Chapter 5

Information Model, Services and Framework

5.1 Introduction

In the last three chapters, a general mesh generation strategy has been proposed for supporting semiconductor device simulators such as PISCES [PINT84] and STRIDE [WU90]. A mesh generator has limited usage for TCAD tools unless it is well incorporated into a TCAD system. This chapter discusses a general framework architecture which facilitates the integration of various utilities/services with TCAD tool development.

Before we proceed on the discussion, some terminologies need to be clarified. A TCAD *tool* is a piece of software (consisting of one or more programs) which is not configurable by the tool users. In other words, the behavior of a TCAD tool is determined by the tool developer(s). A TCAD *system* is a piece of software consisting of multiple tools and is configurable at the tool level by the system users. A TCAD system user can alter the system's behavior by deleting, replacing or adding a TCAD tool. A *tool developer* is the person who designs, implements, maintains and enhances the tool software. A *framework developer* is the person who designs, implements, maintains and enhances the framework software which is used by the tool developers. A *TCAD tool/system user* is the person who uses TCAD tools/systems.

The TCAD framework has mainly been served as an infrastructure for integrating various simulation tools, including process, lithography and device simulators. In the past, several integration framework systems have been implemented. A good overview on different TCAD framework systems can be found in [CHIN92]. An integration framework provides the TCAD system users many advantages including a consistent management of the data, a common graphical user interface(GUI) and a robust and efficient inter-tool communication (ITC).

In this chapter, a different aspect of the framework is discussed. Instead of integrating various tools, the framework is used as a tool development environment. The framework architecture proposed in this thesis has two components: the information model and services. The information model provides a consistent data model and the services enable the tool developers to focus more on computational issues specific to TCAD applications. The objectives of the development framework are

(i) to make the TCAD software more robust, better to maintain and easier to enhance by promoting a modular approach as the software development methodology;

(ii) to shorten the development cycle for a TCAD system by sharing common services.

Based on the proposed framework architecture, a prototype is implemented. The information model provided by the framework is based on the Semiconductor Wafer Representation (SWR)[1], a standard information model to characterize the behavior of the wafer state. The services supported by the prototype are the geometry modeling, mesh generation and visualization. To validate the framework architecture, several application tools have been incorporated, including a 3D device structure generator (ET3D [TSE93]), a 2D device simulator (PISCES [PINT84]) and an interconnect simulation tool (FASTCAP [NAPO92]).

This chapter is organized as follows: Section 5.2 discusses the needs for a development framework followed by a description of an architecture which satisfies the requirements for TCAD tool development. Section 5.3 provides a general discussion on the information model and the object-oriented implementation. In addition, a wafer state information model is discussed. Section 5.4 details various services which are required by

1. Detailed discussion on the SWR is found in Section 5. 5.

TCAD tool development, especially for 3D device applications. Section 5.5 presents a prototype implementation of the development framework.

5.2 A Tool Development Framework

5.2.1 Needs for a Development Framework

The complexity of TCAD tools has greatly increased in the last two decades. Several reasons are accounted for. First the device and process simulators are widely accepted in both the industry application and the academic research community. This demands more sophistication for tools to satisfy the needs for both expert and novice users. Experienced users demand more control of the tools in terms of flexible parameter calibration, easy incorporation of new physical models and selection of different numerical algorithms. New and casual users would like to minimize the learning period on how to use the tools. For them, the easy usage of tools is the main concern. Too many parameters and options provided by the tools confuse these users and discourage them from using the tools. To serve both the expert and the novice users, a more intelligent user interface with a modern graphic user interface (GUI) capability should be developed. Such a capability often requires a more complex software architecture.

Second, the usage of TCAD tools to predict and control the process and device behavior becomes a cost effective methodology for the process and device design. Simulation results are used to complement the experimental data to extract the circuit behavior. This mandates a more accurate and reliable simulation results from TCAD tools. To achieve this goal, more complex physical models and more sophisticated numerical algorithms need to be employed.

Third, the availability of the advanced software technology, including fast and reliable inter-tool communication, friendly graphical user interface and object-oriented database systems, has forced tool developers to incorporate these technologies which often require a more complex supporting data structure.

The increase in complexity of the tools has a major impact on how to develop, test, maintain and enhance a TCAD tool. The impact is evidenced by the fact that the development and test time for new tools become longer; the enhancement of an existing tool becomes more difficult and more tools are developed jointly by a large group of people. Software engineering requires a much large share of development effort for modern TCAD tools. Complex software architecture and multi-developer environment demands a well designed software development methodology. The key to the methodology is the modular approach and the object oriented design. Using the modular approach, an application tool is decomposed into a set of modules. Each module encapsulates its implementation and the module's behavior is characterized by its functional interface. With encapsulation, modules can be developed by multiple people and any implementation change of one module does not effect the rest of the tool development effort as long as the module's functional interface remains the same. Incorporating modular approach with encapsulation, the complexity of the implementation effort can be reduced without sacrificing functionality. Such a methodology has been adopted for developing several large software systems[BOOC91].

