OBJECT-ORIENTED WAFER-STATE SERVICES

T. Binder and S. Selberherr
Institute for Microelectronics, TU Vienna
Gusshausstr. 27–29, A-1040 Vienna, Austria
Phone +43-1-58801-36036, FAX +43-1-58801-36099
e-mail: Thomas.Binder@iue.tuwien.ac.at

KEYWORDS

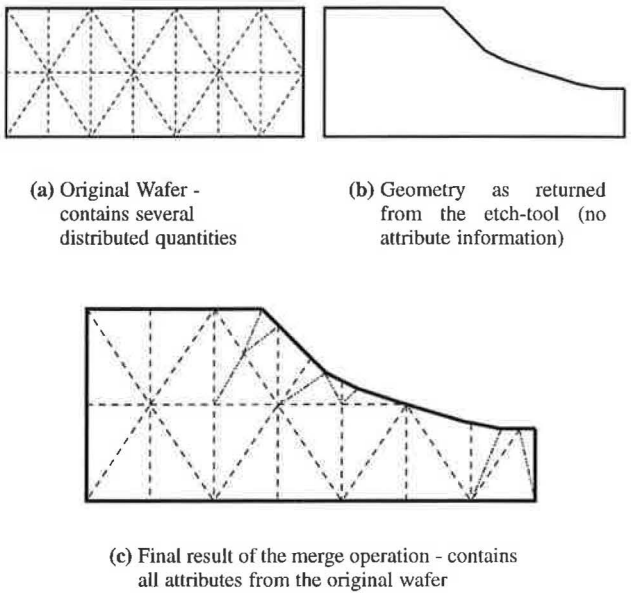
TCAD, Simulators, Optimization, Intelligent simulation environments, Semiconductor technology

ABSTRACT

In a modern Technology CAD (TCAD) simulation environment data exchange between different simulators often remains as unsolved challenge. One natural way of data interchange is to use a file format common to all tools involved in the process flow. This approach, however, lacks the functionality often required between certain steps. First, there is the need to ensure a consistent input-wafer for each individual simulator. Second, depending on the type of process simulation at hand, it must be possible for a certain simulator to operate only on a sub-set of the data contained on a wafer. For instance, for a topography tool like an etching simulator only the geometry and material informations are of concern, whereas data like impurity concentrations and meshes are usually ignored. Problems arise when the etching tool is writing back its results into a file. Since it has no knowledge of other data contained in the input wafer, there is no way for the tool to write a valid wafer-state. This problem can only be solved by merging the newly generated geometry of the etch-step (Fig. 1(b)) with the original input-file (Fig. 1(a)) , to create a new consistent wafer-state (Fig. 1(c)). The resulting geometry is automatically regridded (new elements were inserted in this example), and all distributed quantities are transferred onto the new points.

WAFER-STATE SERVER

Our wafer-state server addresses this kind of problems and presents a standardized application programming interface (API) common to all simulators and tools. This API defines a strong protocol the simulators must adhere to. Tools must manipulate data in the wafer-state exclusively through this protocol. The wafer-state server also contains gridding and regridding capabilities. These are required for repair steps as outlined in the above example and are invoked transparently. The strategy chosen for interpolation allows different interpolation methods for each attribute without any reflection in the API. The user simply requests the value of an attribute at a certain point. The wafer-state server chooses the appropriate (configured) interpolation method for the attribute, and returns the interpolated value. Prior to interpolating, the grid-element in which the point is contained has to be found (point location). This task is achieved with a finite-quad-tree and finite-oct-tree [Binder and Selberherr, 1999] based data structure for two-dimensional and three-dimensional applications respectively. These tree based data-structures support the wafer-state server in (a) performing efficient point-locations and (b) in identifying the grid elements in the repair step for which an intersection calculation has to be carried out.



