, College Park, MD, USA.

[25] Chang, H.-Y., Adomaitis, R.A., Kidder Jr., J.N., & Rubloff, G.W. (2001). Influence of gas composition on wafer temperature in a tungsten chemical vapor deposition reactor: experimental measurements, model development, and parameter estimation. *Journal of Vacuum Science and Technology B, 19*, 230-238.

[26] Cho, S., Henn-Lecordier, L., Liu, Y., & Rubloff, G.W. (2004). In situ mass spectrometry in a 10 Torr W chemical vapor deposition process for film thickness metrology and real-time advanced process control. *Journal of Vacuum Science and Technology B: Microelectronics and Nanometer Structures, 22*(3), 880-887.

[27] Choo, J.O., Adomaitis, R.A., Rubloff, G.W., Henn-Lecordier, L, & Liu, Y. (2005). Simulation-Based design and experimental evaluation of a spatially controllable CVD reactor. *AIChE Journal, 51*(2), 572-584.

[28] Choo, J.O., Adomaitis, R.A., Henn-Lecordier, L., Cai, Y., & Rubloff, G.W. (2005). Development of a spatially controllable chemical vapor deposition reactor with combinatorial processing capabilities. *Review of Scientific Instruments, 76* (062217).

[29] Choo, J.O. (2004). *Development of Spatially Controllable Chemical Vapor Deposition System*. Ph.D thesis, University of Maryland, College Park.

[30] Cooper, J.W. (2000). *JAVA Design Patterns*. Addison-Wesley.

[31] Cota, B.A., & Sargent, R.G. (1992). A modification of the process interaction world view. *ACM Transactions on Modeling and Computer Simulation, 2*(2), 109-129.

[32] Danner, P. (2002). The importance of IT infrastructure. *Semiconductor International*, July.

[33] Dieterich, E.E., & Eigenberger, G. (1997). The ModuSim concept for modular modeling and simulation in chemical engineering. *Computers and Chemical Engineering, 21*, Suppl., s805-s809.

[34] E, W. and Engquist, B., Notices of the AMS, **50(9)**, 1062-1070 (2003).

[35] E, W., Engquist, B., Li, X., Ren, W., and Vanden-Eijnden, E., The Heterogenous Multiscale Method: A Review, *http://www.math.princeton.edu/multiscale/review.pdf*.

[36] E, W. and Engquist, B., Comm. Math. Sci., **1**, 87-133 (2003).

[37] Eberhart, A., & Fischer, S. (2002). *Java Tools, Using XML, EJB, CORBA, Servlets and SOAP*. John Wiley & Sons, Ltd.

[38] Edgar, T.F., Campbell, W.J., & Bode, C. (1999). Model-Based Control in Microelectronics Manufacturing. *Proceedings of the IEEE Conference on Decision and Control, 4*, 4185-4191.

[39] Eversteijn, F.C., Severin, P.J.W., van den Brekel, C.H.J., & Peek, H.L. (1970). A stagnant layer model for the epitaxial growth of silicon from silane in a horizontal reactor. *Journal of the Electrochemical Society, 117*(7), 925-931.

[40] Fagley, J.C., & Carnahan, B. (1990). The sequential-clustered method for dynamic chemical plant simulation. *Computers and Chemical Engineering, 14*, 161-177.

[41] Fotiadis, D.I., & Jensen, K.F. (1990). Symmetry breaking phenomena in vertical and horizontal CVD reactors. *Proceedings - The Electrochemical Society, 90*(12), 92-98.

[42] Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley.

[43] Ginkel, M., & Stelling, J. (2001). Modular modeling of cellular systems. *The Third Workshop on Software Platforms for Systems Biology*, California Institute of Technology, Pasadena, USA.

[44] Hall, M. (2002). *More Servlets and JavaServer Pages*. Prentice Hall PTR.

[45] Harold, E.R. & Means, W.S. (2002). *XML in a Nutshell*. O'REILLY.

[46] Heinle, N. & Pena, B. (2002). *Designing with JavaScript*. 2nd Edition, O'REILLY.

