Abstract—This paper describes the Semiconductor Wafer Representation (SWR) for representing and manipulating wafer state during process and device simulation. The goal of the SWR is to provide an object-oriented interface to a collection of functions designed for developing and integrating Technology Computer-Aided Design (TCAD) applications. By providing functions which can be common across many applications, we aim to greatly reduce tool development and integration time. Corporate, vendor, and university TCAD developers have worked together under the auspices of the CAD Framework Initiative to create an architecture and C++ programming interface for an SWR 1.0 draft standard. Here we will describe this architecture and the results of creating and using a prototype implementation of the standard both to integrate existing TCAD tools and to develop simple new tools.

I. INTRODUCTION

1.1. Background and Motivation

Technology computer-aided design (TCAD) is computer-aided design and engineering used in semiconductor device design, fabrication process design, technology characterization for circuit design, manufacturing yield optimization and process centering, and computer-integrated manufacturing. It includes detailed numerical simulation of fabrication process steps, fast simulation of complete fabrication processes, and detailed numerical simulation of the electrical performance of small (cell-level) circuits. TCAD is essential for the successful design of today’s devices and circuits because of their increasing complexity and the increasing cost and delay of experimental design iteration.

Practical TCAD problem solving usually requires the use of several different simulation programs. Since each program has its own, incompatible way of representing, reading, and writing design data, considerable time and effort can be wasted writing conversion routines between the various formats. Each program’s representations may not be complete, so auxiliary information must be saved elsewhere and merged into the design description again, when it is required if data fidelity is to be preserved. Since the simulators have incompatible user interfaces, accessing designs must typically be done at a low level, making it hard to automatically control simulators for statistical studies and design optimization. These problems are not unique to the TCAD area, and frameworks have been found to help solve them.

A framework is a set of interrelated services that provide infrastructure and support for computer-aided design system development and deployment [1]. The functions or services provided by a framework can be divided into sets of domain-dependent and domain-independent functions. The domain-independent services, for example, database and intertool communication, are generic in that their value does not depend on the framework’s application (e.g., electrical CAD or mechanical CAD). Other services, such as design data representation, are domain-dependent. Development of framework standards is being promoted by the CAD Framework Initiative (CFI), a consortium dedicated to defining interface standards for design automation tools and design data that benefit end users and vendors worldwide.

The physical wafer structure is only one part of the information that must be available in a TCAD framework. In addition, users may wish to describe manufacturing process parameters and sequences, lithography mask information from circuit layouts, and device characteristics for circuit simulators. These requirements are being addressed by other technical committees within CFI. The Semiconductor Process Representation (SPR) working group is creating an information model and programmatic interface for describing process step modules or sequences and the details of unit process steps [2]. One highly desirable use of the SPR is to support “simulation management” utilities which use process sequence and process step class information to chain the invocation of different simulators or simulation modules. It is also valuable to be able to reference process sequences from inside wafer state descriptions to “tag” materials with information about the process that generated them, as suggested by recent material property modeling efforts [3]. The Device Modeling and Verification (DMV) working group is developing standards for device characterization information and an information model (IM) for device representation. The goal of the DMV work, which uses the EXPRESS and EXPRESS-G languages [4], is to develop an IM that not only encompasses the data representation needs of the SWR, but also fulfills the modeling requirements of the VISTA system [5] and BPIF [6]. The value of an IM for electronic CAD applications has been demonstrated by Lau and Kahn [7].
This paper is about the emerging SWR standard and the lessons that were learned implementing a prototype server and clients for process simulation, device simulation, and model visualization.

1.2. Previous Work

Approaches to the general purpose representation of device structure and simulation information have followed a clear evolutionary path. The earliest approaches used file formats based closely on the internal data structures of individual tools. For example, the SUPREM-IV [8] file format has been used for data communications since it interfaces to PISCES-IIB [9], and can read/write information for use with SAMPLE and SIMPL [10]. Next came the definition of generic "standard" file formats. The Profile Interchange Format (PIF) standardizes on a tool-independent ASCII file format for the storage of device structure information [11]. The PIF format is very flexible, but this makes it difficult to develop parsers able to read all PIF files. In addition, the "intersite" PIF provides no help in the reading and writing of PIF information: each tool wishing to access PIF files must provide its own input/output routines.

The next significant evolutionary step was exploration of "intertool" versions of the PIF, focusing on programming interfaces to enable application programs to store and access shared wafer information. Formal object models, programming language bindings in C and LISP, and layering on object-like databases are explored in PIF/Gestalt [12], BPIF [13], and VISTA [5]. While each of these pushes programming interfaces and the use of PIF in different directions, each is very closely tied to the PIF object model and each strives for close compatibility with the PIF. While higher level functionality is considered, each is fundamentally constrained in implementation by low level representational details dictated by the PIF.

Apart from the PIF, other programming interfaces and databases for the storage and exchange of wafer information have been studied. The DAMSEIL system [14] defines a common Fortran data structure for 2-D geometries. The CMU Chip Database (CDB) [15] provides a database and programming interface optimized for brick-shaped geometries and interpolated 1-D doping profiles. While not constrained by the PIF, these systems are limited in terms of data and programming models.

The present work embodies the next significant evolutionary step in wafer representation: the definition and use of a high-level, object-oriented interface for the direct manipulation of wafer information by the tools. The SWR differs from previous work by (1) expressly divorcing itself from details of implementation algorithms and data structures (there is no requirement that low-level representations be PIF-like); (2) providing comparatively thorough 2-D geometry and field description mechanisms; and (3) providing powerful manipulative services beyond shared storage for use by application programs. The separation between interface and implementation details allows data structures to be improved over time to suit new algorithms without impacting application programs. The emphasis on high-level operations allows the consistency of the representation to be guaranteed. For example, a low-level operation to add an edge to a geometric object could lead to an inconsistent state (unconnected edge in the geometry) unless it were followed by low-level operations to properly connect the edge into the object structure. A high-level operation to intersect two objects results in a fully consistent final object state in a single (user level) step.

The question of inter-site data exchange is not directly addressed by SWR, and we have not defined any new external file format for this purpose. We view this as an important application-level problem where reader and writer applications communicate with SWR through the defined interface and communicate with each other through an external format of their choosing. This could be a private format or an existing format such as the ASCII PIF.

