Rigorous Integration of Semiconductor Process and Device Simulators

Thomas Binder, Andreas Hössinger, and Siegfried Selberherr, *Fellow, IEEE*

***Abstract*—We deal with problems arising in the coupling of process and device simulators. It is analyzed what kind of data and algorithms such simulations are based on. An overview of existing technology computer-aided design data models is given. A generic *object-oriented* data model suitable for three-dimensional process and device simulations, the so-called WAFER-STATE-SERVER is presented. By taking advantage of *object-oriented* abstraction mechanisms, key tasks in the coupling of simulators are relocated from the simulator into the WAFER-STATE-SERVER. The new data model allows an efficient data exchange between existing process and device simulators. It is capable of managing geometries of different dimensions, and handling grids and distributed quantities stored therein. The data model also defines algorithms to perform geometrical operations as they occur in topography simulations. Three process simulators based on the new data model were developed, one of which is presented in this work.**

***Index Terms*—Microelectronics, process simulation, semiconductors, wafer state.**

I. Introduction

TECHNOLOGY computer-aided design (TCAD) simulation programs are well accepted in the semiconductor industry. They present an invaluable help in improving existing technologies and can drastically reduce development time for new emerging technologies and downscales. Many tools are available today, some of those come from universities, others are developed and distributed commercially. A tool is usually focused on a particular process step. To simulate a whole semiconductor fabrication process flow a rigorous coupling of individual simulators is inevitable. Ideally, it should be possible to choose among all of the tools available on the market and use them as needed. In practice, however, the necessary tool integration is at best possible among the tools of one vendor. A simple coupling of simulators is very complicated due to a missing standardized data model.

We present a new *object-oriented* data model for the TCAD field. This data model, the so-called WAFER-STATE-SERVER, gives a unification of what data is common to all tools. The data model aims at the mentioned integration of process and device simulators. The data model is realized as a C++ class library and deals with several aspects of TCAD simulations. These aspects include *I/O* operations, *meshing*, and *algorithms* like the extraction of interfaces between two simulation domains. By taking advantage of the *object-oriented* programming paradigm, key tasks in the coupling of TCAD simulators can be handled directly in the library (as opposed to being handled in the simulator). One advantage of the new approach is that the tool developer is partly relieved from the integration task,[1] which means that more emphasis can be put on the physical problem to solve. The usage of well-defined interfaces brings another advantage, namely, the possibility to easily exchange algorithms without breaking the application code (the simulator).

The final target of the tool integration aims at simulating a whole flow of a semiconductor fabrication process where tools are exchangeable. It should be possible to exchange a simulator performing a specific step with any other suitable simulator that is available.

Before the new data model is developed, we will take a glimpse at existing solutions. Many solutions known to the author that are used in commercial or university TCAD suites more or less lack a clean, *object-oriented* definition of a data model from the point-of-view of the TCAD tool developer. Instead, the data model is reduced to the file level, where a file format and a set of data access libraries are used to read and write data.

A. Viennese Integrated System for TCAD Applications (VISTA)

VISTA is a framework that was designed to meet all needs of a TCAD user. VISTA introduced three levels of abstraction.

- Task Level: It provides an extension language (TCAD shell) that allows for solving high-level engineering tasks (e.g., optimization).
- Tool Level: It provides the tools to carry out the simulations and also the necessary integration of that tools.
- Data Level: It takes care of the representations of the data that are needed in TCAD simulations and copes with the problem of data exchange.

The data level of the VISTA system is realized as the profile interchange format (PIF) file format and a set of supporting libraries [1], [2]. It is intended as a database for TCAD simulation data. The PIF implementation provides an ASCII and binary representation of the data and a conversion tool to convert data between the two representations. The ASCII representation (*intersite representation*) is used to transfer PIF files between different architectures,[2] and is based on a proposal by Duvall [3]. The binary representation provides fast access to the data and was intended to exchange data between tools (*intertool representation*).

Manuscript received July 4, 2002; revised December 10, 2002. This paper was recommended by Associate Editor W. Schoenmaker.

The authors are with the Institute for Microelectronics, Technical University of Vienna, A-1040 Vienna, Austria (e-mail: binder@iue.tuwien.ac.at).

Digital Object Identifier 10.1109/TCAD.2003.816219

[1] The file format for an input file only needs to be selected, not implemented.

[2] This is to handle different byte-ordering schemes and the differences between 32- and 64-bit CPUs.

PIF is designed in a strict bottom-up fashion. The PIF application layer (PAL) does not have any semantical constraints on the data; only the pure syntax of the file is mapped. There is no definition of the status of a WAFER before and after a simulation run. Instead, the interpretation of the data was delegated to high-level libraries or directly to the simulation tools. As a consequence, several incompatible interpretations were evolving over the time the PAL was integrated with various tools. Due to the bottom-up design, semantic constraints on the data (the PIF cookbook) were introduced at a time where tools were already fully integrated with the PAL and could not be changed easily. Moreover, a great percentage of the developed functions are not used in any of the libraries and tools that are in existence today.

B. DF-ISE

DF-ISE [4] is the file format of the TCAD suite of the ISE software company [5]. As with PIF, a binary and ASCII representation is available although the binary representation is only capable of storing floating point and integer values. Not all of the data are stored in a single file; instead DF-ISE distinguishes several different file types. These file types are

- *layout* geometry for patterning operations;
- *cell* structures used for three-dimensional (3-D) solid modeling;
- *recursive-tensor* contains tensor product grids;
- *boundary* contains a boundary description of the simulation domain;
- *grid* holds an unstructured grid of the simulation domain;
- *dataset* this file type holds optional quantities that are stored on a grid or boundary. A dataset file is only useful in conjunction with a grid or a boundary file. Only scalar and vector values are supported.
- *property* holds a material database that is used by all ISE tools.

Depending on the application at hand, only a subset of the data is retrieved from file. Data in a DF-ISE file are organized in a sequence of blocks which in turn can contain other blocks. A restriction of the DF-ISE file format is that a grid file may contain only one global grid that comprises the whole simulation domain. Interfaces between different simulation regions (e.g., semiconductor ↔ oxide) must be boundary-conforming.

The starting geometry and the photo lithography data (masks) are stored in a layout file. This file is required as input for the process simulator (DIOS).

C. TMA Interchange Format (TIF)

TIF is the file format of parts of the Synopsys TCAD suite. The name of the file format comes from the name of the company (TMA) that originally offered the tools. The products of TMA, however, were first incorporated by Avant!, and later by Synopsys [6]. The two-dimensional (2-D) process simulator TSUPREM-IV[7] and the 2-D device simulator MEDICI [8] are based on this file format. Only an ASCII representation of TIF exists, which results in large files and, thus, in slow data access if complex WAFER structures are to be stored. Although the two mentioned simulators in principle use the same syntax to store the data on a TIF file, those files are not semantically compatible. Instead, TSUPREM-IV supports an option to write a MEDICI-compliant TIF file.

The successor of the 2-D TSUPREM-IV and MEDICI simulators is the 1-, 2-, and 3-D TAURUS [9] TCAD suite. TAURUS is comprised of a set of individual tools. Among others, these tools include TAURUS-Device and TAURUS-Process. The TAURUS-Process tool supports the creation of input descriptions suitable for TAURUS-Device.