(a) Original Wafer - contains several distributed quantities

(b) Geometry as returned from the etch-tool (no attribute information)

(c) Final result of the merge operation - contains all attributes from the original wafer

Figure 1: Merge operation after an etch-step

Modularity

Due to the object-oriented approach (Fig. 2) the wafer-state server allows for easy integration of new file formats and meshing tools. It is worthwhile mentioning that the underlying file format and the (re-)gridding process are totally hidden from the user. The API contains no functions to directly manipulate a file or a grid. Instead, the user instantiates the READER and GRIDDER object of his choice. This mechanism enables the developer of a simulator to easily choose among all supported file formats and gridding algorithms.

The use of several individual modules (READER, GRIDDER, ...) as opposed to using only a single one (WAFER) may seem complicated at first sight, however, it

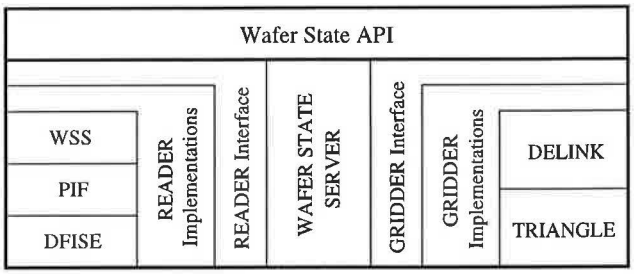


Figure 2: Basic block diagram of the wafer-state server

introduces two major advantages: On the one hand details about the underlying file format and about the gridding algorithms are very well hidden to the wafer-state server's core functions. This ensures that reading support for another file-format can be added later by implementing a READER for this file-format. The same holds true for supporting various WRITERS and GRIDDERS. In general, any class that implements the interface to a certain module qualifies as a wafer-state module. On the other hand, settings specific to a certain implementation of a GRIDDER (e.g. quality constraints) can directly be accessed by the simulator without the need for extra wafer-state functions.

Currently READERS for PIF [Fasching et al., 1991], DFISE [ISE, 1997] and the newly developed WSS file formats as well as the GRIDDERS DELINK [Fleischmann and Selberherr, 1996] (three dimensional) and TRIANGLE [Shewchuk, 1996] (two dimensional) are implemented.

Protocol between application and wafer-state server

The first step in the protocol (Fig. 3) from an applications' point of view is to instantiate appropriate READER and GRIDDER objects. The choice of what READER to instantiate depends on the file-format in which the data are stored. Next, the actual wafer object is instantiated by supplying the READER and GRIDDER objects as well as a CONFIG object to the wafer class. The CONFIG object holds information about the different kinds of supported attributes and about the process step which is to be carried out.

At this point the data in the wafer-state server are ready for the application to be requested.

In the next protocol step the simulator requests the geometry and thereon stored attributes. Geometries and attributes are identified by names. These names are usually stored in a so called input-deck which is read by the simulator independently from the wafer definition. This input-deck contains several settings for the simulator, among other details, it identifies which regions are to be treated in the simulation.

All relevant data have now been transferred to the simulator. Once the simulator has finished its calculations the results