[47] Hess, D.W., Jensen, K.F., & Anderson, T.J. (1985). Chemical vapor deposition: a chemical engineering perspective. *Reviews in Chemical Engineering, 3*(2), 97-186.

[48] Hillestad, M., & Hertzberg, T. (1986). Dynamic simulation of chemical engineering systems by the sequential modular approach. *Computers and Chemical Engineering, 10*(4), 377-388.

[49] Hitchman, M.L., & Jensen, K.F. (1993). Chemical vapor deposition - An overview. *Chemical Vapor Deposition – Principles and Application*. Hitchman, M.L., & Jensen, K.F. eds., Academic Press, New York.

[50] Hutchison, H.P., Jackson, D.J., & Morton, W. (1986). The development of an equation-oriented flowsheet simulation and optimization package - I. The quasilin program. *Computers and Chemical Engineering, 10*(1), 19-29.

[51] Hutchison, H.P., Jackson, D.J., & Morton, W. (1986). The development of an equation-oriented flowsheet simulation and optimization package - II. Examples and results. *Computers and Chemical Engineering, 10*(1), 31-47.

[52] Ignatius, H.J., & Simas, T. (2001). The E-Manufacturing domino effect. *Semiconductor International*, July.

[53] Ingle, N.K., Theodoropoulos, C., Mountziaris, T.J., Wexler, R.M., & Smith, F.T.J. (1996). Reaction kinetics and transport phenomena underlying the low-pressure metalorganic chemical vapor deposition of GaAs. *Journal of Crystal Growth, 167*(3-4), 543-556.

[54] *International Technology Roadmap for Semiconductors, Modeling and Simulation*, 2003 Edition.

[55] Jenkinson, J.P., & Pollard, R. (1984). Thermal diffusion effects in chemical vapor deposition reactors. *Journal of the Electrochemical Society, 131*(12), 2911-2917.

[56] Jensen, K.F., & Graves, D.B. (1983). Modeling and analysis of low pressure CVD reactors. *Journal of the Electrochemical Society, 130*(9), 1950-1957.

[57] Jensen,K.F., Fotiadis, D.I., Moffat, H.K., Einset, E.O., Kremer, A.M., & Mckenna, D.R. (1987). Fluid mechanics of chemical vapor deposition. *ASME, 2*, 565-585.

[58] Jensen, K.F., Fotiadis, D.I., Mountziaris, T.J., Einset, E.O., & Kuech, T.F. (1991). Models of chemical kinetics and transport phenomena in chemical vapor deposition systems.*Proceedings - The Electrochemical Society, 91*(4), 142-160.

[59] Kepler, G.M., Tran, H.T., & Banks, H.T. (2000). Reduced order model compensator control of species transport in a CVD reactor. *Optimal Control Applications and Methods, 21*(4), 143-160.

[60] Kleijn, C.R. (1991). A mathematical model of the hydrodynamics and gas-phase reactions in silicon LPCVD in a single-wafer reactor. *Journal of the Electrochemical Society, 138*(7), 2190-2200.

[61] Kleijn, C.R., & Werner, C. (1993). Modeling of Chemical Vapor Deposition of Tungsten films. *Progress in Numerical Simulation for Microelectronics, Vol 2*, Birkhauser Verlag.

[62] Kleijn, C.R. (1995). Chapter 4: Chemical vapor deposition processes. *Computational Modeling in Semiconductor Processing*. M. Meyyappan, editor. Boston: Artech House.

[63] Kuijlaars, K.J., Kleijn, C.R., & van den Akker, H.E.A. (1995). A detailed model for low-pressure CVD of tungsten. *Thin Solid Films, 270*(1-2), 456-461.

[64] Langtangen, H.P. (1999). *Computational Partial Differential Equations - Numerical Methods and Diffpack Programming*. Springer-Verlag.

[65] Langtangen, H.P., & Munthe, O. (2001). Solving systems of partial differential equations using object-oriented programming techniques with coupled heat and fluid flow as example. *ACM Transactions on Mathematical Software, 27*(1), 1-26.