D. Silvaco

The Silvaco [10] TCAD suite consists of the process-simulation module ATHENA [11] and the device-simulation module ATLAS [12]. ATHENA is limited to the simulation of 2-D processes. Data exchange is also performed via a file.

E. Semiconductor Wafer Representation (SWR)

The SWR [13] is based on an *object-oriented* client server architecture that consists of a set of C++ class definitions. Conceptually, the interface of the SWR is split into a field and a geometry server. The geometry server describes the boundary information of the WAFER and is used to represent and manipulate material regions. It supports three basic operations:

- the creation of geometric regions from inputs like a list of points or a mesh;
- it provides topological functionality like querying the list of edges incident to a point;
- it defines Boolean operations on geometries to support topography simulations.

The field server defines data structures to handle grid-related issues and to represent and manipulate distributed quantities (e.g., doping profiles, stress, etc.) stored on grids. Apart from simple mesh construction mechanisms, the field server provides interpolation functionality. The SWR 1.0 specification of the field server is limited to supporting 1- and 2-D meshes.

In the prototype implementation of SWR 1.0, the geometry and field servers are realized as separate processes. Clients connect to a server via the sockets interface and exchange data via shared memory. File converters are used to integrate existing simulators (e.g., SUPREM-IV [14]). A notable drawback of the SWR specification is that the definition of a general WAFER-STATE is missing.

To overcome the limitation to two spatial dimensions, a 3-D implementation of an SWR field server was presented in [15]. Apart from the supported dimension, this implementation differs from the prototype in that it allows one to choose whether the servers run as separate processes or are linked to the client.[3] The author, however, does not know of any tools that are integrated with this implementation (nor are any applications referenced in the paper).

F. Forest

From the author's point of view, the work by Sahul *et al.* [16] presents the most advanced approach to integrating process simulation tools in two spatial dimensions. It defines three basic models:

[3] This means that clients run in the same process space as the servers.

- *geometry server*: This module is most notably capable of persistently storing geometries on a file, moving the surface of a geometry, and repairing geometries that became invalid during a simulation.
- *grid server*: This module is capable of generating a grid on a given geometry. It defines methods to adaptively refine a grid, move a grid, and repair corrupted grids.
- *surface mesh server*: This module is capable of generating a surface mesh of the geometry, which is used for moving boundary problems.

Interaction between the three servers is necessary if a certain process step introduces an inconsistency. For example, for an oxidation simulation, the geometry has to be updated after the grid was moved by the grid server. If the geometry server detects inconsistencies in the geometry, these are repaired, a new grid is computed, and the grid server is updated in turn.

Two tools that were integrated with Forest are a restructured version of the SUPREM-IV process simulator, and the program SPEEDIE [17] that is used for etching and deposition simulations.

Although the interface of Forest is dimension independent, an implementation for three spatial dimensions is not available.

II. Data Modeling Aspects

If we look at the needs from a TCAD user's perspective, then there are several major issues to take into account. According to [18], three keys to a successful operation of TCAD are

- *prediction*: TCAD users expect the tools to predict how a yet-to-be-developed technology will perform.
- *fast calibration*: A calibration must be finished before a certain process changes and whenever a new version of the software is released.
- *importance of the grid*: Process and device simulations are very sensitive to the underlying simulation grid. Errors that can lead to a wrong result might be introduced if no attention, or not enough, is paid to the grid.

We would like to add another point to the above list of issues, namely, the importance of tool interoperability. There has been a strong user-driven urge to support file wrappers that convert data between the file formats of different vendors. Users would like to compare the results of different simulators and also to freely interchange tools in a process flow simulation.

To date, there is no accepted standard regarding the data crucial to carry out coupled TCAD simulations. As a matter of fact, existing solutions are based on a file format instead of a data format. The difference between file and data formats lies in the algorithms that are associated with data structures. Per definition, a data structure is defined via its representation (data in memory, data on disk) and a set of operations that are valid on that representation. On the other hand side, a file is only a means to persistently store data on a computer, thus, it is just one representation of the data.

To make things worse, the existing file formats are usually not even semantically compatible with each other. As some of the file formats are optimized for a certain purpose like, e.g., device simulation, they are not intended to pass data among process simulators. As a consequence, there does not exist a rigorous solution to couple existing tools in a way such that one particular process simulation step could be exchanged by another. Thus, if interoperability of such tools is desired, the only solution for the user is to use data wrapping tools to achieve at least a partial compatibility on the file level.

The above sketched issues lead to the following requirements that must be met by a TCAD data model.

A. Requirements for a TCAD Data Model

The four major aspects that need to be dealt with in a TCAD simulation from the tool's perspective are

- Persistent storage of simulation data. The tool must store the results of a simulation for later reference (visualization, input for another simulation) and the user does not want to bother with file format specifics.
- Handling of gridding steps. The tool needs a standardized way to interact with different gridding tools. It must be possible to switch to another gridding algorithm with the least possible effort.
- The user needs support to extract topological information, to manipulate the underlying geometry after a topography step.
- All data structures and algorithms offered by the data model should be available in all three geometrical dimensions.

The four requirements above mean that the identified algorithms are used exclusively via interfaces rather than being tightly integrated with a simulator. It is delegated to a library (the WAFER-STATE-SERVER) to provide one (or more) implementations for a certain interface. This separation of interface and implementation makes it possible to seamlessly use other algorithms from the simulator. The advantage here is clearly that the simulator(s) need not be changed as long as they adhere to the defined interfaces. A certain implementation of an algorithm like, e.g., the file format in which the input file is stored, can be selected at runtime (e.g., by passing an argument on the command line).

The observed semantical incompatibilities in existing solutions are often due to oversimplifications. On the one hand, such simplifications can help in managing the data for one kind of simulator, but complicate the generation of the data itself on the other hand side. One such simplification is to enforce boundary-conforming grids on the data stored on a file as it is the case with the DF-ISE approach. Another (design-related) simplification is the lack of so-called WAFER-STATE semantics. A WAFER-STATE defines a state between any two simulation runs that are carried out in the simulation of a process flow. It is this definition that is most important to a tool, and yet, no standard has evolved so far.

In the following, we will focus on the data structures and the API that were developed to support the tool developer with the desired operations. All of the above-defined requirements are met and reflected in the API. The API and data structures are referred to under the term WAFER-STATE-SERVER. Note that in our application, the term "SERVER" does not denote a traditional client/server architecture as it is used in, e.g., Web-based applications, but is rather meant to denote the set of classes that comprise and enforce the WAFER-STATE definition.