1.3. General Architecture

A Semiconductor Wafer Representation defines an object-oriented architecture and programmatic interface for describing device and wafer structures. The architecture of the SWR specifies the components of the wafer representation and the services available, which are the operations provided for the wafer components. We currently express this in terms of a C++ language programming interface which specifies the C++ objects and methods which may be used by TCAD applications. Since all interactions between the application and the wafer state are by way of this interface, there is a clear separation between the functionality of the SWR and the underlying representation. This separation is advantageous because it allows for multiple, different implementations of the interface. For example, one implementation may use a node and element table to represent a mesh, while another may use a graph-based data structure. Different implementation architectures are also possible. For example, one may choose to implement SWR as a server in a fully linked in with the TCAD application, while another may choose to provide an interface to a separate server process. We chose C++ as the first language for defining a programming interface because it provides object-oriented features without sacrificing efficiency, is widely available, and is highly standardized with strong typ checking to promote portability of code between machines. Programming interfaces for other languages could be defined in the future, but this has not been the focus of our work to date.

The SWR interface is conceptually divided into field and geometry components, which account for the two most common TCAD application views of the wafer. Capabilities, such as attributes, which are used by both components are contained in a utilities library. These views are explained in the example MOSFET device cross-section shown in Fig. 1. The geometry component describes the boundary information of the wafer state, which in this case consists of the heavy solid lines that outline the silicon, silicon dioxide, and gate materials. These outlines are described by piecewise linear segments, such as the ones connecting the vertices shown in the left silicon dioxide region of the MOSFET. Fields can be attached
A cross-sectional view of a MOSFET showing the division of the SWR into geometry, field, and attribute components.

Fig. 1.

The SWR architecture showing (1) how tools communicate through the SWR interface, (2) the partitioning of the SWR, and (3) direct communication between field and geometry components.

Fig. 2.

Implementation of the SWR, with example clients described in Section VI. Finally, conclusions based on our experience in developing and using the SWR are offered in Section VII.

II. GEOMETRY INTERFACE

The geometry component of the SWR allows for the representation and manipulation of (1) the shapes of the various material regions in the wafer and (2) the properties and interrelationships of these regions. In two dimensions these regions are two-dimensional point sets called faces, and their common interfaces are one and zero-dimensional point sets called edges and vertices. (In three dimensions the material regions are volumes and the interfaces are faces, edges and vertices respectively.) Vertices, edges, faces, and volumes are cells, and a collection of cells make up a cell complex.

To satisfy the requirements of the applications, the geometry interface supports several kinds of operations. First, it allows for the construction of geometric regions from various input information. The input information can be simple, such as the list of points describing a simple polygon, or it can be complex such as a complete list of faces, edges and vertices of a cell complex together with their interconnection. Second, the interface supports geometric queries, including point classification and queries of the interrelationships between the cells. For example, we can query for "the list of edges incident on vertex v, radially sorted around v." Third, the interface provides robust set point operations, such as union, intersection and subtraction. This is necessary, for example, to simulate an etch operation by removing (subtracting) a chunk of material.

Finally, there is a need to associate additional information with parts of the geometry. This is done using the attribution mechanisms of SWR (see Section IV). Attribution is used to associate fields with faces of the geometry. These attributes are updated according to the semantics of the set operations. In the above etch example, if a portion of a face f is removed from the geometry by a set-subtraction, and f had a field associated with it, then the remainder of f will have the field associated with it. The field itself is not automatically changed by such an operation.

2.1. Approach

Different TCAD applications require different perspectives on the geometry. For example, an application such as a diffusion modeler requires a representation of the wafer bulk, while a lithography modeler may need just surfaces. The geometry interface has to provide each application with the relevant perspective, while at the same time maintaining consistency between these two views. Hence the representation that we use is a cell complex, which is a collection of cells, or vertices, edges, and faces. A cell complex allows either of these differing perspectives to be used with equal facility.

The fundamental power of the geometry interface arises from this idea of a cell complex representation. Physically-based modelers frequently require properties such as boundary conditions to be associated with material interfaces, so existing geometric modelers used for semiconductor process simulation like OYSTER [17] or ECHIDNA [18] suffer because
interface information is not part of the geometric representation. To obtain the interface between two material regions, OYSTER copies each, grows both copies infinitesimally and then intersects them, obtaining a thin lamina if the two original regions abut. This is a time-consuming and possibly numerically unstable operation. ECHIDNA does not even have that capability—the geometry is obtained just by using set operations. These limitations are overcome by the cell complex approach used in the geometry interface. Just as the 2-D mesh representation in SUPREM-IV represents edges as the interface between two mesh elements, so does the geometry engine represent faces as the interface between two volumes. In this way it is possible to obtain material interface information directly from the representation. The engine also provides set operations that work directly on this cell-complex representation, and which are used to implement two-dimensional set operations by the SWR geometry server. A detailed description of the underlying geometry engine used in the prototype and its representation of cell complexes can be found in [19].

2.2. Capabilities

In this section we present some of the features of the geometry interface: construction of cell complexes, boundary and incidence queries, set operations, and attribute propagation. We will restrict our presentation to two-dimensional examples (the SWR prototype), although a one-dimensional version has been implemented and a three-dimensional version is nearly complete. The examples are presented to be faithful to the C++ programming interface of SWR, although we take notational shortcuts for convenience.

2.2.1. Construction of Cell Complexes: The simplest kind of two-dimensional cell-complex that might be constructed is a rectangle.

```c
SwrgPoint2D p(0, 0);
SwrgPoint2D q(1, 1);
SwrgGeometry2D cellComplex(p, q);
```

The constructed cell complex is an axis-aligned square in the xy-plane, with four vertices, four edges, and one face. Note that the "2D" suffix on the data types corresponds to the spatial dimension of the cell complex. We can also define the topological dimension of a cell independent of the spatial dimension in which it lives: 0 for vertices, 1 for edges, and 2 for faces.

If we give three points rather than two we obtain a triangle, and if we give a circularly-ordered point-sequence (see also Section 4.1), then we obtain a polygon. It is also very easy to construct a cell complex from a two-dimensional mesh m, as shown below:

```c
//given a two dimensional SwrgElementMesh "m"
SwrgGeometry2D cellComplex(m);
```

The algorithm used by the geometry subserver prototype to implement this example is fairly complicated, but instructive of the power of the geometry programming interface:

1. Use a boundary-following "iterator" supplied by the field interface to trace out the loops of the boundaries of the mesh;
2. Determine for each area which loops are outer ones, and which are inner ones;
3. Construct SwrgGeometry objects for each loop, and use set operations to subtract the areas corresponding to inner loops from the outer one.

Finally, it is possible to construct a cell complex piecemeal in a way similar to that allowed by PIF [11]. Essentially, construction is done by (1) naming each vertex and giving its coordinates; (2) naming each edge and giving its two vertex names; (3) naming each face; and (4) for each edge/face incidence, naming the edge and the face and giving the orientation of that incidence (±1).