[66] Leboreiro, J., & Acevedo, J. (2004). Processes synthesis and design of distillation sequences using modular simulators: a genetic algorithm framework. *Computers and Chemical Engineering, 28*(8), 1223-1236.

[67] Lee, H.H. (1990). *Fundamentals of Microelectronics Processing*. McGraw-Hill, New York.

[68] Lee, K.J., & Yoon, E.S. (1994). The flexible modular approach in dynamic process simulation. *Computers and Chemical Engineering, 18*, suppl., s761-s765.

[69] Li, X., Shao, Z.J., & Qian, J.X. (2004). Module-oriented automatic differentiation in chemical process systems optimization, *Computers and Chemical Engineering, 28*(9), 1551-1561.

[70] Lin, Y.H., Chang, H.Y., & Adomaitis, R.A. (1999). MWRtools: a library for weighted residual method calculations. *Computers and Chemical Engineering, 23*(8), 1041-1061.

[71] Lu, G., & Kaxiras, E. (2004). An overview of multiscale simulations of materials. *http://arxiv.org/PS_cache/cond-mat/pdf/0401/0401073.pdf*. arXiv:cond-mat/0401073, v1, 7 Jan.

[72] Maroudas, D. (2000). Multiscale modeling of hard materials: Challenges and opportunities for chemical engineering. *AIChE Journal, 46*(5), 878-882.

[73] Maruyama, H., Tamura, K., & Uramoto, N. (1999). *XML and Java, Developing web applications*. Addison-Wesley.

[74] McConica, C.M., & Krishnamani, K. (1986). *Journal of the Electrochemical Society, 133*(12), 2542-2548.

[75] McInerney, E.J., Srinivasan, E., Smith, D.C., & Ramanath, G. (2000). Kinetic rate expression for tungsten chemical vapor deposition in different WF6 flow regimes from step coverage measurements. *Z. Metallkd., 91* (7), 573-580.

[76] McLaughlin, B. (2000). *Java and XML*. O'Reilly.

[77] McLaughlin, K.J., Edgar, T.F., & Trachtenberg, I. (1991). Real-time monitoring and control in plasma etching. *IEEE Control Systems Magazine, 11*(3), 3-10.

[78] Meyer, B. (1988). *Object Oriented Software Construction*. Prentice Hall.

[79] Michaelidis, M., & Pollard, R. (1984). Analysis of chemical vapor deposition of boron. *Journal of the Electrochemical Society, 131*(4), 860-868.

[80] Moffat, H.K., & Jensen, K.F. (1988). Three-dimensional flow effects in silicon CVD in Horizontal. *Journal of the Electrochemical Society, 135*(2), 459-471.

[81] Oosterlaken, T.G.M., Leusink, G.J., Janssen, G.C.A.M., & Radelaar, S. (1996). The hydrogen reduction of WF6: A kinetic study based on in situ partial

pressure measurements. *Journal of the Electrochemcial Society, 143*(5), 1668-1675.

[82] Patel, N., & Niemyski, P. (2004). Model-based process control for 300 mm manufacturing: Part I - Lot-level control. *Future Fab Intl., 16*.

[83] Perkins, J.D. (1983). Equation-oriented flowsheeting. *Proceedings of the Second International Conference on Foundations of Computer-Aided Process Design*. Westerberg, A.W., & Chien, H. H. eds., CACHE, Austin, 309-367.

[84] Peskin, A.P., & Hardin, G.R. (1996). An object-oriented approach to general purpose fluid dynamics software. *Computers and Chemical Engineering, 20*(8), 1043-1058.

[85] Pidd, M., & Castro, R.B. (1998). Hierarchical modular modeling in discrete simulation. *Winter Simulation Conference Proceedings, 1*, 383-389.

[86] Plummer, J.D., Deal, M.D., & Griffin, P.B. (2000). *Silicon VLSI Technology: Fundamentals, Practice and Modeling*. Prentice Hall, Inc.

[87] Ponton, J.W. (1983). Dynamic process simulation using flowsheet structure. *Computers and Chemical Engineering, 7*(1), 13-17.