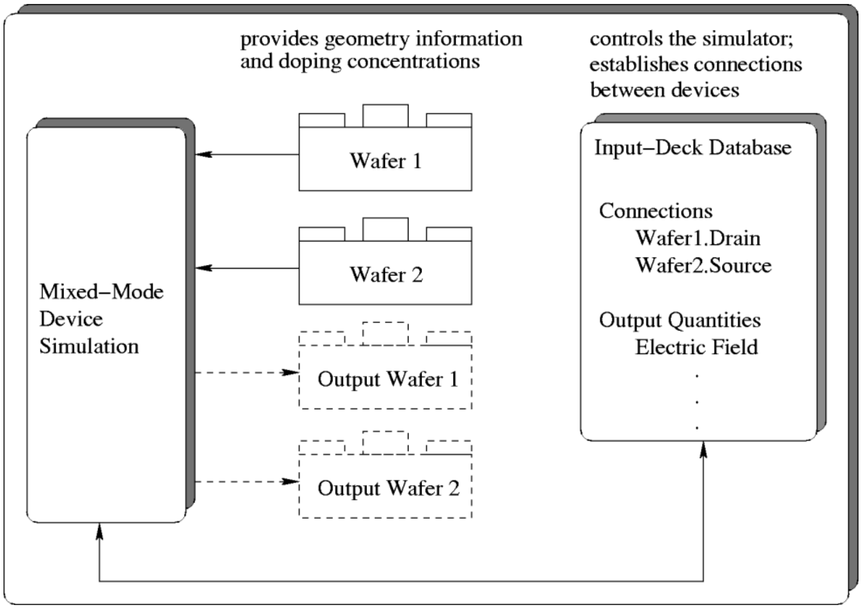
provides geometry information and doping concentrations
controls the simulator; establishes connections between devices
Mixed–Mode Device Simulation
Wafer 1
Wafer 2
Output Wafer 1
Output Wafer 2
Input–Deck Database
Connections
Wafer1.Drain
Wafer2.Source
Output Quantities
Electric Field

Fig. 1. Data flow for a mixed-mode device simulation. In this example, geometry and dopant concentrations from two wafers (Wafer 1 and Wafer 2) are taken as input to the device simulator. The names of the devices and contacts that are connected are stored in a separate file, the input-deck. This file also contains simulator specific options like the models to use in the simulation or if and what quantities have to be stored on the optional output WAFERs.

B. WAFER Description

Commonly, the term "WAFER" denotes a circular disk that serves as base material in the semiconductor fabrication process. Hundreds of process steps are performed on a WAFER to create devices like transistors or diodes. These devices are arranged in functional units, so called dies, each comprising an integrated circuit (IC). Depending on the functional complexity of the IC, the used process technology, and the size of the WAFER disk, there are up to several thousand dies that are produced on one WAFER. This amounts to a gigantic number of individual devices. In TCAD simulations, only a very small number of devices or even just a fraction of a single device is considered. Compared with a circuit or logic simulator, a TCAD simulator operates at a very low level of abstraction. Therefore, in the TCAD field, the term WAFER is used to denote the very small fraction of a WAFER disk that is used during TCAD simulations.

A TCAD conform WAFER description contains the geometry (topography) of the device structure, and quantities as they are used by the simulator models (e.g., dopant concentrations, forces, stress, etc.). Boundary information is used to identify parts of the surface of a simulation domain, and is necessary to define the geometrical region for boundary conditions (e.g., contacts of a device). A boundary information may also hold properties like, e.g., the material type of a contact or quantities like interface charges between two regions.

C. Simulator Control

Additionally, to the definition of geometry and grid information and the quantities associated with a grid or a geometry, a simulator needs some extra information. This information includes

- the definition of the data flow (i.e., the names and file formats of the persistent input and output WAFERs);
- the definition of the models to apply and the quantities to treat in the simulation respectively;
- the definition of simulator internal parameters like, e.g., the time step length, the number of time steps, or the iteration scheme that is to be used in the equation solver;
- a material database;
- some simulations require the definition of circuitry information.

This additional information is either stored in an extra file, the so-called input-deck, or passed on the command line of the simulator, if applicable (depending on the number of parameters). Sometimes, a combination of command line options and input-deck is useful (e.g., to supply input and output file names on the command line, but use an input-deck for model parameters).

An input-deck database (IPD) that is capable of holding an arbitrary number of (simulator specific) parameters was developed at the Institute for Microelectronics [19].

This input-deck database is also used to define circuitry information necessary to run mixed-mode device simulations with the device simulator MINIMOS-NT [20], [21]. In the case of a mixed-mode simulation, several input WAFERs are used. Connections between the WAFERs (i.e., the circuitry information) are established by identifying the contacts of the devices. Fig. 1 depicts the data flow for the case of a mixed-mode device simulation.

D. Classification of Process Steps

In order to design a library that meets all of the requirements defined above, the different kinds of process steps must be identified. In a coarse differentiation of simulation steps one could think of the class of simulators that only manipulate or read quantities (diffusion, implantation, device simulation), versus the topography manipulating tools (etching, deposition, oxidation).

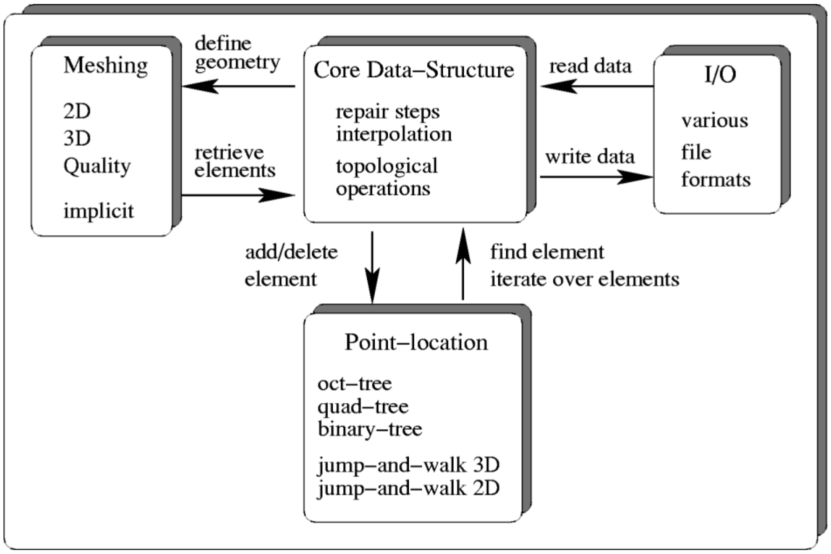

Fig. 2. Concept of the WAFER-STATE-SERVER. The three major problems existing in a TCAD data model are expressed by means of several dimension independent interface classes. The meshing interface consists of classes and methods to define a geometry, to start the gridding mechanism and to retrieve the generated grid elements. The *I/O* interface comprises a set of classes and methods to retrieve data from and to store data on a persistent WAFER, respectively. The core interface contains data structures to hold WAFER data and methods to perform data manipulations.

The first class of tools does not modify the underlying geometry at all. Instead, distributed quantities are used to perform a computation. A demand on the data model is to provide a boundary conforming grid.

Topography simulators need a geometrical view of the data. They need boundaries of the WAFER to ambient and interfaces between different regions. The topography is altered during such a process simulation. A deposition step will introduce completely new regions. In an etching step, existing regions can completely vanish, can be split into several regions, or can be merged into a single region. During an oxidation process, parts of the silicon region will turn into oxide. Some simulators also need information stored on a per region basis (e.g., the material type), the so-called properties, or distributed quantities (e.g., for simulating diffusion-coupled oxidation).