2.2.2. Interrogating Cell Complexes: The cells of a cell complex can be examined in two ways: given a cell we can ask it for its boundary, and given a cell complex Q containing cell c of topological dimension d we can ask Q for the incident cells of larger topological dimension. For example, we can ask a cell complex for the edges incident to a vertex or the faces incident to an edge.

For both incidence and boundary queries the cells are often given as sequences. For example, the boundary of a face is a partially ordered sequence of loops (the outer loop comes first), and in two dimensions the edges incident to a vertex are radially sorted around the vertex on the xy-plane. In three dimensions the faces incident to an edge are radially sorted around the edge.

One common procedure that an application might need is to find an interface between two faces f and g. For example, f might have the material type oxide, and g nitride, and we could use the following code to obtain an edge of the oxide/nitride interface:

```c
SwrgEdge2D
interfaceEdge(const SwrgGeometry2D& Q,
const Face2D& f, const Face2D& g)
{
(1)SwrgEdgesOfFace2D edges = f.edges();
(2)while (edges.advance())
{
(3) SwrgEdge2D
  e = edges.current();
(4) SwrgFacesOfGeometry2D faces
    = Q.faces(e);
(5) if (faces.cnt() == 2 && faces[faces[0]
      == f ? 1: 0] == g)
      return e;

(6) return SwrgEdge2D::Null;
}
```

The above procedure is typical of the use of the geometry interface. The declarations in statements (1) and (4) are objects called iterators that examine sequences of cells in the geometry representation. In statement (1) we obtain the edges of a face and use that sequence to initialize an iterator. Statement (2) advances the iterator to the next edge, if possible, and statement (3) obtains the current edge. Statement (4) is
an incidence query, obtaining the faces incident to the edge, and statement (5) tests to see if the other incident face is \( g \).

The Boolean expression in statement (5) shows the use of the \( \text{cnt}() \) method, which returns the size of the sequence, and the subscript operator, which is defined to work for sequences. It is also worth noting that the equality operator used here is defined for cells of a cell complex. Finally, statement (6) returns a special null value if no edge on the interface exists.

This example demonstrates the intent of the geometry interface design to make the geometry representation as easy to use as possible. Procedure \( \text{interfaceEdge}() \) does not require any storage management, operations like subscript and equality are defined, and access to collections of objects using all of the different kinds of provided iterators is uniform.

2.2.3. Set Operations: Aside from the Boolean set operations, there are some simpler kinds of set operations provided.

The simplest is point \( \text{classification} \). Given a point \( \text{SwrgPoint2D} \), there is a procedure that determines which \( k \)-cell of the cell complex intersects the point in its interior. There is also a procedure that takes a cross-section of a \( \text{SwrgGeometry2D} \), obtaining a \( \text{SwrgGeometry1D} \).

There are a variety of set operations defined for cell complexes, shown in Fig. 3. For example, we can \( \text{inset} \) a cell complex \( C \) into a cell complex \( D \). All the old points which existed in \( D \) are replaced with the points of \( C \), keeping the attributes of the \( C \) points, and discarding those of the \( D \) points. A second operation is the intersection of a cell complex \( C \) with a cell complex \( D \). This operation removes from \( C \) all points not contained in \( D \). Similarly, we can \( \text{subtract} \) \( D \) from \( C \) which removes from \( C \) all points contained in the interior of \( D \). The \( \text{merge} \) operation takes two cell-complexes with a single \( d \)-cell each, and gives a cell complex containing the cell(s) that is their union. Finally, the \( \text{glue} \) operation takes two non-overlapping cell-complexes and produces a cell-complex containing all of their \( d \)-cells.

![Fig. 3. The common set operations.](image)

3.1. Approach

The field architecture specifies both the objects that are available (e.g., nodes, elements, fields) and the operations on them (e.g., create node, evaluate field). Applications use the field interface in two ways. Some wish to work directly with the detailed structure of the mesh. Low-level mesh operations are provided which allow full control over the detailed specification of the objects. These operations are specific to the individual mesh types, so the application must be aware of the differences between, for example, a triangular element mesh and a tensor product mesh. Other applications wish to use fields without knowing the details of the field representation. High-level functions are provided that allow operations such as evaluation to be performed independent of the underlying field representation. In the future, we expect an increasing number of high level functions to be made available to reduce the amount of work being done by applications, as is already the case for the geometry interface. Agreement on such functionality and definition of the interfaces is part of the continuing work of the CFI SWR working group.

The components of a meshed field in SWR 1.0 are shown in Fig. 4 using EXPRESS-G notation, where the bold lines indicate derivation and the thin lines indicate aggregation. A \( \text{FieldOnMesh} \) object is composed of three pieces: a \( \text{Mesh} \) object, a \( \text{Shape} \) object, and a \( \text{MeshData} \) object. The \( \text{Mesh} \) object describes the mesh itself according to one of the choices of mesh type shown. The \( \text{Shape} \) object describes how the \( \text{MeshData} \) values are to be interpolated across mesh elements, and is used when evaluating fields or interpolating between them. A \( \text{Mesh} \) object is either an \( \text{ElementMesh} \) or a \( \text{TensorProductMesh} \). The common derivation indicates that there are some operations common to both mesh types, such as the ability to query the nodes. Similarly, the \( \text{Element} \) object collects together operations common to all element types. The element types shown here are available in SWR 1.0, and were the ones present in the prototype implementation. A wider variety of prebuilt element shapes, the ability to construct arbitrary elements hierarchically from volumes, faces, edges, and nodes, and mixed element grids are planned for subsequent versions of the specification.
3.2. Capabilities

In this section we will describe some of the operations that are provided by the field interface. The construction and querying operations are mainly low-level operations, in that they are specific to the underlying representation type and allow detailed manipulation of the representation objects. We will also describe some high-level operations which provide more complex and powerful capabilities. As before, the examples will use the C++ programming language interface and focus on two-dimensional examples. Further details about all of the objects and operations can be found in the Procedural Interface document [20].

3.2.1. Low-Level Construction: The topmost object inside the field representation is called a SwrfFieldSet, which collects together related mesh and field information. All field objects associated with a single cell complex in the geometry representation are usually part of the same SwrfFieldSet. Once a SwrfFieldSet object has been created or retrieved, it becomes the currently selected SwrfFieldSet and all new field objects are added to it.

