Proceedings of the IASTED International Conference
Software Engineering and Applications
November 6-9, 2000 Las Vegas, Nevada

OBJECT-ORIENTED DESIGN PATTERNS FOR PROCESS FLOW SIMULATIONS

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

ABSTRACT

Although simulators capable of dealing with the different kinds of processes in the simulation of semiconductor device fabrication are widely available, the combination of several of them to a whole process flow is still a complicated endeavor. This paper points out the difficulties arising in data exchange between simulators and we present as a solution the WAFER-STATE SERVER. Our WAFER-STATE SERVER is implemented as an object-oriented software package which allows the coupling of various tools of different vendors. The package also contains support for Technology CAD (TCAD) analysis tasks like optimization of the electrical characteristics of a semiconductor device or inverse modeling of doping profiles.

Keywords: Object-Oriented Analysis and Design, Semiconductor technology, TCAD environment, Process Flow

1 INTRODUCTION

In a modern Technology CAD (TCAD) simulation environment like SIESTA or VISTA [1] data exchange between different simulators often remains an unsolved challenge. One approach to attack this problem is to use a file format common to all tools involved in the process flow. The PIF [2] (profile interchange format) file format presents an implementation of such an approach. In a PIF file data are stored as a set of lists. A powerful API for C, FORTRAN and LISP accounts for ease-of-use for various simulators and applications like file converters or meshing tools. It turned out however, that the definition of the pure file format is not sufficiently addressing all kinds of problems arising in a process flow. In order to illustrate the difficulties the following section will briefly discuss of what data a wafer-state description actually must consist of.

1.1 WAFER-STATE DESCRIPTION

Since a real-world wafer may consist of several million devices (transistors, diodes, capacitors) per die only a small number[1] of them can be treated in a process or device simulation. As a consequence a suitable wafer-state description need not contain any circuit or macro-model information as it is used in circuit simulators like SPICE [3]. Instead, the information required consists of:

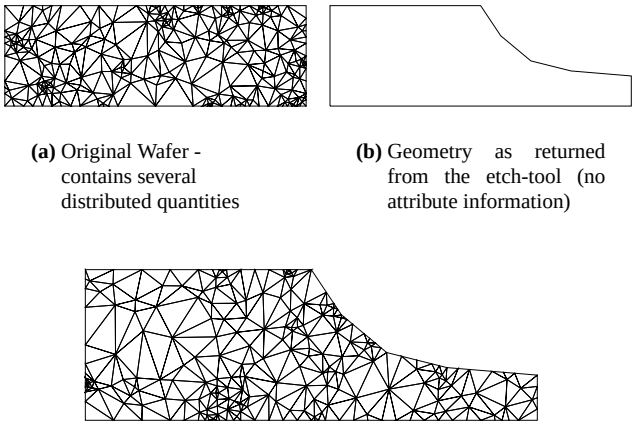
- geometry description
- properties (e.g., material type)
- distributed quantities (e.g., dopants)
- and grids on which the quantities are stored

Depending on the type of simulation carried out only a certain subset of the available data may be requested as input. Given that, one can already imagine the problems arising:

First, there is the need to ensure a consistent input-wafer for each individual simulator. This means that one needs a strong definition of what a certain simulator must read (input) what it must deliver as a result and also what (unhandled) quantities from previous simulation results are invalidated by the simulation and thus need to be removed.

Second, depending on the type of process simulation, it must be possible for a simulator to operate only on a subset of the whole data contained on a wafer. For instance, for a topography tool like an etching simulator only the geometry and material properties are of concern, whereas data like impurity concentrations and meshes are usually ignored. Problems arise once the etching simulation is finished and the tool is storing the results into a file. Since the result of the tool consists only of a pure geometry, the question of what happens to quantities stored on the (input) wafer arises. Such problems 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)). Note that the resulting geometry must be re-gridded (new elements were inserted in this example), and all distributed quantities need to be interpolated onto the new points.

[1] In case of a process simulation it is only one device or maybe even just a part of one device.