Among the above classifications, the topography simulators clearly impose the most complex data manipulations on the WAFER data structures, since they modify the underlying geometry.

III. WAFER-STATE-SERVER

The major requirements, as outlined in the previous section, are attacked by providing the tool developer with a class library, the WAFER-STATE-SERVER. The library is realized as a set of modules, each representing one of the analyzed problems (gridding, *I/O*, WAFER algorithms). The core data structures that hold the data make use of this modules. Fig. 2 gives an overview of the modules and how they are used in the core-data structures. The library is designed adhering to the *object-oriented* paradigm, by strictly separating the interface to a module (API) from its implementation. This ensures that the implementation, i.e., the algorithm of a given problem, is completely decoupled from the application program and can easily be exchanged without the need to change the application itself. The WAFER-STATE-SERVER provides a dynamic instantiation mechanism that allows one to easily implement a command line selectable choice of available modules (e.g., gridder or *I/O* module).[4] Fig. 3 gives an overview of the interface classes that are visible from the application.

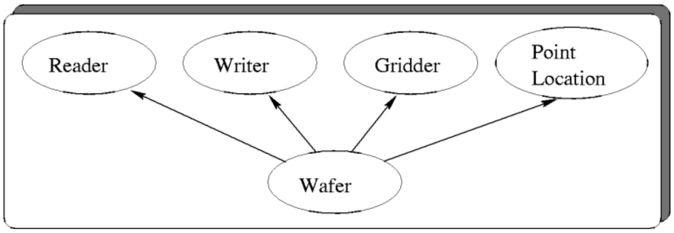

Fig. 3. Public interface classes of the WAFER-STATE-SERVER.

A. Data in a WAFER

There are two representations of the WAFER data. One comprises objects that are stored in memory (core data structures), and in the other, data are stored persistently on disk. Fig. 4 depicts the persistent data representation of a WAFER. Data are organized in sections. Some of the sections recursively contain subsections. There are two sections that must be present at the top level. These are the Segments and the Points section.

The points section contains the coordinates of all points of the wafer, it is a global list of points. All grid elements share this point list. This prevents the storage of redundant coordinate information.

The segments section may contain an arbitrary number of so-called stand-alone grids and, also, an arbitrary number of attributes, boundaries, and properties. Attributes are used to store

[4] Depending on the runtime environment (dynamic linker, libraries), dynamic instantiation is directly performed via the dl interface (dlopen) or a programmatic construct (switch). In the latter case, the instantiation is not really dynamic in the narrower sense, because all possible implementations must be known at compile time. The mechanism is, however, transparent to the application.

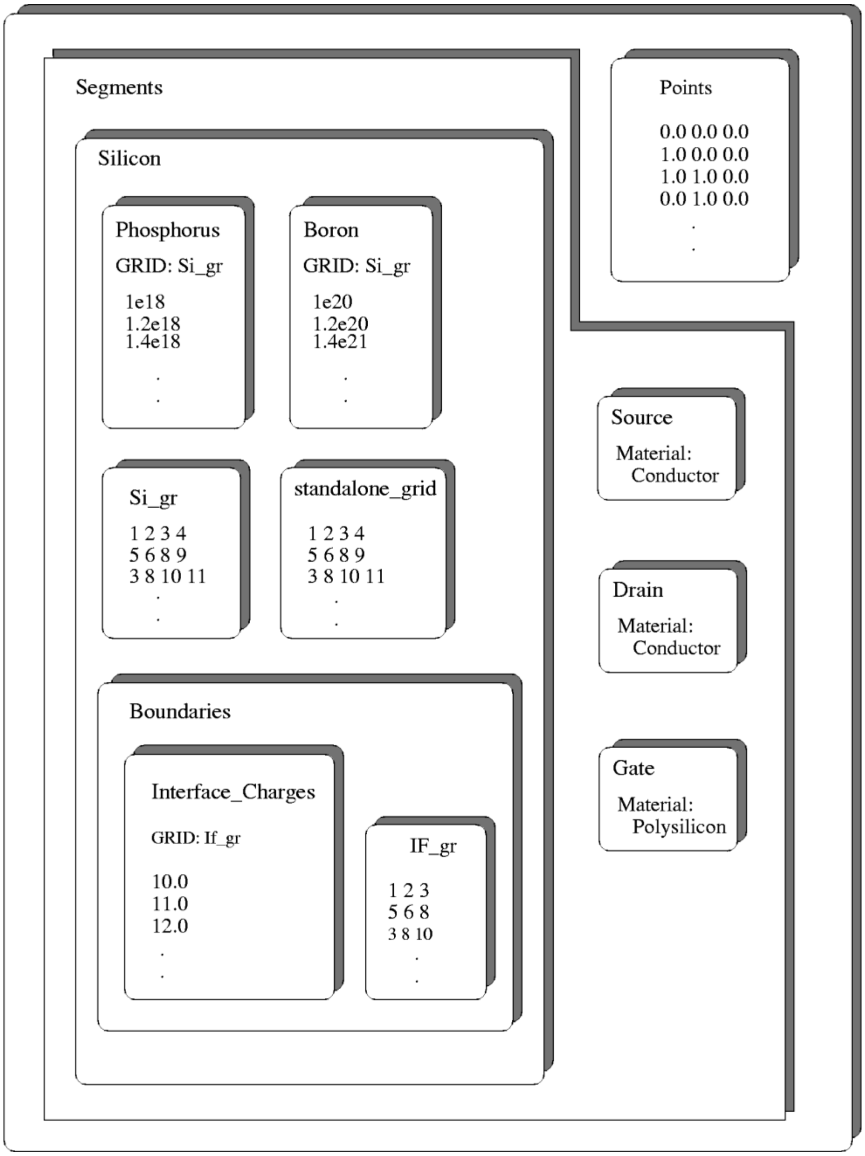

Fig. 4. Data in a WAFER.

distributed quantities. A distributed quantity is always defined on a grid, by defining a value for each grid point. The list of attribute values is separated from the grid. This makes it possible that one grid is shared among an arbitrary number of attributes. A boundary holds an optional number of attributes and properties. Properties are like attributes but do not reference a grid. They are constant over the whole segment. Properties are used to store information like the material type of a segment. Stand-alone grids are optional in case at least one distributed quantity is defined on the segment, but must be present (to define the geometry of a segment) otherwise. A segment that consists only of the pure geometry information might be created from a topography simulator or from any kind of geometry modeling tool.

B. I/O Module

The *I/O* module is split into two parts, the reader and the writer. Each part consists of several interface classes that are used to traverse the data. The implementation of these classes takes care to handle data transfer to or from the underlying file or database.[5] This mechanism ensures transparency of the actual file format which, thereby, provides the basis for the data interchange among existing simulators. Fig. 5 depicts the algorithm that is used to read data from a persistent WAFER into memory. The data are retrieved hierarchically. All methods that start with the name next are iterators over the contents of a section. They return an instance of an object or indicate the end of a section. The sequence of the method calls is mandatory and must occur as described below. The actual reading process is started with an object of type reader. Each invocation of the nextPoint method reads exactly one point from the persistent data source. A call to the nextSegment method returns an object of type RdSegment or indicates the end of the WAFER data. The method nextGrid of the RdSegment class returns an object of type RdGrid for each stand-alone grid. The method nextWafGridEl retrieves a grid element. For each attribute of a segment, the method nextAttribute

[5] The actual location (file or database) of the data is transparent.