[88] PressMan, R.S. (1997). *Software Engineering: A Practitioner's Approach*. 4th edition. The McGraw-Hill Companies, Inc.

[89] Princeton Softech and Brooks-PRI enhance application performance and availability of leading factory automation application. October (2002). *http://www.princetonsoftech.com/news/press/brooks.htm*.

[90] Quirk, M., & Serda, J. (2001). *Semiconductor Manufacturing Technology*. Prentice-Hall, Inc.

[91] Raimondeau, S., & Vlachos, D.G. (2002). Recent developments on multiscale, hierarchical modeling of chemical reactors. *Chemical Engineering Journal, 90*(1-2), 3-23.

[92] Ramirez, F.P., Acosta, R.P., & Rivera, W. (2001). An object-oriented framework for parallel incompressible flow simulations. *http://mayaweb.upr.clu.edu/crc/crc2001/papers/freddy-perez/pdf*

[93] Ramirez, F.P., & Rivera, W. (2003). An object-oriented framework for computational fluid dynamics simulations. *http://mayaweb.upr.clu.edu/crc/crc2003/papers/FreddyPerez.pdf*

[94] Ray, E.T. (2001). *Learning XML*. O'REILLY.

[95] Roenigk, K.F., & Jensen, K.F. (1987). Low pressure CVD of silicon nitride. *Journal of the Electrochemical Society, 134*(7), 1777-1784.

[96] SAS, Brooks to deliver semiconductor industry's first yield execution solution. April (2002). *http://investor.brooks.com/ReleaseDetail.cfm?ReleaseID=77250*.

[97] Shacham, M., Macchieto, S., Stutzman, L.F., & Babcock, P. (1982). Equation oriented approach to process flowsheeting. *Computers and Chemical Engineering, 6*(2), 79-95.

[98] Shade, B. (2001). Increase productivity through E-Manufacturing. *Semiconductor International*, July.

[99] Shalloway, A., & Trott, J.R. (2002). *Design Patterns Explained*. Addison-Wesley.

[100] Singer, P. (2001). E-Diagnostics: Monitoring tool performance. *Semiconductor International*, March.

[101] Stadtherr, M.A., & Vegeais, J.A. (1985). Recent progress in equation-based process flowsheeting. *Proceedings of the Summer Computer Simulation Conference*, 325-330.

[102] Stuber, J.D., Trachtenberg, I., & Edgar, T.F. (1994). Model-based control of rapid thermal processes. *Proceedings of the IEEE Conference on Decision and Control, 1*, 79-85.

[103] SYNTRICITY Introduces dataConductorEP first web-native extensible platform for semiconductor yield improvement. *http://www.syntricity.com/news/PressReleases/071702.htm*

[104] Takahashi, R., Koga, Y., & Sugawara, K. (1972). Gas flow pattern and mass transfer analysis in a horizontal flow reactor for chemical vapor deposition. *Journal of the Electrochemical Society, 119*(10), 1406-1412.

[105] Theodoropoulou, A., Adomaitis, R.A., & Zafiriou, E. (1998). Model reduction for optimization of rapid thermal chemical vapor deposition systems. *IEEE Transactions on Semiconductor Manufacturing, 11*(1), 85-98.

[106] Toprac, A.J., Edgar, T.F., & Trachtenberg, I. (1993). Modeling of gas-phase chemistry in the chemical vapor deposition of polysilicon in a cold wall system. *Journal of the Electrochemical Society, 140*(6), 1809-1813.

[107] Wahl, G. (1977). Hydrodynamic description of CVD processes. *Thin Solid Films, 40*, 13-26.

[108] Xu, Y.H. (2001). *Real-time in-situ chemical sensing, sensor-based film thickness metrology, and process control in W CVD process*. Ph.D. thesis, University of Maryland. College Park, MD, USA.

[109] Zeigler, B.P. (1984). *Multifaceted Modeling and Discrete Event Simulation*. Academic Press.

[110] Zeigler, B.P. (1990). *Object-oriented Simulation with Hierarchical Modular Models: Intelligent Agents and Endomorphic Systems*. Academic Press.