(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

1.2 TRANSPARENT INTERPOLATION

Interpolating values of distributed quantities onto newly generated grid points leads to the need for an automated mechanism. This is especially important since there are different attribute value types (scalar, vector). In order to keep the introduced interpolation error as small as possible different interpolation methods must be applied to the various attribute types. Some attributes (e.g., doping concentrations) require a logarithmic interpolation (the values are logarithmized before a linear interpolation takes place), others (e.g., stress components) need to be interpolated linearly.

2 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 re-gridding 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 binary tree, a finite-quad-tree and a finite-oct-tree [4] based data-structure for one-dimensional, 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 intersections with the geometry have to be computed.

2.1 MODULARITY

The WAFER-STATE SERVER is organized as a set of independent modules each dedicated to a certain task. These modules are a direct reflection of the problems discussed in the previous section. These modules are:

- READER module
  The READER module takes care of reading data from a certain file-format or database.
- WRITER module
  This module takes care of writing data to a certain file-format or database.
- GRIDDER module
  Handles all gridding problems. This is needed for repair steps.
- FINITE-OCT-TREE module
  This module represents the internal data-structure of the WAFER-STATE SERVER. It is used for point-location and to determine the grid elements which need to be repaired after a topography change.

Since the modules are realized as independent libraries, it is possible to use only some of them in a certain application. It is for example feasible to use the READER and WRITER modules to implement a file converter. There is no need to link an application against all of these modules or against the whole WAFER-STATE SERVER in case the provided services are not needed.

Each module is used by the WAFER-STATE SERVER via an interface class. Fig. 2 gives an overview of all used interface classes.

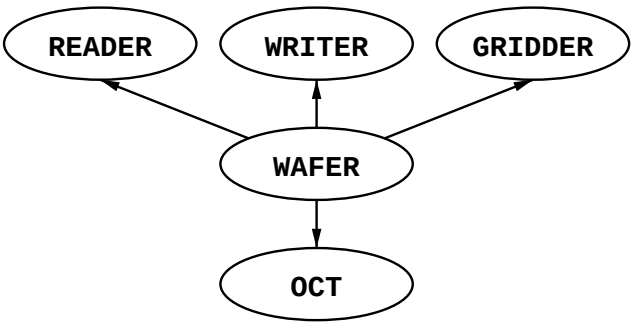


Figure 2: Overview of class interfaces used from WAFER class

Note that the use of abstract interfaces rather than concrete classes keeps the implementation details away from the WAFER-STATE SERVER's core classes in a typical object-oriented way. This design allows future extension of some

of the capabilities like, e.g., supported file-formats, gridding algorithms or even of the internal data-structures the elements are stored on.

2.2 Interfaces

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 introduces two major advantages: On the one hand details about the underlying file format and about the gridding algorithms are well hidden to the WAFER-STATE SERVER's core functions. As already mentioned above this ensures that reading support for another file-format can be added later by implementing a READER module for this file-format. The same holds true for supporting various WRITER and GRIDDER modules. 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 module (e.g., quality constraints) can directly be accessed by the simulator without the need for extra wafer-state functions.

Currently READER modules for PIF, DFISE [5] and the newly developed WSS file formats as well as support modules for the GRIDDER DELINK [6] (three dimensional) and TRIANGLE [7] (two dimensional) are implemented. Note that the READER and WRITER interfaces are not restricted to file based data access. It is also conceivable to implement READER/WRITER modules that directly connect to a certain database engine.

READER Interface

Fig. 3 shows classes with their attributes and methods of the READER interface. All depicted classes are interface classes and are derived and implemented by a concrete class of the given file-format.

In order to read the file identified by a READER the WAFER-STATE SERVER first invokes the `next_segment` method. This will either return an instance of an implementation of a `RD_SEG` interface or indicate `end-of-file` in case no more segments are available.

The methods `next_attr` and `next_grid` of the `RD_SEG` interface allow iteration over all attributes and grids of a segment respectively. Note that grids are allowed both at segment and attribute level. These methods return an implementation of a `RD_GRID` or `RD_ATTR` interface class, respectively. Again `end-of-file` indicates the last grid or attribute of this segment was reached.