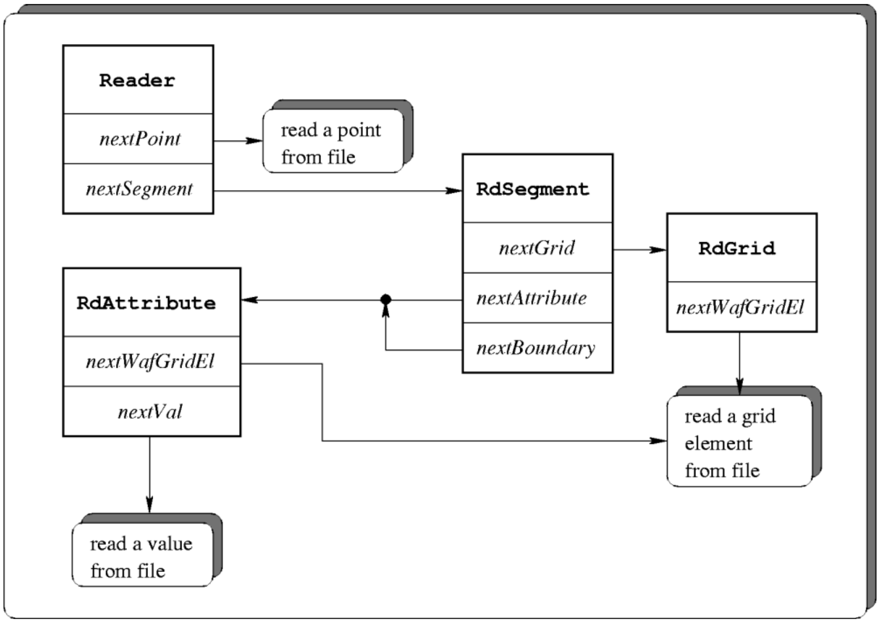

Fig. 5. *I/O* reading algorithm.

returns an object of type RdAttribute. This class contains a predicate isConstAttr that must be used to determine whether the attribute is stored on a grid. For distributed quantities the method nextWafGridEl must be used to retrieve the grid elements of the attribute. Note that the same grid may be shared by several attributes, therefore, a grid may only be retrieved at the very first occurrence. The predicate isNewGrid of the RdAttribute class indicates the first occurrence of a grid. After the last attribute was retrieved, the method nextBoundary must be invoked for each stored boundary. Since boundaries are quite similar to attributes, they are also handled by the RdAttribute interface class.

The algorithm to make data persistent is realized in a similar fashion. Again, several interface classes must be used hierarchically.

Presently, the following implementations for the *I/O* module exist.

- DF-ISE: The textual representation of the file format of the ISE company is supported. An implementation for a reader and a writer is available.
- FEM: This is the underlying file format used in the SMART ANALYSIS PACKAGE (SAP) [22]. SAP is capable of extracting capacities, resistances, and inductances of interconnect structures, and to perform thermal analysis of interconnect structures. Only the Reader is implemented.
- WSS: This is a recently developed ASCII file format. Its main purpose is to allow easy-by-hand generations of simple WAFER structures. Both reader and writer are supported.
- HDF: This file format was also developed recently. It is intended as the native file format for all simulators developed at the Institute for Microelectronics. It supports data compression, parallel *I/O*, and is binary compatible to all major computer architectures available today. Both reader and writer are implemented.

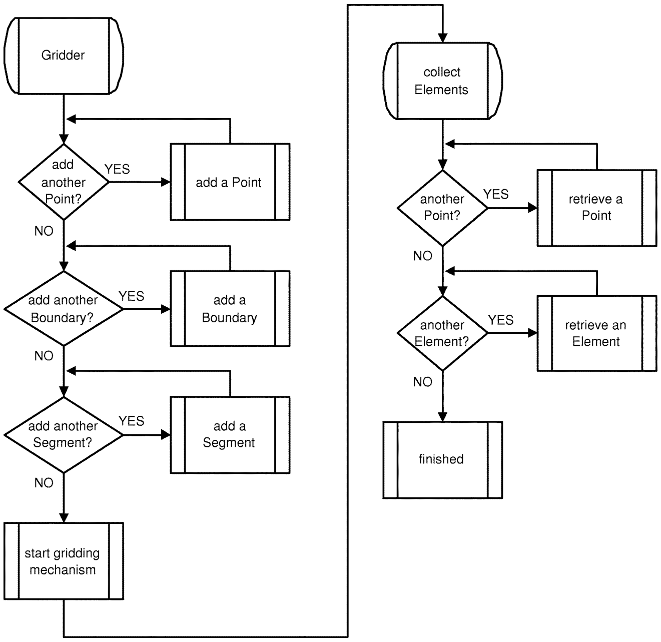

Fig. 6. Gridding algorithm.

C. Gridding Module

The gridding module allows to handle the creation of grids. The API consists of a part that is used to define the geometry and a part to retrieve the grid elements. Fig. 6 depicts the whole gridding algorithm. First, all points of all of the segments need to be defined. This also includes the points that are not part of the hull of a segment (inner points). The points are numbered consecutively in the way they are inserted. In the next step, the boundaries of all regions are defined. This is done by adding one boundary element at a time. For a 3-D gridder, a boundary element consists of three point references (surface triangle), a 2-D gridder expects two point references (surface line). The boundaries are numbered in the same way as the points. As a last step, the boundary elements that define a segment must be identified. Each segment is defined through a list of boundary element numbers and a unique user defined number. When the grid elements are retrieved, each element contains the segment number of the segment it belongs to. As a last step, the actual gridding process must be initiated by calling the start method. Once this method returns, the gridding process is finished and the generated points and grid elements can be retrieved from the gridder data structures.

The interface is realized in a completely dimension-independent way. The data structure of the returned grid element has a fixed size (four point references, the number of valid point references, and the segment number). The geometrical dimension of the grid element is indicated by the number of valid point references.

Note that in order to implement the WAFER-STATE-SERVER gridding interface for a certain gridding algorithm, no access to the source code of the algorithm is necessary. A library that provides a well-documented API will suffice for that purpose. Currently, two implementations of the gridding interface are available. These implementations are the 2-D grid generator TRIANGLE and the 3-D grid generator DELINK.

1) TRIANGLE: TRIANGLE was developed at Carnegie Mellon University [23]. It supports the generation of high-quality Delaunay triangulations with a number of tuning parameters to define quality criteria. It is implemented in the C programming language. The software is available as a stand-alone program and as a library. To integrate TRIANGLE into the WAFER-STATE-SERVER the library version was used and a C++ wrapper class was implemented to hide implementation details like memory management. This wrapper class was then used to implement the WAFER-STATE-SERVER gridder interface.