As more TCAD tools become modularized, many modules can be shared across different tools. For example, geometry modeling and mesh generation are needed by most device as well as process simulators. Both modules are difficult to implement, particularly for 3D applications. It is desirable for them to be shared by different simulators. Any duplicate effort prolongs the development time and makes the maintenance and future enhancement more difficult.

To facilitate the sharing of modules, an infrastructure should be created. A development framework proposed in this chapter represents an effort to accomplish this goal. In the rest of this chapter, an architecture which satisfies the needs for TCAD tools is proposed and implementation issues are discussed.

5.2.2 An Infrastructure Architecture

An information model based and service oriented framework is proposed as an infrastructure to facilitate TCAD tool development. The information model is served as a backbone to unify tools as well as services provided by the framework, as shown in Figure 5.1. TCAD tools can access information through the application procedure interface (API) and request services through the API as well. This approach combines the information model and services through a common interface mechanism, i.e. the application procedure interface. The tight coupling of the information model and services distinguishes our framework from the others which are more aimed at tool integration [DUVA88, WONG91].

Two issues should be addressed regarding to an information model. The first is what information should be provided by the information model. In our implementation, the wafer state information is provided. A *wafer* is an abstract entity which consists of one or more devices. The wafer state contains the information about the geometry structure and the physical properties. For example, the gate oxide thickness is the information about the geometry of a wafer and the electron mobility in the substrate is the information about the physical property. Two other information models, which are closely related to the wafer state, are the mask layout information model and the process flow information model. The mask layout information model is used to bridge the gap between the circuit level information model and the device level information model. The process flow information model provides the information about the process steps and the related equipment and environment parameters. In this thesis, only the wafer state information model is discussed.

The second issue regarding to the information model is how to access the information. Different methods have been proposed and implemented. One method is to provide the tool developers the complete implementation structure and let tool developers to find out

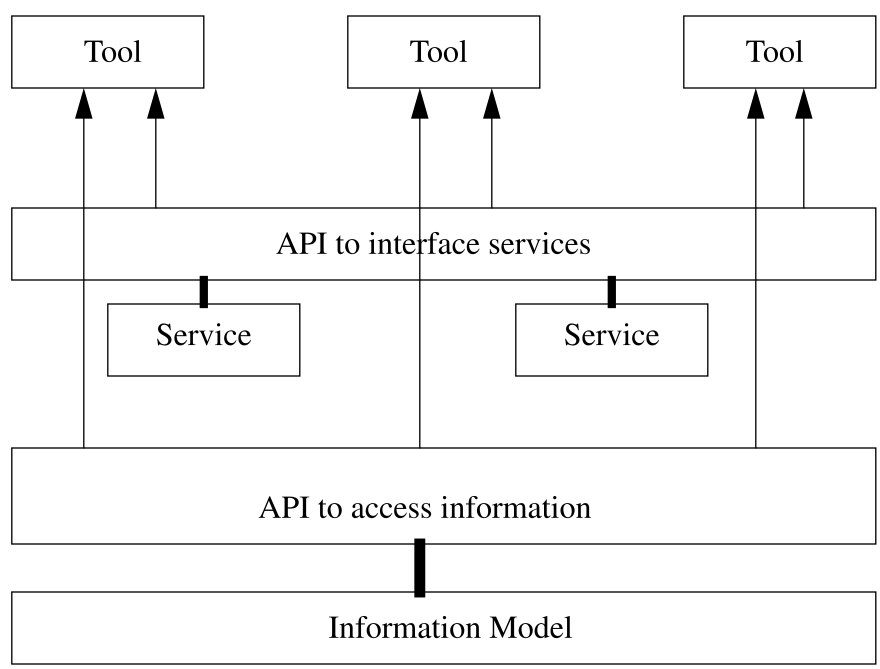


Figure 5.1 Architecture of a development framework.

the desired information by navigating through the structure. For example, if an intermediate file format is used to store a common information model, then it is up to the tool developers to find the information based on the syntax of the file format. Similarly, if an in-memory data structure is provided for storing the information model, then the tool developers find the information by following the data structure. In either case, the implementation is completely visible by the tool developers, which is not desirable because any implementation change of the information model requires a re-work of all application tools. A different approach is based on predicate logic. Information is accessed based on logical relationships among different entities. This approach hides all the implementation detail. One such implementation is the database technology based on relational algebra. Relational database systems have been successfully used in many financial software packages. But it is less successful in handling engineering problems with large data. A new approach based on the object-oriented technology is emerging as an alternative for engineering applications. A more detailed discussion on the object-oriented information modeling is presented in 5.3.3.

An illustration of the architecture for the semiconductor device simulation system is shown in Figure 5.2, where the framework consists of the wafer state information model and five services, which are the geometry modeling service, the mesh service, the visualization service, the graphic user interface service (GUI), the inter-tool communication service (ITC) and the persistency service. There are four application tools, a 3D device structure generator, a device simulator (2D or 3D), a mixed mode simulator and an interconnect simulator. A prototype system has been implemented. The information model is based on the SWR and supported services include geometry modeling, mesh generation and visualization. Application tools supported by the system include ET3D, PISCES and FASTCAP. ET3D is a simple 3D device structure generator which uses geometry modeling capability to generate 3D device structure based on the mask layout and process flow information [TSE93]. PISCES is used for both 2D device simulation and mixed mode simulation [PINT84, ROTE93]. FASTCAP is a capacitance extractor for interconnects developed at MIT[NAPO92].