Construction of a mesh begins with creating the coordinate objects where mesh nodes will lie. SwrfCoordinate objects provide a link between nodes at the same place in space which are not part of the same mesh, avoiding the need for floating-point comparisons of coordinates. Element meshes are built by creating an empty mesh and adding nodes and elements to it. A SwrfFieldOnMesh can be constructed by supplying the mesh as an argument, then setting the desired shape function and solution values. Fields are parameterized by value type, which may be Int, Real, or Complex. SWR 1.0 supports scalar fields with linear or logarithmic nodal shape functions. Edge-based and element-based shape functions have also been prototyped and will be supported in the future, as will vector and tensor field value types. Tensor product meshes are constructed by describing the grid lines which lie parallel to the coordinate axes. The attribute tag list mechanism described in Section IV can be used to associate other information with SwrfFieldSets and SwrfFieldOnMeshes. In addition, these objects can themselves serve as values in an attribute tag name-value pair. The value can be retrieved by a later application and used to construct a reference to that field object.

3.2.2. Low Level Queries: Each object in the representation provides functions to query each of its components. As with the geometry interface, when a sequence of objects is to be returned an iterator is provided. For example, to print out all of the node solution values from a meshed field, an application would use the following code:

```cpp
SwrfMeshNodeIter nodes = fom.mesh().nodes();
while (nodes.advance())
    cout << fom.getvalue(nodes.current());
```

Since fom is a SwrfFieldOnMesh <Real>, the function getvalue returns a Real number.

Fields can be evaluated at any point in space, and always evaluate to zero outside their domain. This operation makes use of the shape function to interpolate across elements of the mesh.

```cpp
Real s = fom.evaluate(0.25, 0.6);
```

3.2.3. High Level Operations: High level operations make the field interface easier to use by combining many low level operations into a single service. For mesh generation, SWR 1.0 provides a simple interface for element meshes which allows the application to specify the required local mesh spacing as a Real input field.

```cpp
// Assume the existence of a spacing field called 'hints'
// and a SwrgGeometry2D called 'wafer'
SwrfElementMesh m(TRIANGLE, wafer, hints);
```

Uniformly spaced tensor product meshes can also be created with a single function call

```cpp
// dim is the mesh dimension
// nmeshx, nmeshy are the number of mesh
// dx, dy are the mesh line spacings
// orgx, orgy is the mesh origin
SwrfTensorProductMesh t(dim, nmeshx, nmeshy, dx, dy, orgx, orgy);
```

These interfaces to mesh generation are still quite primitive, and more powerful capabilities are being investigated for later versions of the SWR. These will be expressed as additional constructors for meshes using different arguments.

Interpolation between meshes is possible independent of the representations used for the meshes since evaluation is possible at any point in the domain through the shape function. A new field which contains an existing field interpolated into a new mesh can be constructed as follows:
// fom is a SwrfFieldOnMesh <Real> based on an element mesh
// t is a SwrfTensorProductMesh
SwrfFieldOnMesh <Real> new_fom = fom.evaluate(t);

Here, new_fom is tensor product based using mesh t and has the same shape and attribute information as fom.

IV. UTILITIES

A set of utility methods common to both the geometry and field components and to TCAD applications has been provided. These utilities include an attribute library, iterators, and sequences.

4.1. Attributes

Attributes are name-value pairs attached to geometry or field objects. Attributes serve two purposes. First, they provide a link between geometry and field objects, i.e., a geometry can reference a field with an attribute since a field is an attribute value type. Second, attributes provide a means for clients to attach semantics to geometry and fields (e.g., a geometry has material type polysilicon, a field is the electric potential). Attributes are interpreted by applications, and so provide a general means for representing information about the wafer state that is not specifically included in the current SWR standard. The value field can contain arbitrary data specified and evaluated by application-supplied methods.

Applications make attributes known to the SWR by adding them to the attribute dictionary. Methods are provided for creating, modifying, and examining attribute lists, and for associating these lists with geometry and field objects. The lists are propagated by the geometry interface according to set operations in a very simple way. If, during a set operation, a cell of one complex intersects with another cell complex to form a cell of the result, then the attribute lists of the two input cells are concatenated. Applications are free to modify the resulting attribute lists if this default action is not sufficient.

The following example shows the use of attributes in a procedure which plots all Real fields associated with a geometry face:

```c++
void plotFieldsOnFace(const SwrgFace<2>& face)
{
    (1) const SwruTag & fieldname =
        SwruTagDict::find("SwrfField <Real>");
    (2) SwruLabels labels = face.labels();
    (3) while (labels.advance() & labels.current().name() == fieldname)
        ;
    (4) SwrgValuesOfLabel values = labels.current().values();
    (5) while (values.advance())
        {
            (6) SwrfField<Real> field(values.current());
            (7) plotField(field);
        }
}
```

Statement (1) examines the attribute dictionary to determine the internal name for the SwrfField<Real> attribute. Statement (2) constructs an iterator which allows all of the attributes associated with the face to be examined, and statement (3) selects the attribute holding the desired fields. Since this attribute may hold many fields, a second iterator is used to step through them (statements (4) and (5)). When the attribute value is passed to the SwrfField<Real> constructor in statement (6), an object representing that field is created. Finally, statement (7) passes that object to another procedure to plot the field.

In future versions of the standard, as we gain more experience with SWR attributes, we will include commonly-used attributes in an attribute taxonomy and more powerful ways of controlling their propagation during SWR operations. This will reduce the need for application code to manipulate them, and enforce consistent usage across applications. The attribute specification provides for a future attribute type hierarchy.

4.2. Sequences and Iterators

Sequences are used to specify a list of objects to the SWR interface. An example would be to construct a face out of a circularly-ordered sequence of points. Unordered, ordered, and circularly-ordered sequences are provided. Methods are provided for creating, manipulating, and querying sequences. Iterators are methods that access sequence elements in order.

V. PROTOTYPING

To investigate the strengths and weaknesses of our approach, a subset of the SWR has been prototyped using C++ [21], [22]. In a demonstration of the prototype, eight different client TCAD programs used an SWR server in a scenario involving process and device simulation of a MOSFET [23], [24]. Most client programs were existing process simulators which were enclosed in software "wrappers" to translate between each simulator's external file format and the SWR standard interface. Both topography simulators such as SAMPLE [25] and SPEEDIE [26], and bulk simulators such as SUPREM-IV [8] and MINIMOS [27] were included. Simple client programs for deposition and stripping were written to directly manipulate SWR objects without maintaining their own internal data structures.

This section describes the implementation of the prototype SWR server. Our experience with client applications is described in Section VI.