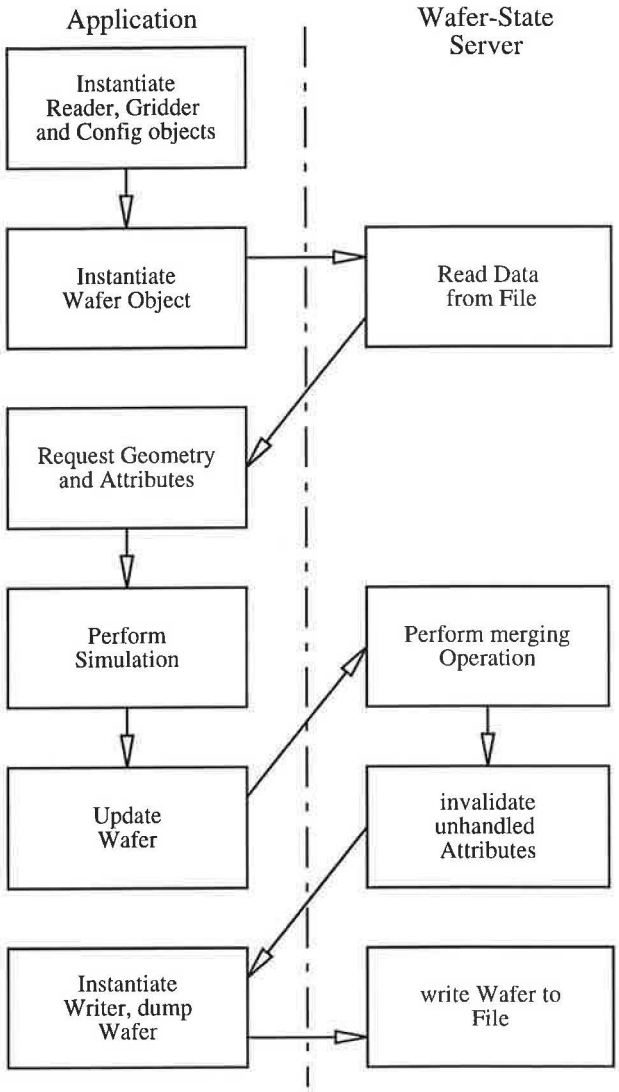


Figure 3: Basic protocol between wafer-state server and application

must be merged with the wafer-state. This so called update operation is performed in several individual steps. Each attribute (including the grid it is stored on) which has been requested must now be stored back onto the wafer. The wafer-state server then checks whether all attributes were received. Next the attributes which are invalidated by this process step are deleted.

Now the repairing mechanism is invoked by the wafer-state server. The newly added geometry is clipped with the one stored on the wafer-state. Grid points are taken over from the old grid where possible. The regions are then meshed using the supplied GRIDDER. All attributes which were not treated by the simulator and which are not configured to be invalidated are interpolated onto the new geometries. Attributes which lie on no longer existing regions (e.g. a segment was altered by the simulator) are discarded.

After the repair operation the application instantiates the appropriate WRITER object and invokes the dump method

of the wafer class to permanently store the simulation results on a file.

Definition of process steps

Each class of process step is configured in a database. The necessity for such a classification is best illustrated in an example. Take, for instance, a diffusion step: If the input wafer contains any stress components, the diffusion simulator either must take them into account (use them in the simulation) and perform an update when writing back the results, or the wafer-state server has to invalidate these components right after the diffusion step. All attributes which are modified either directly by the simulation or indirectly (e.g. invalidated stress component) must be listed.

```
Diffus
{
    Invalidate
    {
        inv = "*"; // quantities to invalidate.
        exc = "";  // qu. to exclude from invalidation
    };

    read  = "Boron, Arsenic";
    write = "Boron, Arsenic";

    topography = false;
};
```

Figure 4: Configuration of diffusion step

Fig. 4 shows a possible configuration of a diffusion step. The keywords `inv` and `exc` specify quantities which are to be invalidated and excluded from invalidation respectively. The keyword `read` lists all attributes which must be treated by the simulator (here: `Boron, Arsenic`). Similarly `write` specifies all attributes which must be supplied in the update operation. An asterisk (`*`) can be used with `inv` and `exc` to denote all contained attributes, however, attributes in the `read` and `write` statements will overrule this notion. This means that in our example the quantities `Boron` and `Arsenic` are not invalidated automatically. In case the wafer contains other attributes (e.g. stress) they will be removed upon update. The keyword `topography` is used to indicate whether the simulator changes the topography (`true`) in which case the wafer-state server must merge the new geometry with the existing one.

The types of the various attributes are also defined in the database. Fig. 5 depicts the configuration of attributes of type Concentrations.

```
Concentrations
{
    interpolation = "log";
    unit          = "1/cm^3";
    members       = "Donors, Acceptors, Boron,
                     Phosphorus, Arsenic, Indium,
                     Antimony, Nitrogen, Oxygen";
    datatype      = "Scalar";
};
```

Figure 5: Configuration of attribute type Concentrations

The definition of an attribute class consists of the unit (cm^{-3}) the data type (scalar, vector, tensor), the name (Concentrations), a list of all possible instances (Arsenic, Phosphorus, Boron, donors, acceptors, ...), and the method of interpolation (linear, logarithmic).