5.3 Information Model

5.3.1 Information Model for TCAD Tools

An information model has two components: *data* and *schema*. The data stores the information and is created during the execution of the application programs. The schema, also called *metadata*, describes the structure of the data and is defined when an information model is designed. One most commonly used information model is the *entity-relationship* model [CHEN76]. In the entity-relationship model, entities are characterized by its attributes. Interactions among entities are described by relationships. The relationship can be *one-to-one*, *one-to-many* and *many-to-one*. In addition, a relationship can involve more than two objects, which is called a *ternary* relationship.

Figure 5.3 shows a partial information model to describe the geometry structure of a semiconductor device. There are three entities, namely *volume*, *boundary* and *vertex*. There is one attribute, *material_type*, for a volume, none for a boundary and two (the x and y coordinates) for a vertex. *Boundary_of* is a one-to-one relationship from a boundary entity to a volume entity. *Vertex_set* is a one-to-many relation from a boundary entity to a set of vertices. *Neighbor_of* is a many-to-many relation from a volume to another volume.

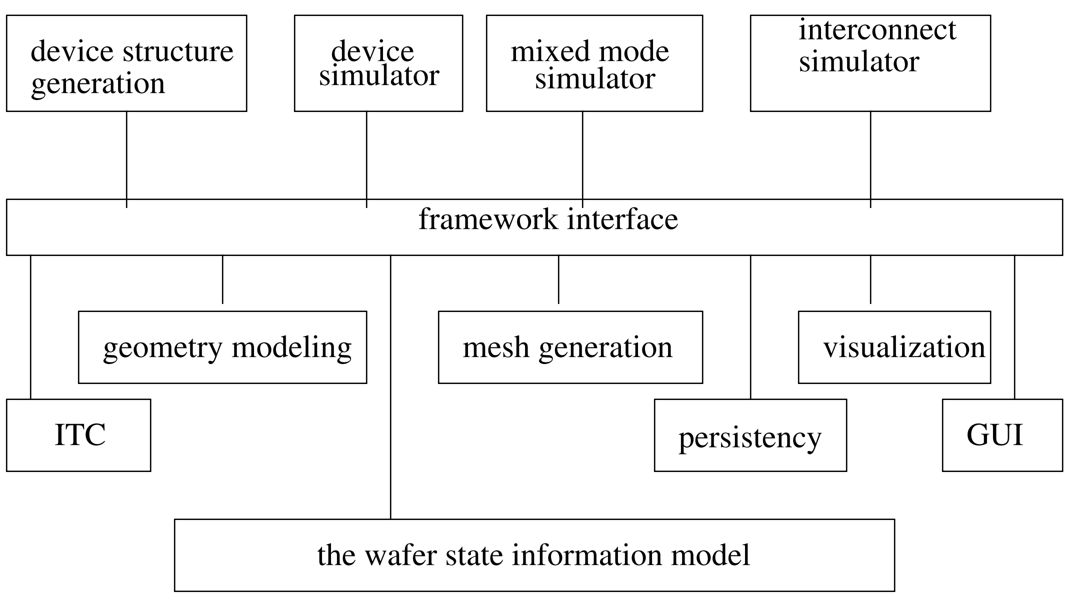


Figure 5.2 A device simulation system.

The semantics of graphic symbols used in the figure follows notations by Cattell [CATT91]. Boxes are used to denote entities, ovals to denote relationships and arrowheads to indicate multiplicity.

For the example shown in Figure 5.4, there are two volume entities *A* and *B*. *A* is the silicon substrate and *B* is the gate oxide. Based on the relationships defined in Figure 5.3, we have:

$neighbor\_of(A) = B$;