5.1. Prototype Architecture

The prototype architecture is shown in Fig. 5, and is a multi-process client-server implementation based on a similar approach used earlier by the IBM geometry engine [19]. The subservers for geometry and fields run as separate processes which remain alive while individual client applications appear, perform their operations, and exit. The utility library is linked in with both the server and the clients. Using the SWR interface, clients establish socket connections with the server and map areas of shared memory containing the server objects. Client object queries run in the client process space by using
server-supplied functions to examine the shared memory data structures. Client object creation and modification requests are passed across the socket to the server for execution there. Since the client does not have write permission to the shared memory areas, applications cannot accidentally write over any of the server data structures.

The mechanisms outlined here are invisible to application programs because they are contained within the classes and member functions that the server supplies to the client. We could replace this implementation with one which operates entirely as a library linked in with the clients provided only that it conforms to the same interface specification. This ability to hide implementation details is a clear advantage of an object-oriented approach.

5.2. Prototype Performance

Server execution times were measured for both the geometry subserver and the field subserver to determine the performance of the prototype.

To assess the performance of the geometry subserver prototype we looked at two operations: subtraction, which is a "server-side" operation, and point location, which is a "client-side" operation. Subtracting one SwrgGeometry2D from another is used to incorporate the results of etch simulations into the global SWR wafer state. Point location is used in the SWR prototype visualizer to determine which fields are defined for an input point. Table I shows how long it takes to:

1. Subtract one n-faceted approximation to a circle from another one.
2. Use point classification to locate 100 points in the interior of an n-faceted approximation to a circle.

Execution times (in seconds) were measured on an IBM RS/6000 model 530 workstation.

The prototyped geometry subserver uses a three-dimensional modeler so the circle subtractions are actually implemented by subtracting cylinders. An n-faceted approximation to a circle is represented as a cylinder with n + 2 faces, 3n edges, and 3n faces. Point location shows sub-linear time for small and medium structures—the computation time grows sub-linearly with the number of facets in the circle approximation. This shows one of the reasons for using a complex representation like the SWR: efficient and correct implementations of complicated operations are provided as part of the representation. Because the prototype uses a three-dimensional geometric modeler, the underlying cell-complex representations are quite large, and system overhead, such as paging begins to be significant as the models become large.

To assess the performance of the field subserver prototype, timing comparisons were made between code using the server and the same functions written directly in C++. Table II shows execution times in seconds for simple operations on 20 000 coordinates running on a SPARCstation II. Although the prototype was not written with high speed as a main objective, some optimization of the coordinate code was performed allowing a realistic performance estimate to be made. For creation operations, we found that the prototype field subserver ran 10 times slower than direct code. The overhead can be attributed primarily to communications costs in our multiprocess client-server implementation. Query operations ran 2.6 times slower because of the cost of traversing the class hierarchy which hides implementation details from the application. In real applications, query overhead should be insignificant compared to the cost of the simulation algorithms themselves. For example, Table II shows that just the cost of calculating a square root dominates the server overhead time.

Existing techniques for tool integration make use of a common data file format [11], and therefore create files to pass information between tools. Table II shows a comparison between creating and examining coordinates using the server and writing and reading the same coordinate information using an ascii file. The server implementation offers substantial improvements in performance in transferring data, and even greater savings are possible in practice since the server can be selectively interrogated, whereas the entire file must be read in a file based system.

VI. CLIENT APPLICATIONS

This section describes our experiences in using the prototype SWR server for TCAD tool integration and tool development. The purpose of this effort was to verify that the proposed architecture and programming interface were sufficient for tool integration purposes, to discover areas where improvements could be made, and to determine the impact of our client-server implementation on system performance.

---

**TABLE I**

<table>
<thead>
<tr>
<th>Number of edges in circles</th>
<th>Point Location</th>
<th>Subtraction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(total)</td>
<td>(cpu)</td>
</tr>
<tr>
<td>8</td>
<td>.07</td>
<td>.12</td>
</tr>
<tr>
<td>32</td>
<td>.14</td>
<td>.56</td>
</tr>
<tr>
<td>128</td>
<td>.39</td>
<td>3.92</td>
</tr>
<tr>
<td>512</td>
<td>1.33</td>
<td>43.25</td>
</tr>
<tr>
<td>2048</td>
<td>6.19</td>
<td>768.87</td>
</tr>
</tbody>
</table>

**TABLE II**

<table>
<thead>
<tr>
<th>Prototype Performance for 20 000 Coordinates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td>Create 2-D coord.</td>
</tr>
<tr>
<td>Examine x location</td>
</tr>
<tr>
<td>Calculate $\sqrt{x^2 + y^2}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Server</th>
<th>File</th>
<th>Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create ascii file</td>
<td>1.99</td>
<td>36.9</td>
</tr>
<tr>
<td>Read ascii file</td>
<td>0.07</td>
<td>5.35</td>
</tr>
</tbody>
</table>
Client applications can be categorized as either 'wrapped' or 'SWR direct.' Wrapped tools are programs which are left unchanged, but are enclosed in a software wrapper which communicates with the SWR server on tool startup and shutdown to convert between the SWR representation and the tool’s representation. The value provided here by SWR is the integration of tools together through the common SWR representation. SWR-direct tools are programs which are created or modified to use the SWR server as their primary representation for the wafer cross-section. The value provided here is to make new program creation easier and more robust, with the additional benefit of allowing integration of the resulting program with other SWR-compatible tools. Wrapped tools impose less performance requirements on the server and allow the integration of existing tools without modification but require a general purpose conversion between representations, which is not always an easy task. SWR-direct tools require no representation conversion steps but do make more use of the server within their simulation algorithm and so impose stricter requirements on server performance.

The following sections describe some of the clients that have been implemented. This includes wrapped clients with both a field-based viewpoint (SUPREM-IV) and a geometry-based viewpoint (SAMPLE), an SWR direct client for poisson equation solution, and an integrated collection of clients for hybrid simulation of plasma processing.

6.1. Wrapped SUPREM-IV Client

Since the SWR representation has much in common with the SUPREM-IV internal representation, it was straightforward to produce a wrapper that translated the data fields and mesh both in to and out of the SWR representation. To convert from SUPREM-IV to SWR, low-level SWR operations were used to create coordinate, mesh, and field objects corresponding to the information in SUPREM-IV. Since SUPREM-IV does not have a separate geometry representation, the geometry sub-server was used to create SWRGWafer2D geometry objects for each material layer directly from the mesh information already loaded into the server using a single high-level construction operation. These geometric pieces were assembled using the glue operation, and the attribution mechanism used to associate fields with their corresponding geometry. Converting back from SWR to SUPREM-IV similarly required the use of low-level operations to query the current coordinate, mesh, and field objects in the server. Reading representations that may have been modified by geometry based tools produced two problems.