Finally, to iterate over all grid elements of a grid, the method `next_el` of the `RD_GRID` or `RD_ATTR` interface is invoked. Note that the implementation of the `next_el` method must take care to instantiate the points of the grid element as well as thereon stored quantities.

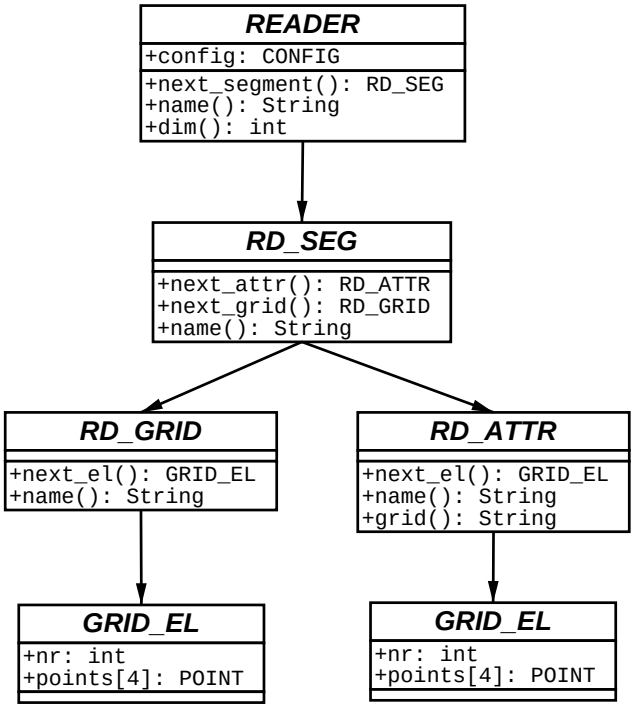


Figure 3: Attributes and methods of READER interface

The READER interface contains a so called `CONFIG` object. This object holds information about which interpolation method must be applied to a certain attribute, about what fundamental data-type (scalar, vector) an attribute has and also what attributes at all are supported. This object is needed to properly instantiate the points of a `GRID_EL` object.

WRITER Interface

In Fig. 4 the classes comprising the WRITER interface are shown. In a similar manner to the READER several interfaces are used to transfer data to a certain file.

First, the WAFER-STATE SERVER invokes the method `write_points` to indicate that a list of all points is following. The method returns an implementation of a `WR_POINTS` interface class. By using the method `next_point` all points are transferred to the WRITER module.

Next the segments and thereon stored data are created. For this purpose the method `write_segments` is called to retrieve an implementation of a `WR_SEGMENTS` class. This object is henceforth used to transfer grids and attributes to the file. The method `next_segment` of the `WR_SEGMENTS` object introduces a new segment. It returns an object of type `WR_SEG`. This object is used to de-