$neighbor\_of(B) = A;$

$boundary\_of(A) = p_0p_1p_2p_3;$

$boundary\_of(B) = p_4p_5p_6p_7;$

$vertex\_set(boundary\_of(A)) = \{p_0, p_1, p_2, p_3\};$

$vertex\_set(boundary\_of(B)) = \{p_4, p_5, p_6,p_7\}.$

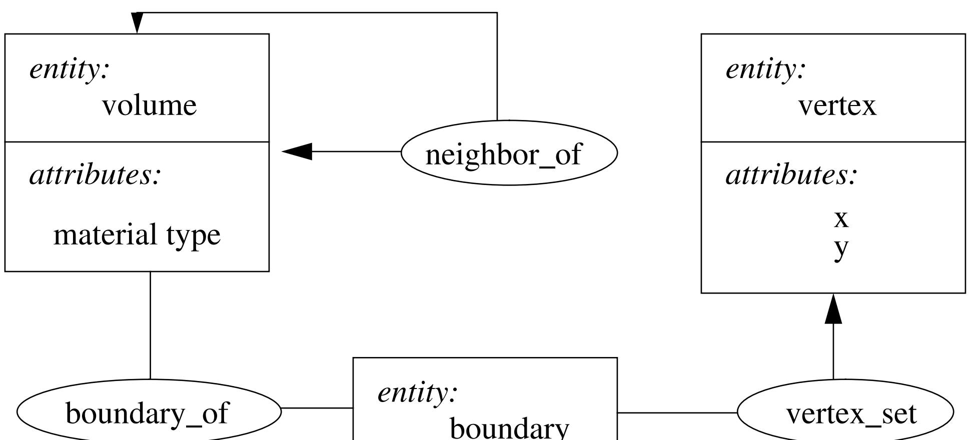


Figure 5.3 The entity-relationship for describing device structure,

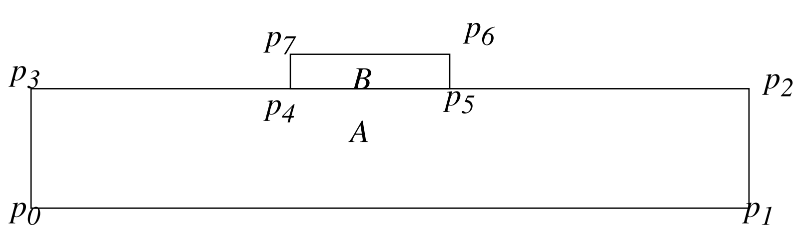


Figure 5.4 The MOS device geometry structure.

5.3.2 Information Modeling System

An information modeling system is to provide a programming interface such that tool developers can (i) create, access and destroy entities, (ii) access and modify attributes of an entity and (iii) access the relationships between entities. Several issues regarding to implementation of an information modeling system need to be addressed. First, the information integrity should be maintained. Any access methods provided by the information modeling system should not introduce any inconsistent data. For the example shown in Figure 5.4, suppose the information modeling system provides a vertex move operation which can be used to change a vertex location. By moving the vertex p_5 downwards, a potential inconsistency may occur due to the overlap of the silicon substrate and the gate oxide.

Second, the access methods should be sufficient for tool development. For the example shown in Figure 5.4, suppose we want to check if a point is inside, outside or on the boundary of the silicon substrate. The checking would take a significant amount of programming effort if only the relationships described in Figure 5.3 are provided. In other words, such an information modeling system has very limited usage.

Third, access methods should be implemented efficiently. A major concern for using a relational database for TCAD applications is its lack of efficiency. Relational algebra based query language is powerful for information access, but the execution time is not adequate for most TCAD applications.

5.3.3 An Object-Oriented Information Modeling System

Systems based on object-oriented information models stemmed from the object-oriented programming paradigm. The abstract data type declaration is called a *class*. The invocation of a class is called an *object*. Each class declaration is divided into two parts, the private and public part. The public part describes the characteristics and the behavior of an object. The explicit declaration of the private and public part encapsulate the implementation details. Another difference between a procedural language and an object oriented language is that the behavior of an object can be expressed by a public procedure attached to the object, which is called the *method* of the object. In the past, different object oriented programming languages have been developed, such as SmallTalk, Ada and C++. In the discussion of this thesis, C++[ELLI90] is used exclusively.

As discussed in the previous subsection, the implementation of an information modeling system depends on (1) how an entity is described, (2) how relationships are described and (3) what information access methods are provided. Different implementation architectures provide different ways to answer these questions. A good overview of different approaches can be found in [ULLM89]. In the last several years, the object-oriented implementation has emerged as a promising solution for engineering applications.

In an object-oriented information modeling system, entities are mapped to objects, which may have behavior as well as data. The data of an object is implemented through public attributes and public methods. The behavior of an object is expressed by public methods only. For the MOSFET example in Figure 5.4, we can use the following class declaration to describe volume objects, i.e. the silicon substrate and the field oxide.

```
CLASS Volume
{
    PUBLIC:
        char *material;
        float doping(float x, float y);
    PRIVATE:
        ...
}
```

where the material attribute is accessed through a public data field *material*. The doping concentration is accessed through a public method *doping*.

5.3.4 Information Model for the Wafer State

The information model for the wafer state provides two pieces of information, the geometry structure and physical property. A boundary based solid modeling representation is used to describe the geometry structure of a device. The physical properties such as doping concentration, mobility and etc. are defined using a field with three components; a mesh, values on the mesh vertices and an interpolation method.

Geometry objects such as volume, shell, face, loop, edge and vertex, and the relationships among these objects are used to represent a general 3D geometry structure. A *volume* is a 3D polygonal object. A *shell* is a segment of the boundary for a volume and a shell is connected. If a volume does not have any void interior, then the boundary consists of one shell. A shell is made of one or more faces. A *face* is a bounded and connected planar region. Each face is bounded by one or more loops. If a face is singly connected, the boundary of the face consists of one loop. A *loop* consists of one or more edges. An *edge* is a singly connected and bounded segment of a line with two vertices. A *vertex* is a point in the space which has x, y and z coordinates. The example shown in Figure 5.5 has one volume bounded by one shell. The shell has 10 faces. Except for the top and the bottom faces, each face has only one loop. For example, the front face $p_0p_1p_4p_5$ has one loop with four edges, p_0p_1, p_1p_5, p_5p_4 and p_4p_1. The top face of the shell has two loops, $p_4p_5p_6p_7$ and $q_4q_7q_6q_5$. Similarly the bottom face has two loops, $p_0p_1p_2p_3$ and $q_0q_1q_2q_3$.