The first problem was in identifying and building the exposed surface. SUPREM-IV stores the surface by placing a gas node at each exposed coordinate. Those edges that exist on the exposed surfaces needed to be identified and then appropriate gas nodes and flags needed to be built from this information. This was not difficult to perform, as the necessary access functions were provided by the SWR interface.

The second problem is more difficult to address appropriately. When geometry based codes modify existing boundaries or add new pieces, the new region shapes must be retrian-gulated for use in SUPREM-IV. Retriangulating an arbitrary region is a difficult task, especially when considering that internal data fields must be represented accurately and that a minimum of points should be used to keep computation time reasonable during subsequent simulation steps. Although the SWR 1.0 interface provides access to a triangulation function, this may not be ideal for each case. Providing an accurate triangulation proved to be the most challenging task in reading data into SUPREM-IV, and this is an area for continuing study.

6.2. Wrapped SAMPLE Client

In general, a wrapped SAMPLE client has three components: Input deck generation, input structure generation, and output string incorporation. Input deck generation involves translating process parameters obtained from the SPR Server [28] into the appropriate SAMPLE simulation parameters. During input structure generation, the current SWR state is examined to extract the information needed by SAMPLE. SAMPLE calculates its result as a new surface string which is read and used to update the wafer geometry. As described below, the tasks of input structure generation and output string incorporation are greatly simplified by composing cell complex queries and set operations defined in the SWR.

Depending of the process being simulated, input structure generation may involve either sectioning or invoking cell complex queries. Since SAMPLE assumes the resist layer and underlying topography are planar, the prototype wrapped SAMPLE lithography client only required taking a 1-D section of the 2-D cross section. This is a single high-level operation provided by the geometry interface. On the other hand, for etching and deposition, cell complex queries have been composed to traverse the geometric boundaries generating a continuous string of points which bound material layers of the wafer.

Output string incorporation exploits set operations provided by the SWR geometry interface. For lithography and deposition, the output string and the original wafer top surface are stitched into one or more non-self-intersecting polygons. These polygons are added to the wafer geometry using the inset operation. In the case of etching, the output string and top corners of the wafer bounding box are stitched into a clipping polygon. The clipping polygon is subtracted from the original wafer to generate the etched wafer. In either case, the use of high-level geometry operations allows a robust implementation of the wafer modification effects to be quickly made.

6.3. SWR-Direct Poisson Equation Solution

While the SUPREM-IV wrapper example shows the use of the field subserver for tool integration, the ultimate goal of the field subserver is to support physical model and simulator algorithm development. Common modules in numerical simulators are equation assembly and matrix solution. Placing matrix information in the field subserver can reduce data traffic between the client and server during the assembly phase. The global stiffness matrix representing the discretized set of equations can be directly updated from a client without the
use of local data structures with proper use of fields within
the subserver and arithmetic operations.

To determine the required methods to support matrix as-
semble and to investigate the overhead of the field subserver
on the matrix assembly phase, a prototype Poisson equation
solver has been developed using the following algorithm:

```
initialize
symbolic factorization
initial guess
do {
    assemble the matrix
    factor the matrix
    solve
    update solution
} while (not converged)
```

The Poisson equation was discretized using the finite vol-
ume discretization over a triangular mesh. A non-linear it-
eration scheme is required to solve the resulting equations
since the carrier concentrations are functions of the potential.
Typically, global matrix elements are filled in element-by-
element. Values that need to be computed are the length
of the triangle edges, the length of the perpendicular edge
bisectors, the Voronoi area associated with a point, and the
carrier concentrations. Methods invoked during the assembly
phase are traversal of an iterator over all of the elements
and a traversal over the nodes of each element. Calculation
of this equation matrix provides an upper bound on the field
subserver overhead because matrix entries for more physically-
based models will be more complicated functions of mesh
connectivity and solution variables.

Table III shows the percentage of CPU time used by routines
in the Poisson solver for a 945 node pn-diode. Twenty-seven
non-linear iterations are required for convergence. From the
table, over fifty percent of the total run time is spent factorizing
the matrix and less than twenty percent of the total time is
spent for assembling the matrix. The matrix solution time is
a very small portion of the total simulation time. Table III
also contains a breakdown of matrix assembly time to show
the percentage due to field server assembly access functions.
The field server overhead is less than five percent of the total
CPU time. Solution values, coupling coefficient information,
matrix representation, and intermediate solution information
are stored locally instead of updating the field subserver,
reducing the data traffic to the subserver. These results show
that the overhead in extracting connectivity information during
the assembly phase from the field subserver is not a dominant
factor.

6.4. Application of SWR in Hybrid Simulation

Integration of process simulators with empirically derived
equipment models has been traditionally difficult. The models
used in simulators and the Response Surface Models (RSM's)
used in empirical equipment modeling do not span the same
domains. The equipment models are explicit mappings from
the equipment controls to the change in wafer effects, while
the process simulators describe the change in wafers effects
in terms of the treatments on the wafer surface (e.g., wafer
temperature profiles, partial pressures, etc.). Moreover, wafer
representations and process flow representations for both these
model forms have been vastly different. Users are generally
unable or unwilling to use both forms of models in the
same environment. The SWR server prototype made possible
the implementation and demonstration of a hybrid simulation
system which merges the physically based modeling used in
traditional process simulators and the RSM's used in empirical
equipment modeling.

Equipment RSM (polynomial) models of TI Advanced Vac-
um Pressure Processor (AVP) operation for plasma assisted deposition
and etching were developed through designed experimentation.
These AVP models describe the change in wafer state as
a function of equipment control settings (e.g., pressure, RF
power). For polysilicon deposition the outputs of interest were
the deposition rate, impurity concentration and spatial nonuni-
formity. For polysilicon etch the characteristics of interest
were etch rate and etched profile shape parameters. Simulation
modules using these equipment models were implemented in
C++ using calls to the SWR server to model the effect on
the wafer, and using calls to the SPR server to get detailed
process setting information.

The SWR provides methods that make implementation and
integration of equipment models comparatively straightforward.
The SWR server made the implementation of AVP clients into the hybrid simulation environment to be simple and efficient.
The SWR provided a representation of the wafer structure that can be modified by the AVP clients using
calls to the SWR methods. High level queries were used to interrogate the prior wafer state, which was then altered
based on the equipment settings using SWR set-operations.