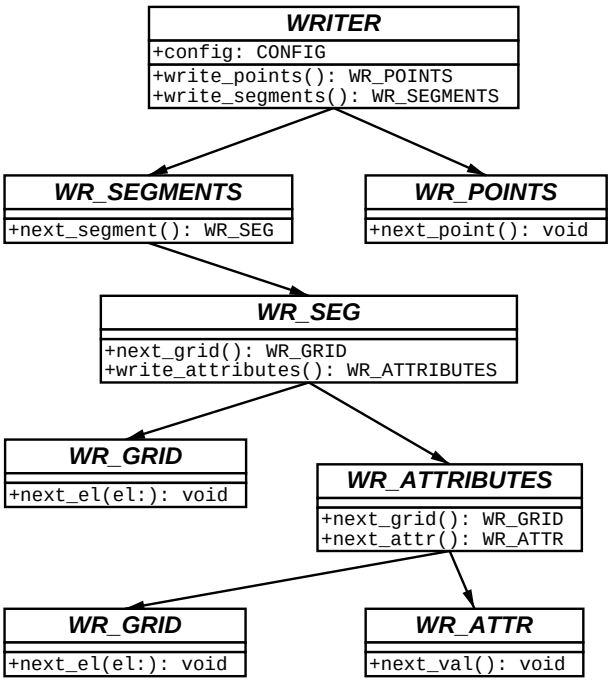


Figure 4: Attributes and methods of WRITER interface

fine (a) stand-alone grids (grids that are not used to store attributes on) and (b) distributed quantities. The stand-alone grids are written using the `next_grid` method. Attributes are stored via the `WR_ATTRIBUTES` object by several invocations of methods `next_grid` and `next_attr`. Note that several attributes can share one grid. The method `next_attr` receives the name of a prior defined grid as one of its arguments. This means that a grid comprising a distributed quantity has to be defined before the definition of the quantity referring to this grid. Each call to `next_grid` or `next_attr` returns an object of type `WR_GRID` and `WR_ATTR` respectively. The methods `next_el` and `next_val` must be used to add a grid element or a value of an attribute.

GRIDDER Interface

In Fig. 5 the classes comprising the GRIDDER interface are shown. A gridding mechanism is invoked by first defining the topography, second starting the actual gridding algorithm and, finally collecting the generated elements.

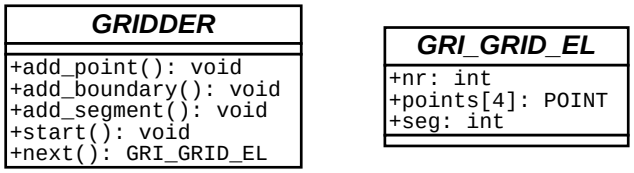


Figure 5: Attributes and methods of GRIDDER interface

The first step in the topography definition is to define all points of the geometry. The method `add_point` of the `GRIDDER` interface class must be invoked once for every point. Next, all elements forming the boundaries of the segments must be defined with the `add_boundary` method. Now the elements comprising the segments are defined using `add_segment`.

Once the topography has been defined the method `start` triggers the actual meshing process.

The elements are now ready to be collected. Method `next_el` is used to iterate over all generated elements. The method returns `false` after having delivered the last element.

It is worthwhile mentioning that the above interface is used to access two-dimensional as well as three-dimensional gridder modules.

2.3 PROTOCOL BETWEEN APPLICATION AND WAFER-STATE SERVER

The first step in the protocol (Fig. 6) from an application's point of view is to instantiate appropriate READER and GRIDDER objects. The choice of what READER module 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.

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 file (or they are passed on the commandline) 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 must be merged with the wafer-state. This so called update operation is performed in several individual steps. Each attribute which is configured as simulator output (c.f. Section 2.4) must now be stored back onto the wafer. The WAFER-STATE SERVER checks whether all attributes were received. Next the attributes which are invalidated by this process step are deleted.

Now the repairing mechanism is invoked. 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 module. All attributes which were not treated by the simulator and which are not configured to be invalidated are

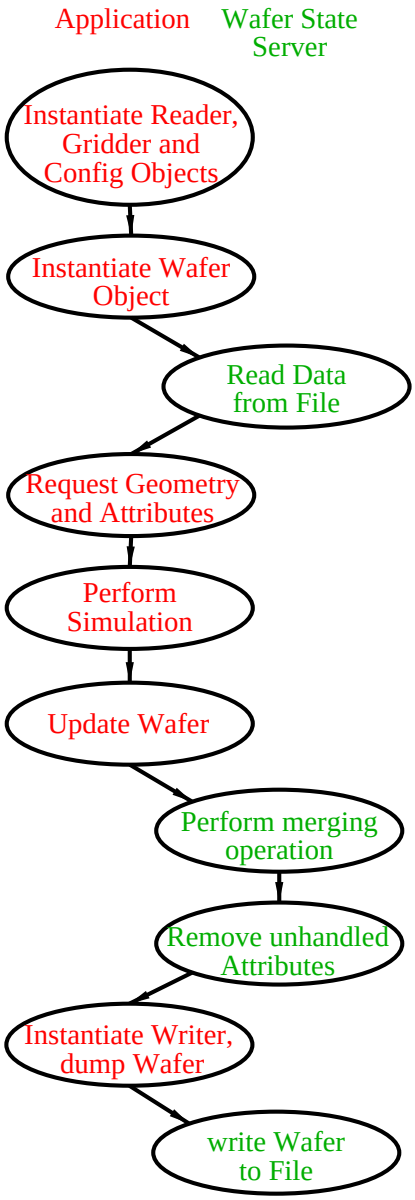


Figure 6: Basic protocol between wafer-state server and application

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 or database.