2) DELINK: DELINK [24] is a 3-D grid generator which was developed at the Institute for Microelectronics. It supports the generation of 3-D high-quality Delaunay meshes with directional refinement. DELINK is available as a stand-alone program and as a library. As with TRIANGLE the program is written in C. Therefore, a C++ wrapper class was developed to properly handle allocation and deallocation of DELINK data structures. For the implementation of the gridder interface, this C++ class was used.

D. Algorithm Module

This module's responsibilities include the following.

- Provide data structures to hold the WAFER data in memory. It uses algorithms of the *I/O* module to retrieve data from and to transfer data to a persistent WAFER.
- Provide algorithms to manipulate the topography of a WAFER to directly support etching, deposition, and oxidation simulations.
- Provide algorithms to extract topological information like boundaries or interfaces between segments. Since this information is redundant, it is not stored on the persistent representation, but computed on demand.
- Provide algorithms for *point location* and interpolation.

1) Topography Manipulating Operations: A topography simulator first reads in data from a persistent WAFER. After the data have been retrieved and (eventually) converted to a simulator specific internal data format (e.g., cellular representation) the actual topography simulation (etching, deposition) is performed. After the simulation has finished, the resulting topography needs to be merged with the WAFER-STATE. This merge operation is fully handled in the WAFER-STATE-SERVER via the GNU TRIANGULATED SURFACES (GTS) [25] library according to the following algorithm.

```
1. The surfaces of the new topography and the old WAFER-STATE are extracted and copied into GTS internal data structures.
2. Boolean (solid modeling) operations are performed with the two surfaces.
3. The result of the previous step is coarsened via the coarsening algorithms of GTS.
4. The coarsened result is then used to extract the new WAFER front. The (new) WAFER-STATE-SERVER internal data structures are built from this result, automatically preserving consistency. Note that this step also includes regridding steps that are necessary to produce grids for modified or newly created regions.
5. As a last step, attributes that were introduced by the topography simulation (e.g., dopant concentrations) are transferred to the newly created regions (grids).
```

2) Point Location and Interpolation: Prior to performing an interpolation, the element containing the point to interpolate must be determined (*point location*). A local interpolation algorithm strongly depends on an efficient *point location* algorithm. Several algorithms exist for the *point location* problem, two of which are implemented in the WAFER-STATE-SERVER. The first algorithm is tree-based and was implemented for all three dimensions (*oct-tree* [26], *quad-tree*, and *binary-tree*). The second algorithm is based on the *jump-and-walk* [27] algorithm and is available in 3-D. Our *jump-and-walk* algorithm uses a *point bucket oct-tree* for the search of the start tetrahedron. The tree-based solution gives a search time of $O(\log(n))$ but has the major drawback of rather long preprocessing times (especially in the 3-D case). The *jump-and-walk* algorithm has higher search times [$\approx O(n^{1/(D+1)})$ for searching a Delaunay triangulation of n elements in dimension D] but does not require any preprocessing of the data. The decision whether an application uses the *oct-tree* or rather the *jump-and-walk* algorithm strongly depends on the total amount of *point locations* that will be performed during the whole simulation. Therefore, the actual algorithm that is used can by chosen by the application developer.

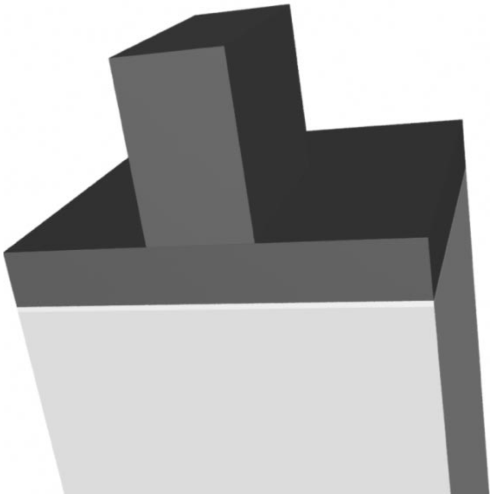

Fig. 7. Input WAFER data as they are input to the etch simulator. The colors of the regions indicate the material type. The blue colored material is a resist material and acts as a mask. The thin gray colored layer depicts an oxide layer, the brown material is nitride, and the green material at the bottom is silicon.

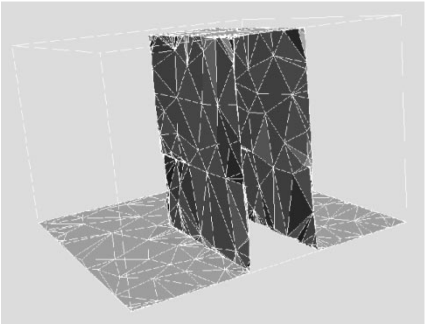

Fig. 8. Etch front as delivered from the etch simulator. The shown polygonal surface is the result of a coarsening algorithm that is applied to the fine initial cellular etch front that is returned by the etching algorithm.

IV. Application

All simulators that are presently developed at the Institute for Microelectronics are based on the new data model. The developed process simulators are a 3-D Monte Carlo ion implantation simulator [28], a 3-D diffusion and oxidation simulator [29], and a 3-D topography simulator. In this section, we will present, as an example, the 3-D etching and deposition simulator ETCH3-D.

Etching is used in various steps during the fabrication of an IC. It is, e.g., used to transfer the pattern comprising the design of the IC layout from the resist layer as it is created by lithography onto the WAFER. Another example for an etch step is to remove layers of material (e.g., masks) from the WAFER. Several techniques to simulate etching and deposition processes were developed in the past.

One of our etching simulators [30], [31] is based on the cellular approach. This simulator poses by far the strongest demands on the WAFER-STATE-SERVER library. The simulator uses the *I/O* module to read and create a persistent WAFER, respectively. Upon start of the program, the data is first transferred from a persistent WAFER into memory. Next, a cellular representation of the WAFER hull is computed and transferred to the internal cell-based data structures of ETCH3-D and the actual etching (or deposition) process takes place. After the etching simulation is completed, a complex postprocessing step that extracts the geometry information from the etching front is performed. Finally, the extracted geometry is merged with the existing WAFER data, which is then made persistent. Figs. 7–11 depict an example of a topography-simulation step.