TCAD ANALYSIS

Performing TCAD analysis tasks [Strasser, 1999] like optimization of VLSI semiconductor devices [Plasun et al., 1997, Plasun et al., 1998] or inverse modeling of doping profiles [Strasser et al., 1999] often results in an enormous number of individual simulation runs. Frameworks like SIESTA or VISTA [Strasser et al., 1997] take care of aspects like describing an experiment and queuing jobs on a cluster of workstations. Another aspect of TCAD analysis is how the input-data for the simulation runs are generated. Thus, the need for a tool to generate a wafer based on a textual description arises.

Input Wafer Creation

The wafer-state services contain a tool (MKWAFER) to create three-dimensional wafers suitable for a process or device simulation. This tool uses as input the input-deck language as it is also used in our device simulator MINIMOS-NT [Simlinger et al., 1995, Binder et al., 1998], and in the configuration of the process steps and attribute types within the wafer-state server.

```
#include "cube3d.ipd"

Contact: ~Cube { Scaling { z = 0.2; } }

Geometry
{
    Bulk: ~Contact { Scaling { y = 5.0; } }
    Silicon: ~Cube
    {
      Offset { z = 0.2; } Scaling { y = 5.0; }
    }
    Drain: ~Contact
    {
      Scaling { y = 1.0; } Offset { z = 1.2; }
    }
    Gate: ~Contact
    {
      Scaling { y = 3.0; } Offset { y = 1.0; z = 1.2; }
    }
    Source: ~Contact
    {
      Scaling { y = 1.0; } Offset { y = 4.0; z = 1.2; }
    }
}
```

Figure 6: Simple device geometry description

Fig. 6 shows the input-deck description to generate the schematic three-dimensional device structure depicted in Fig. 7. Note that, for sake of clarity only a wire-frame of the outline is displayed (no grid elements). For each section in the input-deck file (`Bulk, Silicon, Drain, ...`) a corresponding segment is created. After having processed the last section of the input-deck the program computes the boundary representation of the whole geometry. The computation is achieved by first transferring all

sets of coplanar faces into a two-dimensional representation. Second, a two-dimensional polygon clipping software [Schutte, 1995] based on an algorithm of Kevin Weiler [Weiler, 1980] is used to determine the intersections. Finally the resulting two-dimensional faces are transferred back into three-dimensional space and added to the structure. The boundary representation is then passed on to a gridder (DELINK [Fleischmann and Selberherr, 1996]) in order to generate a mesh of the whole structure. The final device is saved to a file using the WSS WRITER.

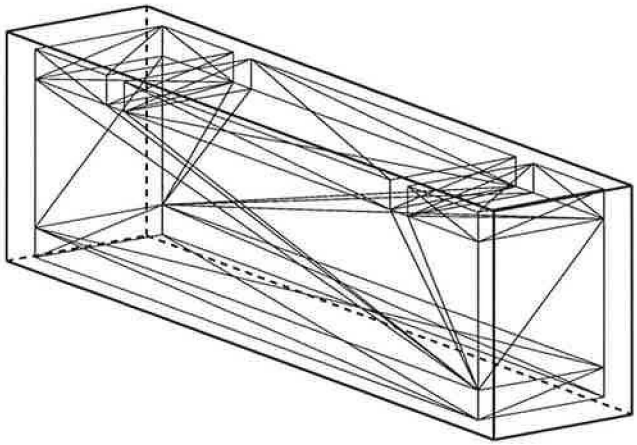


Figure 7: Wire-frame of schematic three-dimensional device

VISUALIZATION

Another important aspect addressed in the wafer-state server is the visualization of attributes and geometries. Due to the abstraction of the file access we only need to support one certain file format (WSS). The chosen visualization environment is the *Visualization Toolkit* (VTK) [Schroeder et al., 1999]. To keep the visualization platform independent the JAVA programming language is used for both parsing the WSS file and for the actual visualization (JAVA-VTK binding). The parser generator used (ANTLR [Schaps, 1999]) to generate the parser code is capable of producing JAVA and C++ parsers from the same language description, which relieves us from maintaining a separate JAVA and C++ version of the very same parser. The visualization runs on Unix and Windows Platforms.