Negligible knowledge of the details of the wafer state rep-

TABLE III

<table>
<thead>
<tr>
<th>Routine</th>
<th>Percentage</th>
<th>Routine</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix Factorization</td>
<td>55.8</td>
<td>Computation of terms</td>
<td>6.30</td>
</tr>
<tr>
<td>Matrix Assembly</td>
<td>18.1</td>
<td>Field server access</td>
<td>4.65</td>
</tr>
<tr>
<td>Symbolic Factorization</td>
<td>14.7</td>
<td>Loading element stiffness matrix</td>
<td>4.00</td>
</tr>
<tr>
<td>Solve</td>
<td>5.6</td>
<td>Miscellaneous</td>
<td>3.15</td>
</tr>
<tr>
<td>Miscellaneous</td>
<td>5.8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
presentation within the SWR server was needed to integrate the clients. Moreover, the integrator was relieved of having to know about the internal wafer representations, and the native input languages, used by the other simulation tools within the system. Given a wafer state in the SWR, we were able to simulate later (or earlier) process steps stored in the SPR.

A hybrid simulation of a simple MOS transistor flow (as described in [23]) was performed in accordance with steps stored in the SPR. The resultant change of the wafer topography can be examined using the visualization client supplied with the SWR prototype. Fig. 6 depicts the wafer geometry after the plasma assisted polysilicon gate etch. With manufacturing models merged into the simulation system via the SWR, one is able to obtain very close matching to experimental features such as polysilicon sidewall slopes.

VII. CONCLUSION

Our success in creating a working prototype system has demonstrated that our architecture is capable of being used for tool integration. Prototyping has also shown several areas where more research is necessary before the promise of a powerful integration and development framework can be fully achieved.

The object-oriented architecture and programming interface describes interactions while retaining the freedom to use a variety of implementation approaches. The prototype used a client-server implementation with multiple processes and shared memory which was read-only from the client side, allowing client queries to run directly in the client process but forcing modification requests to be passed across a network socket to the server. This yielded acceptable performance for query operations, but added some overhead to creation and modification operations. Alternative implementations are now being investigated, but we conclude that our architecture can be implemented efficiently.

The division of the SWR into geometry and field components is an approach to managing the complexity of the problem rather than a fundamental separation. Geometry and fields are two views of the same structure. Nevertheless, particular applications tend to emphasize one view over the other and the SWR components must work together to maintain a consistent representation of the structure. The prototype implementation had almost no coupling between the subservers. At a minimum, a development framework requires new SWR-level functions which modify geometry and fields together in a consistent way. A better approach may require common coordinate objects to avoid roundoff problems, and two-way links to allow changes in one subserver to be propagated to the other. This is particularly important to be able to address moving boundary problems.

A framework valuable for developing new tools requires more than low-level modification and query operations. It serves as a repository for operations required by multiple tools, allowing new tools to be assembled more easily from existing operations. Two common operations required by bulk simulation tools are mesh generation and partial differential equation (PDE) solution. Mesh generation and adaptation in the initial architecture followed the view that mesh control is very problem-specific, and so placed much of the burden on individual applications. Prototyping indicated that this is too much responsibility, and that the SWR should be capable of performing reasonable default actions and/or offer high-level functions to take care of most cases. A general solution will require the specification of an internal interface between the field component and a meshing module defining how meshing constraints are communicated and how meshing history is recorded to allow intelligent remeshing at a later time. Similarly, the need to solve PDEs is common to a number of process and device modeling problems and so would be a valuable service in a development framework, although this was beyond the scope of the first prototype. As with meshing, we should provide an internal interface through the field component to the best of the existing PDE solution packages. This approach has at least two advantages over accessing PDE packages directly from the application. Since the PDE package is more closely coupled with the SWR, it may be able to take better advantage of low level server methods to increase speed. Since the application interface is well defined, improved solution packages could be investigated with minimal application changes. A significant challenge in this area is to effectively describe the model equations and coefficients from the application side without introducing unacceptable overhead. In both cases, any particular application is free to choose to use or ignore any particular service depending on its suitability for that application's needs.

All of the research topics described above need to be guided by a clear understanding of the requirements and constraints of client applications. Prototyping gave much valuable information about real simulation clients. Continuing development of the architecture and implementation will continue to include close ties to client development for 2-D and 3-D applications. A 3-D framework will also be useful for physically-based equipment modeling, although that will require extensions to the data model.

ACKNOWLEDGMENT

The work described here has benefited from the contributions of all of the members of the CFI TCAD TSC, the SWR Working Group, and the participants in the 1991 TCAD demonstrations.
REFERENCES


Martin D. Giles (S'81-M'85) received the B.A. and M.A. degrees in Natural Sciences from Cambridge University, England in 1981 and 1985, and the M.S.E.E. and Ph.D. degrees from Stanford University, Stanford, CA, in 1983 and 1984 respectively.

In 1984 he joined AT&T Bell Laboratories in Murray Hill, NJ as a member of the VLSI device analysis and simulation group. There he developed and applied advanced process models for submicron silicon technology including defect-related diffusion, ion implantation, and sputter deposition. In 1990 he joined the EEC Department at the University of Michigan, continuing his research interests in the modeling of processing technology and process equipment, and the development of frameworks for technology computer-aided design.

Dr. Giles is a member of the Electrochemical Society.

Duane S. Boning (S'83-M'91) received the S.B. degrees in electrical engineering and in computer science in 1984, and the S.M. and Ph.D. degrees in electrical engineering in 1986 and 1991, respectively, all from the Massachusetts Institute of Technology, Cambridge, MA.

He was an NSF Fellow from 1984 to 1989, and an Intel Graduate Fellow in 1990. From 1991 to 1993 he was a Member Technical Staff at the Texas Instruments Semiconductor Process and Design Center, Dallas, TX, where he worked on semiconductor process representation, process/device simulation, tool integration, and statistical modeling and optimization. He is currently an Assistant Professor in the Electrical Engineering and Computer Science Department at MIT. His research interests include process control, process synthesis, tools and frameworks for semiconductor process and device design, and the computer integrated manufacturing of integrated circuits.

Dr. Boning has served as chairman of the EDIF Process/Device Technical Subcommittee, and as chairman of the CFI/Technology CAD Framework Semiconductor Process Representation Working Group. He is a member of Eta Kappa Nu, Tau Beta Pi, Sigma Xi, and the Association of Computing Machinery.

Goodwin Chiu (S'84-M'85-S'86-M'93) received the B.S. degree in electrical engineering and computer science and materials science and engineering from the University of California at Berkeley in 1983, the M.S. degree in materials science from Stanford University in 1984 and the Ph.D. degree in electrical engineering from Stanford University in 1992.

He is currently working at the IBM Thomas J. Watson Research Center. From 1984 to 1985 he worked at General Electric Corporate Research and Development, Schenectady, NY, in the areas of circuit design and advanced technology development. From 1985 to 1987 he worked at GE Intersil in the areas of analog and digital circuit design and advanced process development. From 1988 to 1989 he worked as a private consultant in the area of circuit design.