The example shown is part of a schematic shallow trench isolation step. It is worth mentioning that the simulator is capable of handling arbitrary polygonal geometries. Except for

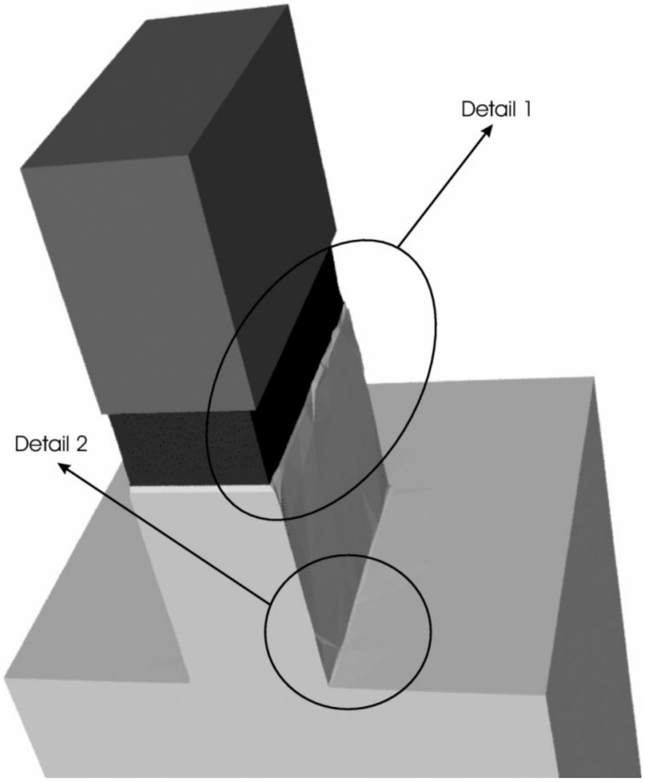

Fig. 9. Final outcome of the simulation as it is stored on the persistent WAFER. The structure results from a boolean operation of the hull of the original input WAFER (Fig. 7) with the etch front (Fig. 8).

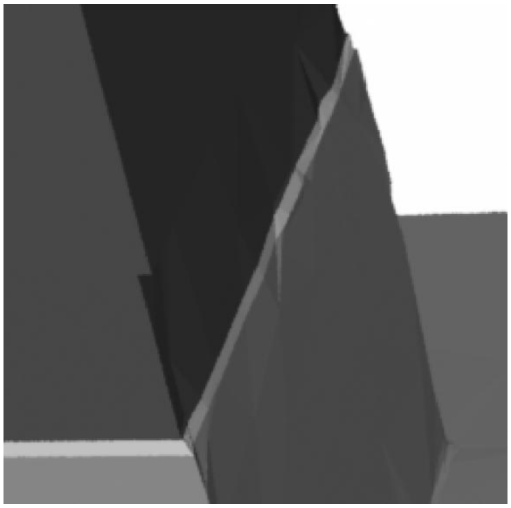

Fig. 10. Detailed view “1” of Fig. 9. The picture shows the under etching of nitride and oxide that takes place due to the imperfect anisotropy of the etching process. The different etching rates that apply to silicon, oxide, and nitride are visible.

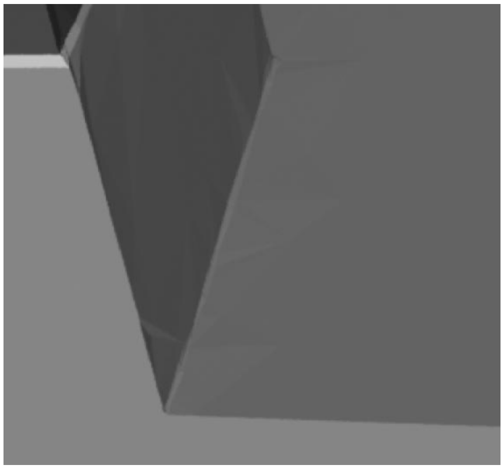

Fig. 11. Detailed view "2" of Fig. 9. The polygonal silicon surface is clearly visible.

the input WAFER (Fig. 7), the depicted examples are no manhattan geometries as they may result from traditional cellular etching algorithms, but are arbitrary polygonal structures. The detail views of Fig. 9 (Figs. 10 and 11) show the typical underetching artifacts as they occur in every etching process.

Note that a topography simulation is clearly the most complex process simulation concerning the output operation that creates the modified WAFER. Other process simulation steps, like, Monte Carlo Implantation might work only on a subset of the data stored on a WAFER. Only those regions that are actually exposed to the ion beam need to be updated. Here, a point-wise update of the dopant that is introduced suffices. For the case of diffusion a dopant is not introduced but changed. Both cases are handled with the same update operation, which is realized by a callback object into the simulator. The object gets invoked by the WAFER-STATE-SERVER once for each point. The simulator receives a spatial point and returns the value of the attribute at that point to the WAFER-STATE-SERVER.

V. Conclusion

We have presented a new *object-oriented* data model suitable for the TCAD tool developer. Major data model related issues in the TCAD field from the user's as well as the developer's perspective were pointed out. These issues include interoperability of tools, grid creation, and algorithms, like boundary extraction or topography manipulations as they are crucial in the TCAD field. These issues directly influenced the top-down design of the WAFER-STATE-SERVER. A strong adherence to the *object-oriented* design paradigm lead to a separation of interface (API) and implementation of a module. The identified problems, namely, what data belong to a WAFER-STATE, a transparent *I/O* mechanism, transparent meshing functions, and support for topological operations are directly supported and implemented in the WAFER-STATE-SERVER. The APIs of the developed modules were presented, and the applicability of the new data model to the TCAD field was demonstrated in a topography simulation application (which poses the strongest possible demands on the data model). Three process simulators presently developed at the Institute for Microelectronics are fully integrated with the WAFER-STATE-SERVER and take full advantage of the new concepts.

Future development of the WAFER-STATE-SERVER will have to focus on meshing issues. For accurately simulating diffusion, oxidation, and implantation processes, the demand to change (refine and coarsen) a grid during the simulation arises. Additionally, oxidation will deform the grid (move boundary and interfaces), such that it usually will no longer conform to criteria that were imposed on the initial grid (by the chosen gridding algorithm). For performance reasons, such meshing tasks clearly can not be attacked by the gridder interface as an invocation thereof results in a copy of the WAFER-STATE internal data structures to the data structures of the gridding algorithm and vice versa. Instead, a local refinement and coarsening strategy is necessary. Ideally, coarsening is realized by simply reverting a previous refinement step. Such algorithms, however, imply that the gridding algorithm must work directly on the simulator's internal data structures. From that point of view, it probably does not make sense to solve the refinement problem from a (external) library like the WAFER-STATE-SERVER. In the design of the refinement/coarsening algorithm, however, care should be taken to design algorithms that are capable of working on more than one data structure in order to allow a good overall reuse of the implemented meshing algorithms. The key concept to that is the template mechanism of the C++ programming language. Such algorithms could then be shared between the gridding module of the WAFER-STATE-SERVER and above mentioned meshing tasks.

The presented data model, as it is presently implemented, supports data exchange with tools from a major TCAD vendor and can easily be extended to also support other vendor's tools. It is, however, up to the vendors to decide whether "intervendor" interoperability of tools will be supported at all in future releases of their products. In case interoperability shall be supported, the WAFER-STATE-SERVER presents one possible solution by providing a strong definition of a WAFER-STATE.

References

[1] F. Fasching, "The Viennese Integrated System for Technology CAD Applications-Data Level Design and Implementation," Ph.D. dissertation, Tech. Univ. Vienna, Vienna, Austria, 1994.

[2] F. Fasching, W. Tuppa, and S. Selberherr, "VISTA-The data level," *IEEE Trans. Computer-Aided Design*, vol. 13, pp. 72–81, Jan. 1994.

[3] S. G. Duvall, "An interchange format for process and device simulation," *IEEE Trans. Computer-Aided Design*, vol. 7, pp. 741–754, July 1988.