IMPLEMENTATION

The chosen programming language for the implementation of the wafer-state server is C++. This language facilitates a full object-oriented design as realized in the wafer-state servers' core components as well as an easy integration of existing programs (DELINK, TRIANGLE, DFISE-READER) thus ensuring good overall code reusability. During the implementation of the wafer-state server care was taken to adhere to the ANSI C++ standard as close as possible.

CONCLUSION

We present a TCAD class library to support the simulation of semiconductor fabrication processes. A solution to the problem of data-exchange among various simulators is given. The strong protocol all simulators must adhere to allows for the simulation of a whole process flow.

Due to the abstraction of the file-format and the gridding mechanisms an integration of tools of different vendors is possible without having access to the source codes. The wafer-state server presents the tool developer with a powerful way of combining several independent modules. By instantiating the appropriate module the programmer has the flexibility of choosing among all available GRIDDERS, READERS, and WRITERS.

Finally, the challenge of optimization and inverse modeling is met by providing tools to create input-data for device and process simulation and to visualize the simulated results.

OUTLOOK

The simulation of whole process flows is still a tedious task in modern TCAD simulation environments. Usually conversion tools have to be invoked in order to couple simulators for different process steps. A conversion from one data format to another usually introduces a certain amount of error (e.g. due to interpolating data onto a new grid). Therefore, wafer-state support for the MONTE CARLO ION IMPLANTATION simulator [Hössinger et al., 1999, Hössinger and Selberherr, 1999] as well as for the ETCHING simulator [Pyka et al., 1998, Pyka and Selberherr, 1998a, Pyka and Selberherr, 1998b] is under development.

BIOGRAPHY

Thomas Binder was born in Bad Ischl, Austria, in 1969. He studied electrical engineering and computer science at the 'Technische Universität Wien', where he received the degree of 'Diplomingenieur' in December 1996. In March 1997 he joined the 'Institut für Mikroelektronik', where he is currently working for 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.

References

[Binder et al., 1998] Binder, T., Dragosits, K., Grasser, T., Klima, R., Knaipp, M., Kosina, H., Mlekus, R., Palankovski, V., Rottinger, M., Schrom, G., Selberherr,

S., and Stockinger, M. (1998). *MINIMOS-NT User's Guide*. Institut für Mikroelektronik.

[Binder and Selberherr, 1999] Binder, T. and Selberherr, S. (1999). A parallel finite oct-tree for multi-threaded insert, delete, and search operations. In *IASTED Int. Conf. Applied Modeling and Simulation*, pages 613–616, Cairns, Australia.

[Fasching et al., 1991] Fasching, F., Fischer, C., Selberherr, S., Stippel, H., Tuppa, W., and Read, H. (1991). A PIF implementation for TCAD purposes. In Fichtner, W. and Aemmer, D., editors, *Simulation of Semiconductor Devices and Processes*, volume 4, pages 477–482, Konstanz. Hartung-Gorre.

[Fleischmann and Selberherr, 1996] Fleischmann, P. and Selberherr, S. (1996). A new approach to fully unstructured three-dimensional Delaunay mesh generation with improved element quality. In *Simulation of Semiconductor Processes and Devices*, pages 129–130, Tokyo, Japan. Business Center for Academic Societies Japan.

[Hahn and Lehmann, 1997] Hahn, W. and Lehmann, A., editors (1997). *Proc. 9th European Simulation Symposium*, Passau, Germany. Society for Computer Simulation International.

[Hössinger and Selberherr, 1999] Hössinger, A. and Selberherr, S. (1999). Accurate three-dimensional simulation of damage caused by ion implantation. In *Proc. 2nd Int. Conf. on Modeling and Simulation of Microsystems*, pages 363–366, San Juan, Puerto Rico, USA.