Another example is shown in Figure 5.6. The volume is created by taking out a small cube from a larger cube, i.e. a solid volume with a void cube inside. In this example, there are two shells, the inner shell and the outer shell. Each shell has 6 faces and each face has one loop with 4 edges.

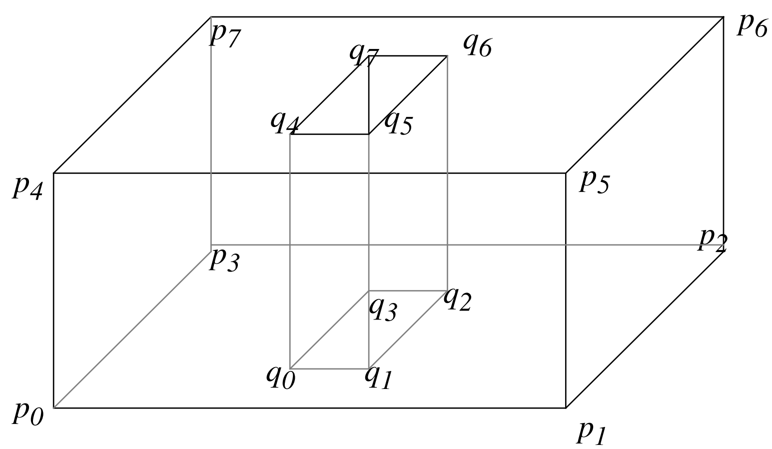


Figure 5.5 An example of a 3D geometry object.

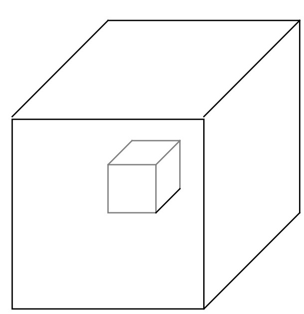


Figure 5.6 A volume with two shells.

The relationship among the geometry objects are depicted in Figure 5.7. All relationships are many-to-many except for the relationship *shell_of*, which is one-to-many because a volume can have multiple shells but each shell belongs to one volume.

Another useful relationship is the location relationship which detects if a vertex is inside a volume, outside a volume or on the boundary of the volume, as shown in Figure 5.8.

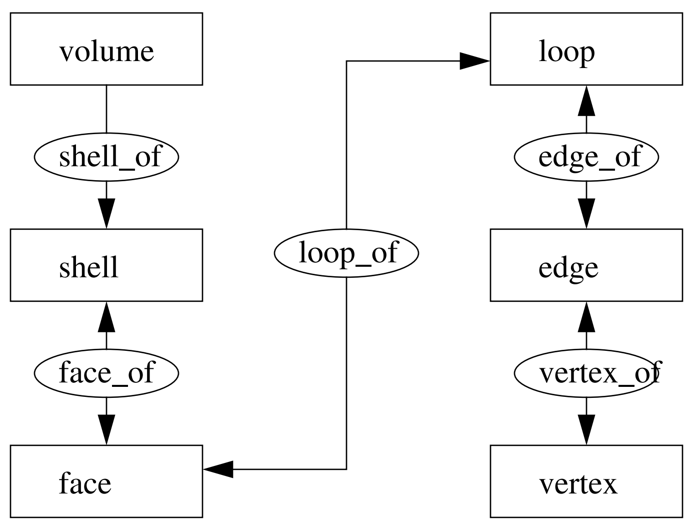


Figure 5.7 Relationships among different geometry objects.

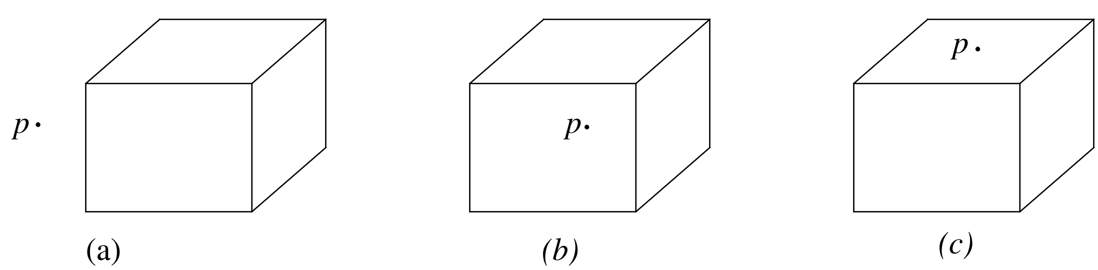


Figure 5.8 Location relationship: (a) *P* is outside the volume. (b) *P* is inside the volume. (c) *P* is on the boundary of the volume.