2.4 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 update them when writing back the results, or the WAFER-STATE SERVER has to remove 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 in the database.

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

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

    topography = false;
};
```

Figure 7: Configuration of a simple diffusion step

Fig. 7 shows a possible configuration of a simple 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. In this example the attribute named `Stress` 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. 8 depicts the configuration for attributes of type `Concentrations`.

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

Figure 8: 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 interpolation method (linear, logarithmic).

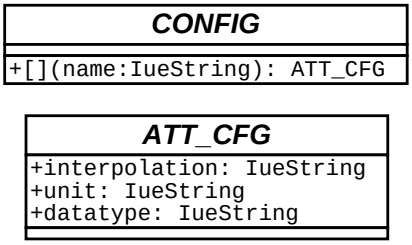


Figure 9: CONFIG object passed to READER and WRITER modules

The attribute type definitions are read into a `CONFIG` class (Fig. 9). An instance of such a class is passed to the READER and WRITER objects. The READER module needs the configuration to (a) identify a certain attribute by name and (b) to be able to properly instantiate the points of the grid elements. Since these points hold the actual attribute values the READER module is the only place where an instantiation can take place. The WRITER module uses the configuration to store attribute type information which is not explicitly contained in the WAFER-STATE SERVER data-structures (interpolation method, data-type) onto the file.

The actual configuration of a certain attribute is obtained via `operator[]` of the `CONFIG` class. The operator returns an `ATT_CFG` object of the named attribute or delivers an error if a configuration for an attribute with the given name does not exist.

3 TCAD ANALYSIS

Performing TCAD analysis tasks [8] like optimization of VLSI semiconductor devices [9, 10] or inverse modeling of doping profiles [11] often results in an enormous number of individual simulation runs. Frameworks like SIESTA or VISTA 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 a file as input which contains commands written in the input-deck language as it is also used in our device simulator MINIMOS-NT [12, 13], and in the configuration of the process steps and attribute types within the WAFER-STATE SERVER. Fig. 10 and Fig. 11 show the input-deck description to generate the three-dimensional heterostructure bipolar transistor device depicted in Fig. 12 and Fig. 13.

For each section in the input-deck file (`Bulk, Base, Emitter, Contacts,...`) a corresponding segment

```
Cube
{
   // coordinates are X/Y/Z
   points =
   [ [0.0, 0.0, 0.0], [1.0, 0.0, 0.0], [1.0, 1.0, 0.0],
     [0.0, 1.0, 0.0], [0.0, 0.0, 1.0], [1.0, 0.0, 1.0],
     [1.0, 1.0, 1.0], [0.0, 1.0, 1.0]
   ];

   // positive orientation of faces identifies the inner
   // region of the cube
   solid =
   [ [0, 1, 2, 3],    // bottom face
     [4, 7, 6, 5],    // top face
     [5, 6, 2, 1],    // front face
     [4, 0, 3, 7],    // back face
     [4, 5, 1, 0],    // left face
     [7, 3, 2, 6]     // right face
   ];

   Scaling { x = 1.0; y = 1.0; z = 1.0; };
   Offset  { x = 0.0; y = 0.0; z = 0.0; };
};
```

Figure 10: Description of a three-dimensional cube

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 [14] based on an algorithm of Kevin Weiler [15] 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 module (DELINK [6]) in order to generate a mesh of the whole structure. The final device is saved to a file using the WSS WRITER module.

4 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) [16]. 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 [17]) 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.

5 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 inte-

```
#include "cube3d.ipd"