[Hössinger et al., 1999] Hössinger, A., Selberherr, S., Kimura, M., Nomachi, I., and Kusanagi, S. (1999). Three-dimensional Monte-Carlo ion implantation simulation for molecular ions. In *Electrochemical Society Proceedings*, volume 99-2, pages 18–25.

[ISE, 1997] ISE (1997). *ISE TCAD Manuals vol. 6, release 4*. ISE Integrated Systems Engineering.

[Plasun et al., 1997] Plasun, R., Pichler, C., Simlinger, T., and Selberherr, S. (1997). Optimization tasks in technology CAD. In [Hahn and Lehmann, 1997], pages 445–449.

[Plasun et al., 1998] Plasun, R., Stockinger, M., Strasser, R., and Selberherr, S. (1998). Simulation based optimization environment and it's application to semiconductor devices. In *IASTED Int. Conf. on Applied Modelling and Simulation*, pages 313–316, Honolulu, Hawaii, USA.

[Pyka et al., 1998] Pyka, W., Martins, R., and Selberherr, S. (1998). Efficient algorithms for three-dimensional etching and deposition simulation. In Meyer, K. D. and Biesemans, S., editors, *Simulation of Semiconductor Processes and Devices*, pages 16–19. Springer, Leuven, Belgium.

[Pyka and Selberherr, 1998a] Pyka, W. and Selberherr, S. (1998a). Three-dimensional simulation of bulge formation in contact hole metalization. In *Proc. International Conference on Modeling and Simulation of Microsystems Semiconductors, Sensors and Actuators*, pages 65–69, Santa Clara, CA, USA.

[Pyka and Selberherr, 1998b] Pyka, W. and Selberherr, S. (1998b). Three-dimensional simulation of TiN magnetron sputter deposition. In Touboul, A., Danto, Y., Klein, J.-P., and Grünbacher, H., editors, *28th European Solid-State Device Research Conference*, pages 324–327, Bordeaux, France. Editions Frontieres.

[Schaps, 1999] Schaps, G. L. (1999). Compiler construction with antlr and java. *Dr. Dobb's Journal*. `http://www.ddj.com/articles/1999/9903/9903h/9903h.htm`, `http://www.antlr.org`.

[Schroeder et al., 1999] Schroeder, W., Martin, K., and Lorensen, B. (1999). *An Object-Oriented Approach To 3D Graphics*. Prentice Hall.

[Schutte, 1995] Schutte, K. (1995). An edge labeling approach to concave polygon clipping. *Submitted to ACM*. `http://www.ph.tn.tudelft.nl/People/klamer/clip.ps.gz`.

[Shewchuk, 1996] Shewchuk, J. R. (1996). Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. In *First Workshop on Applied Computational Geometry*, pages 124–133. Association for Computing Machinery.

[Simlinger et al., 1995] Simlinger, T., Kosina, H., Rottinger, M., and Selberherr, S. (1995). MINIMOS-NT: A generic simulator for complex semiconductor devices. In de Graaff, H. and van Kranenburg, H., editors, *25th European Solid State Device Research Conference*, pages 83–86, Gif-sur-Yvette Cedex, France. Editions Frontieres.

[Strasser, 1999] Strasser, R. (1999). *Rigorous TCAD Investigations on Semiconductor Fabrication Technology*. Dissertation, Technische Universität Wien.

[Strasser et al., 1997] Strasser, R., Pichler, C., and Selberherr, S. (1997). VISTA - a framework for technology CAD purposes. In [Hahn and Lehmann, 1997], pages 450–454.

[Strasser et al., 1999] Strasser, R., Plasun, R., and Selberherr, S. (1999). Practical inverse modeling with SIESTA. In *Simulation of Semiconductor Processes and Devices*, pages 91–94, Kyoto, Japan.

[Weiler, 1980] Weiler, K. (1980). Polygon comparision using a graph representation. In *Computer Graphics, 14*, pages 10–18. SIGGRAPH 80.