The second component of the wafer state information model is the field. A field consists of a mesh, data on mesh vertices and an interpolation method, also called the shape function, as shown in Figure 5.9. The relationship *shape_of* is many-to-one because each field has one mesh but one mesh can serve multiple fields. The relationship *data_of* is many-to-one because each field has one set of vertex data values but one set of vertex data values can belong to different fields due to different shape functions. Similarly the relationship *shape_funcion_of* is many-to-one because each field has a unique shape function while the same shape function can be used for interpolation by different fields. There are different shape functions depending on how smooth the interpolation is and how many data points are provided on each mesh element.

For the example shown in Figure 5.10, the doping field d is defined on a tensor product mesh. The doping data are given on mesh vertices and the rectilinear shape function is used for interpolation. Suppose the point $p(x, y)$ is inside the element bounded by four vertices p_0, p_1, p_2 and p_3. Assume the doping values on four vertices are d_i $(i=0, 1, 2, 3)$. Then the doping value at p is computed as follows:

$$d(x, y) = (1-t_x)(1-t_y)d_0 + t_x(1-t_y)d_1 + t_x t_y d_2 + (1-t_x)t_y d_3$$

A complete information model to represent a wafer state consists both the geometry and the field.

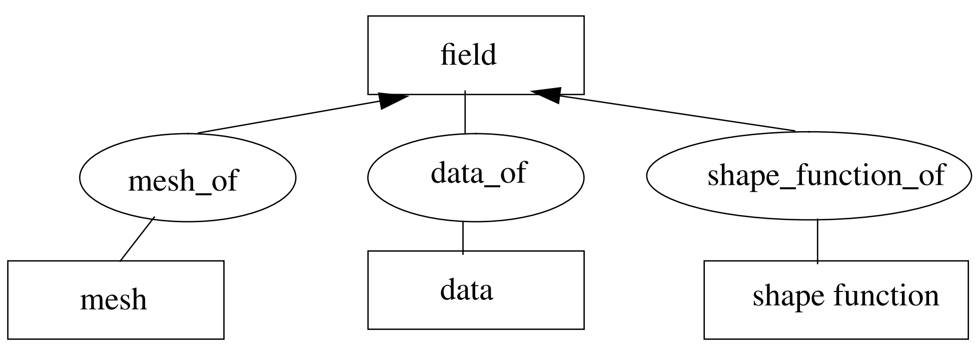


Figure 5.9 Field components: mesh, data and shape function.

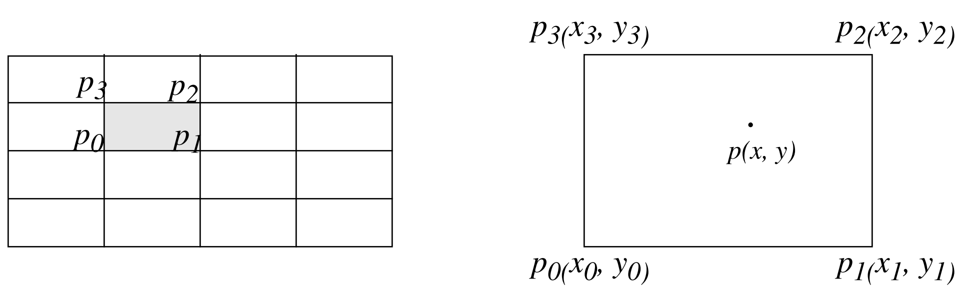
5.4 Services for TCAD Tool Development

Information models provide an integration backbone for TCAD tools. However, to facilitate the tool development, service modules shared by multiple tools should be provided at the framework level. In this section, a number of services common to 3D TCAD tools are discussed. In 5.4.1, several basic services, including the user interface, the data persistency and the extension language, are reviewed. Subsection 5.4.2 provides a detailed discussion about the geometry modeling service. Subsection 5.4.3 describes the mesh service.

5.4.1 Review of Some Basic Services

A framework should provide a common user interface for TCAD tool development. In the past many graphical user interface packages have been developed on top of X window system such as Openlook, MOTIF, Interview and TK/TCL. All these toolkits provide a graphics interface for controlling the display hardware and the interface with application tools is developed through a set of widgets. Widget sets define the “look-and-feel” of an application. Although a widget set is a collection of independent components, a cohesive set often defines conventions for uniform pointer and keyboard input, data display, and inter-component communication.

The support of data persistent storage is another basic service which should be provided by the framework. All database software packages, including the relational database and the object oriented database systems, provide the service. Recently a new



(a) (b)

Figure 5.10 Illustration of a doping field: (a) The doping field.
(b) Enlargement for the shaded element.

HDF-V has been proposed as an alternative storage mean for TCAD applications [YEAG92].

Extension languages have been recognized as an important ingredient for both the Electronic Design Automation (EDA) and TCAD frameworks. It allows users and tool integrators to customize the system in increasingly powerful ways. Several extension languages have been successfully implemented in commercial systems, SKILL from Cadence Design Systems and Genie from Mentor Graphics. VISTA, a TCAD framework developed by Viena Technical InstituteVISTA, has adopted XLisp as its own extension language [HANA93]. Another popular extension language is TK/TCL, a task control language developed at UC Berkeley [OUST90, OUST91].