Geometry
{
   Bulk:   ~Cube {  Scaling  { x = 4.0; y = 3.0;  }  };
   CollectorContact1: ~Cube
   {  Scaling { x = 3.0; y = 0.4; z = 0.3; }
      Offset  { x = 0.5; y = 2.0; z = 1.0; }
   };
   CollectorContact2: ~Cube
   {  Scaling { x = 0.4; y = 2.0; z = 0.3; }
      Offset  { x = 0.5;          z = 1.0; }
   };

   Base1: ~Cube
   {  Scaling { x = 2.0; y = 1.5; z = 0.3; }
      Offset  { x = 1.5;          z = 1.0; }
   };
   Base2: ~Cube
   {  Scaling { x = 2.0; y = 1.5; z = 0.3; }
      Offset  { x = 1.5;          z = 1.3; }
   };
   BaseContact: ~Cube
   {  Scaling { x = 0.8; y = 0.2; z = 0.3; }
      Offset  { x = 2.5;          z = 1.6; }
   };

   Emitter1: ~Cube
   {  Scaling { x = 1.2; y = 0.3; z = 0.3; }
      Offset  { x = 2.0; y = 1.0; z = 1.6; }
   };
   Emitter2: ~Cube
   {  Scaling { x = 0.3; y = 1.0; z = 0.3; }
      Offset  { x = 2.0;          z = 1.6; }
   };
   EmitterContact1: ~Cube
   {  Scaling { x = 1.2; y = 0.3; z = 0.3; }
      Offset  { x = 2.0; y = 1.0; z = 1.9; }
   };
   EmitterContact2: ~Cube
   {  Scaling { x = 0.3; y = 1.0; z = 0.3; }
      Offset  { x = 2.0;          z = 1.9; }
   };