Dr. Chiu is a member of Eta Kappa Nu and Tau Beta Pi.
Walter C. Dietrich, Jr. received the B.S. in mathematical sciences from the University of North Carolina at Chapel Hill in 1979 and his M.S. in computer science from Cornell University in 1983. He is a research staff member in the Manufacturing Research Department at the IBM Thomas J. Watson Research Center. He was chair of the CFI TCAD Technical Subcommittee from 1990 to 1992, and he managed the project that prototyped and demonstrated SWR and SPR. His research interests include software reengineering, object-oriented design and programming, frameworks for modeling and planning in manufacturing enterprises, TCAD frameworks, and distributed systems. Mr. Dietrich is a member of the ACM and the IEEE Computer Society.

Purnendu K. Mozumder (S'85–M'88–S'88–M'89) received the B. Tech Degree in electrical engineering from the Indian Institute of Technology in 1984, and the M.S. and Ph.D. degrees in electrical and computer engineering in 1986 and 1989, respectively, from Carnegie Mellon University, Pittsburgh, PA. He is currently a member of the technical staff at the Semiconductor Process and Design Center of Texas Instruments Inc., Dallas, TX. He is presently working on design for manufacturing, TCAD, and statistical process control for VLSIC fabrication. He has been actively involved in the Microelectronics Manufacturing Science and Technology (MMST) project at TI. His interests include stochastic and adaptive control, statistical process and device simulation, TCAD framework development, and application of statistical and numerical techniques to IC design and manufacturing methodologies.

Dr. Mozumder is presently serving as a guest editor of the IEEE TRANSACTIONS ON SEMICONDUCTOR MANUFACTURING: Special Issue on MMST.

Michael Karasick (S'89–M'89) received the B.Sc. (hons.) in computer science from the University of Manitoba in 1981, and the M.S. and Ph.D. degrees in computer science from McGill University in 1983 and 1988, respectively. He was a member of the robotics research project at the Cornell University Department of Computer Science from 1985 to 1988. He joined IBM as a Research Staff member at the T. J. Watson Research Center in 1988, in the Modelling Systems Project of the Manufacturing Research Department. His chief research interests involve the use of geometric modelling for semiconductor process simulation. Dr. Karasick is a member of the ACM and SIAM.

Lee R. Nackman (S'79–M'81) received the B.Sc. degree in computer science from Brown University in 1976 and the Ph. D. degree, also in computer science, from the University of North Carolina at Chapel Hill in 1982. He has been a research staff member at the IBM Thomas J. Watson Research Center since 1982, and is now manager of the C++ programming environments project. He has previously worked in geometric algorithms and software system structures for computer-aided design systems and in applications of computational geometry.

V. T. Rajan Photo and biography not available at time of publication.

Mark Law (S'79–M'81–SM'92) received the Ph.D. degree from Stanford University, Stanford, CA, in 1988. After obtaining his degree, he spent one year at Stanford as a research associate. During this time, he co-authored the SUPREM-IV process simulator and his research focused on the modeling of point defects and dopant diffusion in silicon. He joined the University of Florida as an assistant professor in December 1988. His research interests are in process and device modeling for technology computer aided design. He is a co-principal investigator in the University of Florida SRC research on advanced device and circuit modeling and has research in the area of IC process modeling funded by NSF, Sematech, IBM, and Silicon. He has written 33 papers in the area of process and device modeling. He has been working in this field for over ten years both as a student and a researcher for Hewlett Packard.

Dr. Law was named a National Science Foundation Presidential Faculty Fellow and earlier he had won an IBM young investigator award. He has served on the technical committees of the Bipolar Circuits and Technology Meeting, the Numerical Process and Device Simulation Workshop (NUPAD), and the International Electron Devices Meeting. He was guest editor of a special issue of the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN devoted to the proceedings of NUPAD. He will be the technical chairman of the 1994 NUPAD-V conference. He has given invited talks at the 1993 SISDEP and the 1991 Spring Electrochemical Society meeting. He is a member of the Electrochemical Society, American Physical Society, Phi Kappa Phi, Tau Beta Pi, Eta Kappa Nu, and Sigma Xi.

D. M. H. Walker (S'77–M'86) received the B.S. degree in engineering with honors from the California Institute of Technology, Pasadena, CA, in 1979, and the M.S. and Ph.D. degrees in computer science from Carnegie Mellon University, Pittsburgh, PA, in 1984 and 1986, respectively.

He is Assistant Director and Research Engineer in the SRC-CMU Research Center of Excellence for Computer-Aided Design in the Department of Electrical and Computer Engineering at Carnegie Mellon University. He has held positions at Hughes Aircraft Company and Digital Equipment Corporation. His current research interests include simulation of local process disturbances, technology CAD frameworks, and design for manufacturability.

Dr. Walker is General Chair of the 1992 IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems and on the editorial board of IEEE TRANSACTIONS ON COMPUTERS. He is a member of the Association for Computing Machinery, the IEEE Circuits and Systems, CHMT, and Computer Societies, and Sigma Xi.
Robert H. Wang (S'89) received the B.S. (with high honors) and M.S. degrees in electrical engineering from the University of California, Berkeley, in 1989 and 1991, respectively. He is currently pursuing the Ph.D. degree in electrical engineering and computer sciences at the University of California, Berkeley. Since 1989, he has been a graduate student researcher in the Electronics Research Laboratory at the University of California, Berkeley. He has held engineering intern positions in the area of technology CAD at Intel Corporation and IBM Research, in 1988 and 1991-1992, respectively. His research interests include technology development, circuit design, and automatic layout of integrated circuits.

Mr. Wang is a member of Eta Kappa Nu and Tau Beta Pi.

Alexander S. Wong (S'87-M'92) received the B.S. degree in computer engineering from the University of Illinois at Urbana-Champaign in 1987, and the M.S. and Ph.D. degrees in electrical engineering from the University of California, Berkeley in 1989 and 1992, respectively.

He has held summer positions at General Electric Semiconductor, AT&T Bell Laboratories, and the Lawrence Livermore National Laboratory from 1985 to 1987. In 1989, he was a consultant for Motorola's Advanced Technology Center in Mesa, AZ. He is currently a senior CAD Engineer in the Technology CAD department at Intel Corporation in Santa Clara, CA. His technical interests include frameworks for integrated circuit and technology CAD, as well as process and device simulation.

Dr. Wong is Chairman of CAD Framework Initiative (CFI) Semiconductor Wafer Representation Group, and is a member of Eta Kappa Nu, Tau Beta Pi, and Phi Kappa Phi.