5.4.2 Geometry Modeling Service

To support the development of 3D TCAD tools, the geometry modeling service, also called the *solid modeling*, should be provided at the framework level. The solid modeling service consists of a set of geometry operations which are categorized into three groups: boolean operations, mixed dimension operations and solid transformations.

The three basic boolean operations are *intersect*, *subtract* and *merge*, as shown in Figure 5.11.

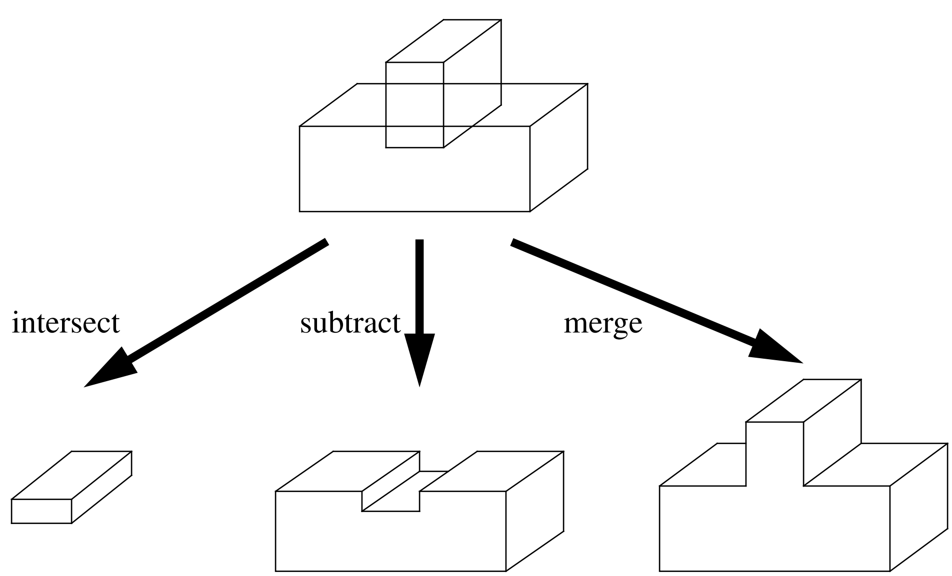


Figure 5.11 Intersect, subtract and union geometry operations in 3D.

Based on some basic geometry structures, i.e. box, tetrahedron, pyramid and prim, we can construct a large set of geometry structure by repeatedly applying the boolean operations. The following SRAM cell, as shown in Figure 5.12, illustrates the capability of boolean operations in constructing the geometry a device.

Mixed dimension operations can be used for constructing 3D geometry objects and computing geometry properties. In this subsection, we only discuss a few of these operations. A *sweep* is a geometry operation which sweeps a 2D face along a loop, as shown in Figure 5.13

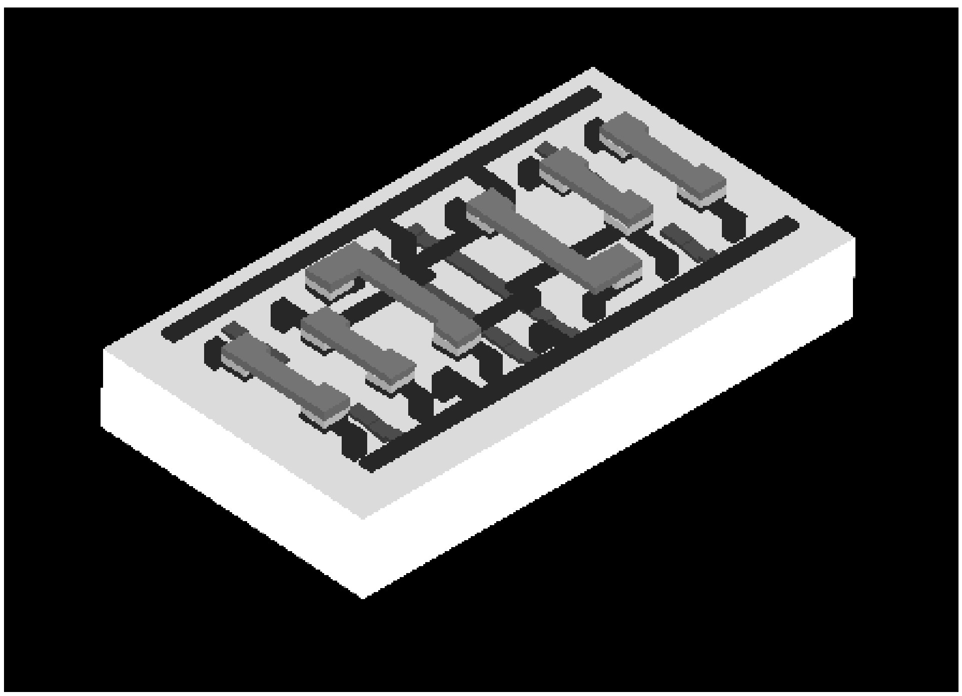


Figure 5.12 A SRAM structure.