[4] *ISE TCAD Manuals*, 4 ed., vol. 6, ISE Integrated Systems Engineering, 1997.

[5] ISE, Integrated Systems Engineering [Online]. Available: http://www.ise.ch.

[6] SYNOPSIS [Online]. Available: http://www.synopsys.com/products/avmrg/product_flow/technology_cad.html.

[7] *TMA TSUPREM-4, Two-Dimensional Process Simulation Program, Version 6.5 User's Manual*, Technology Modeling Associates, Inc., Sunnyvale, CA, 1997.

[8] *TMA Medici, Two-Dimensional Device Simulation Program, Version 4.0 User's Manual*, Technology Modeling Associates, Inc., Sunnyvale, CA, 1997.

[9] SYNOPSIS [Online]. Available: http://www.synopsys.com/products/avmrg/taurus_wb_ds.html.

[10] [Online]. Available: http://www.silvaco.com/.

[11] *ATHENA: 2D Process Simulation Framework, User's Manual*, Silvaco, 1993.

[12] SILVACO [Online]. Available: http://www.silvaco.com/products/vwf/atlas/atlas/atlas_br.html.

[13] M. D. Giles, D. S. Boning, G. R. Chin, W. C. Dietrich, M. S. Karasick, M. E. Law, P. K. Mozumder, L. R. Nackman, V. T. Rajan, D. M. H. Walker, R. H. Wang, and A. S. Wong, "Semiconductor wafer representation for TCAD," *IEEE Trans. Computer-Aided Design*, vol. 13, pp. 82–95, Jan. 1994.

[14] M. E. Law, C. S. Raffery, and R. W. Dutton, "New n-well fabrication techniques based on 2-D process simulation," in *IEDM Tech. Dig.*, 1986, p. 418.

[15] C. Yang and M. D. Giles, "Architecture and implementation of 3D field support in semiconductor wafer representation," in *Proc. Int. Workshop Numerical Modeling Process. Devices Integrated Circuits NUPAD V*, Honolulu, HI, 1994, pp. 81–84.

[16] Z. H. Sahul, K. C. Wang, Z.-K. Hsiau, E. W. McKenna, and R. W. Dutton, "Heterogeneous process simulation tool integration," *IEEE Trans. Semiconduct. Manufact.*, vol. 9, pp. 35–48, Feb. 1996.

[17] J. P. McVittie, J. C. Rey, and K. C. Saraswat, *Speedie User's Manual, Version 0.3*. Stanford, CA: Stanford Univ., 1991.

[18] M. Duane, "TCAD needs and applications from a user's perspective," *IEICE Trans. Electron.*, vol. E82-C, no. 6, pp. 976–982, 1999.

[19] R. Klima, T. Grasser, T. Binder, and S. Selberherr, "Controlling TCAD applications with a dynamic database," in *Proc. Software Eng. Applicat.*, M. H. Hamza, Ed., Las Vegas, NV, 2000, pp. 103–112.

[20] T. Binder, K. Dragosits, T. Grasser, R. Klima, M. Knaipp, H. Kosina, R. Mlekus, V. Palankovski, M. Rottinger, G. Schrom, S. Selberherr, and M. Stockinger, *MINIMOS-NT User's Guide*: Institut für Mikroelektronik, 1998.

[21] T. Grasser, "Mixed-Mode Device Simulation," Ph.D. dissertation, Tech. Univ. Vienna, Vienna, Austria, 1999.

[22] R. Sabelka and S. Selberherr, "SAP — A program package for three-dimensional interconnect simulation," in *Proc. Int. Interconnect Technol. Conf.*, Burlingame, CA, June 1998, pp. 250–252.

[23] J. R. Shewchuk, "Triangle: Engineering a 2D quality mesh generator and delaunay triangulator," in *Proc. 1st Workshop Appl. Computat. Geom.*, 1996, pp. 124–133.

[24] P. Fleischmann, "Mesh Generation for Technology CAD in Three Dimensions,", Tech. Univ. Vienva, Vienna, Austria, 1999.

[25] GTS – The GNU Triangulated Surfaces Library [Online]. Available: http://gts.sourceforge.net

[26] C. Weichselbaum, "Entwicklung Eines Finite Oct-Tree,", Tech. Univ. Vienna, Vienna, Austria, 1999.

[27] E. P. Mücke, I. Saias, and B. Zhu, "Fast randomized point location without preprocessing in two- and three-dimensional delaunay triangulations," in Proc. 12th Annu. Symp. Computat. Geom., 1996, pp. 274–283.

[28] A. Hössinger, "Simulation of Ion Implantation for ULSI Technology,", Tech. Univ. Vienna, Vienna, Austria, 2000.

[29] H. Ceric, A. Hössinger, T. Binder, and S. Selberherr, "Modeling of segregation on material interfaces by means of the finite element method," in *Proc. MATHMOD Conf.*, Vienna, Austria, Feb. 5–7, 2003, pp. 139–145.

[30] A. Hössinger, T. Binder, W. Pyka, and S. Selberherr, "Advanced hybrid cellular based approach for three-dimensional etching and deposition simulation," in *Proc. Simulation Semiconduct. Process. Devices*, Athens, Greece, Sept. 2001, pp. 424–427.

[31] W. Pyka, H. Kirchauer, and S. Selberherr, "Three-dimensional resist development simulation – benchmarks and integration with lithography," *Microelectron. Eng.*, vol. 53, no. 1–4, pp. 449–452, 2000.

Thomas Binder was born in Bad Ischl, Austria, in 1969. He received the Diplomingenieur degree in electrical engineering and computer science at the Technical University of Vienna, Vienna, Austria, in 1996.

During his studies he was working on several software projects, mainly in the computer-aided design, geodesy, and security fields. In March 1997, he joined the Institute for Microelectronics, Vienna, Austria, where he is currently working on his doctoral degree. In autumn 1998, he held a visiting research position at Sony, Atsugi, Japan. His scientific interests include data modeling, algorithms, software engineering, and semiconductor technology, in general.

Andreas Hössinger was born in St. Pölten, Austria, in 1969. He received the Dipl.-Ing. degree in technical physics, in 1996, and the doctorate degree, in 2000, from the Technical University of Vienna, Vienna, Austria, where he is currently enrolled as a postdoctoral researcher.

In 1998, he was a Visiting Researcher with Sony Technology Center, Atsugi, Japan. In 2001, he was a Visiting Researcher at LSI Logic, Santa Clara, CA, within the scope of a cooperate research project on 3-D process simulation. His research interests include process simulation with special emphasis on 3-D applications.

Siegfried Selberherr (M'79–SM'84–F'93) was born in Klosterneuburg, Austria, in 1955. He received the Dipl.-Ing. degree in electrical engineering and the doctoral degree in technical sciences from the Technical University of Vienna, Vienna, Austria, in 1978 and 1981, respectively.

He has held the *venia docendi* on computer-aided design since 1984. Since 1988, he has been Head of the Institute for Microelectronics and, since 1999, has been Dean of the Faculty for Electrical and Information Technology. His current topics are modeling and simulation of problems for microelectronics engineering.