   epsilon = 1e-5;
};
```

Figure 11: Description of a three-dimensional HBT device

gration of existing programs (DELINK, TRIANGLE, DFISE-READER) thus ensuring good overall code re-usability. During the implementation of the wafer-state server care was taken to adhere to the ANSI C++ standard as close as possible. The WAFER-STATE SERVER is running on all platforms that offer an ANSI C++ compiler. The code does not rely on any operating system dependent features.

6 CONCLUSION

We present a TCAD class library to support the simulation of semiconductor fabrication process flows. A solution to the problem of data exchange among various simulators is given. The presented READER, WRITER and GRIDDER interfaces allow for future incorporation of new file formats and gridding algorithms. Furthermore, the strong protocol all simulators must adhere to encourages the simulation of whole process flows. The WAFER-STATE SERVER offers the tool developer a powerful way of combining several independent modules. By appropriately instantiating the programmer has the flexibility of choosing among all available

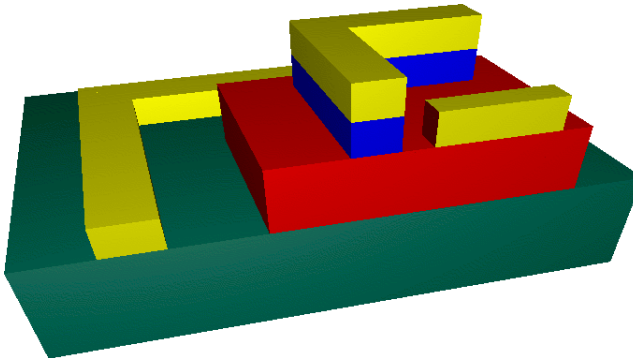


Figure 12: Three-dimensional HBT device

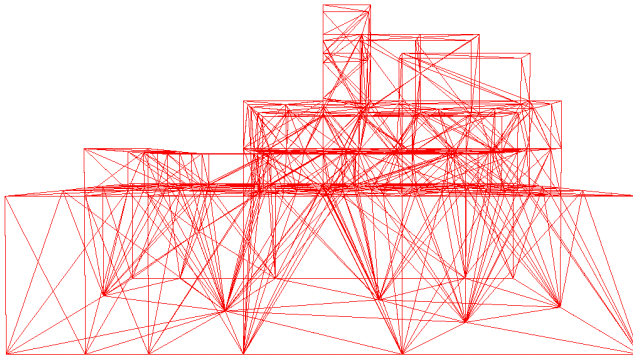


Figure 13: Mesh of the three-dimensional HBT device

implementations of GRIDDER, READER, and WRITER modules.

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. This is of particular interest to the TCAD analyst who would like to freely chose among all available simulators and tools.

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

7 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 [18, 19] as well as for the ETCHING simulator [20, 21] is under development.

References

[1] R. Strasser, Ch. Pichler, and S. Selberherr, "VISTA - a framework for technology CAD purposes," In Hahn and Lehmann [22], pp. 450–454.

[2] F. Fasching, C. Fischer, S. Selberherr, H. Stippel, W. Tuppa, and H. Read, "A PIF implementation for TCAD purposes," in *Simulation of Semiconductor Devices and Processes*, W. Fichtner and D. Aemmer, Eds., Konstanz, 1991, vol. 4, pp. 477–482, Hartung-Gorre.

[3] G. Massobrio and P. Antognetti, *Semiconductor Device Modeling with Spice*, McGraw-Hill, New York, second edition, 1993.

[4] T. Binder and S. Selberherr, "A parallel finite octtree for multi-threaded insert, delete, and search operations," in *Intl. Conf. Applied Modeling and Simulation*, Cairns, Australia, Sept. 1999, pp. 613–616.

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

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

[7] J. R. Shewchuk, "Triangle: Engineering a 2D quality mesh generator and delaunay triangulator," in *First Workshop on Applied Computational Geometry*. Association for Computing Machinery, 1996, pp. 124–133.

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

[9] R. Plasun, Ch. Pichler, T. Simlinger, and S. Selberherr, "Optimization tasks in technology CAD," In Hahn and Lehmann [22], pp. 445–449.

[10] R. Plasun, M. Stockinger, R. Strasser, and S. Selberherr, "Simulation based optimization environment and its application to semiconductor devices," in *Intl. Conf. on Applied Modelling and Simulation*, Honolulu, Hawaii, USA, Aug. 1998, pp. 313–316.

[11] R. Strasser, R. Plasun, and S. Selberherr, "Practical inverse modeling with SIESTA," in *Simulation of Semiconductor Processes and Devices*, Kyoto, Japan, Sept. 1999, pp. 91–94.

[12] T. Simlinger, H. Kosina, M. Rottinger, and S. Selberherr, "MINIMOS-NT: A generic simulator for complex semiconductor devices," in *25th European Solid State Device Research Conference*, H.C. de Graaff and H. van Kranenburg, Eds., Gif-sur-Yvette Cedex, France, 1995, pp. 83–86, Editions Frontieres.

[13] 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.

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

[15] Kevin Weiler, "Polygon comparision using a graph representation," in *Computer Graphics, 14*. SIGGRAPH 80, 1980, pp. 10–18.

[16] Will Schroeder, Ken Martin, and Bill Lorensen, *An Object-Oriented Approach To 3D Graphics*, Prentice Hall, 1999.

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

[18] A. Hössinger, S. Selberherr, M. Kimura, I. Nomachi, and S. Kusanagi, "Three-dimensional Monte-Carlo ion implantation simulation for molecular ions," in *Electrochemical Society Proceedings*, 1999, vol. 99-2, pp. 18–25.

[19] A. Hössinger and S. Selberherr, "Accurate three-dimensional simulation of damage caused by ion implantation," in *Proc. 2nd Intl. Conf. on Modeling and Simulation of Microsystems*, San Juan, Puerto Rico, USA, Apr. 1999, pp. 363–366.

[20] W. Pyka, R. Martins, and S. Selberherr, "Efficient algorithms for three-dimensional etching and deposition simulation," in *Simulation of Semiconductor Processes and Devices*, K. De Meyer and S. Biesemans, Eds., Leuven, Belgium, Sept. 1998, pp. 16–19, Springer.

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

[22] W. Hahn and A. Lehmann, Eds., *Proc. 9th European Simulation Symposium*, Passau, Germany, Oct. 1997. Society for Computer Simulation International.