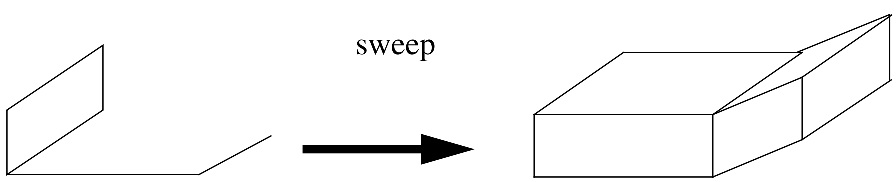


Figure 5.13 Sweeping a face along a loop of two edges to form a 3D volume.

The sweep operation can be used to approximate the process simulation result. The example shown in Figure 5.14 illustrates how a sweep operation can be used to obtain an isotropic oxidation.

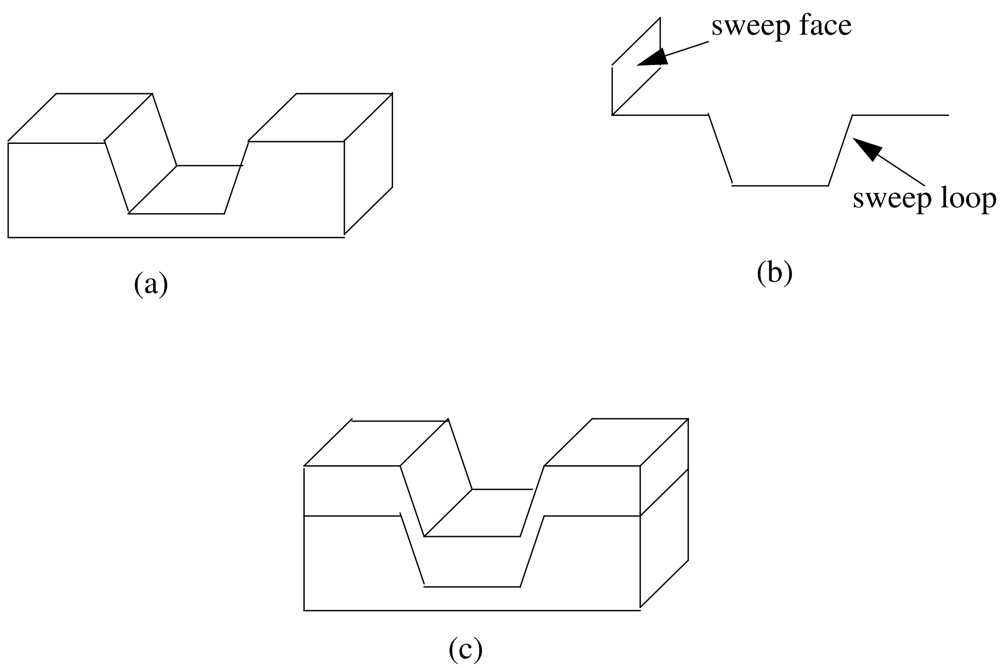
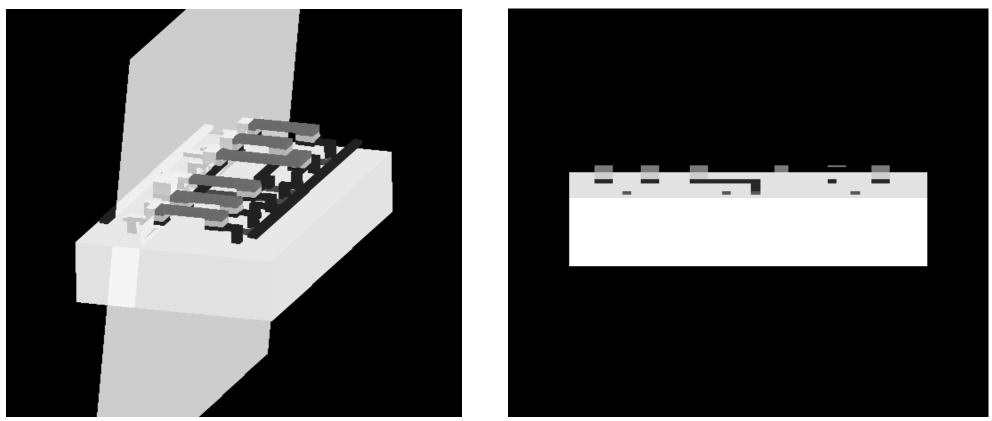


Figure 5.14 An illustration of using sweep operation for isotropic oxidation: (a) Silicon with a trench. (b) Sweep operation. (c) After oxidation.

A *section* operation is an intersection of a plane with a volume and forms a face. The section operation can be used to obtain a 2D cross section from a 3D structure, as shown in Figure 5.15, where a 2D cross section is obtained for the SRAM cell.

Several other useful operations are the intersections of *two faces*, *an edge and a face* or *two edges*, as shown in Figure 5.16.



(a) (b)

Figure 5.15 Illustration of the section operation: (a) Sectioning the SRAM cell. (b) 2D cross section.

(a) (b)

Figure 5.16 Geometry operations with mixed dimensions: (a) Face-face intersection. (b) Line-face intersection.

The third type of geometry operations is the solid transformation, including the *translation* and the *rotation*, as shown in Figure 5.17.

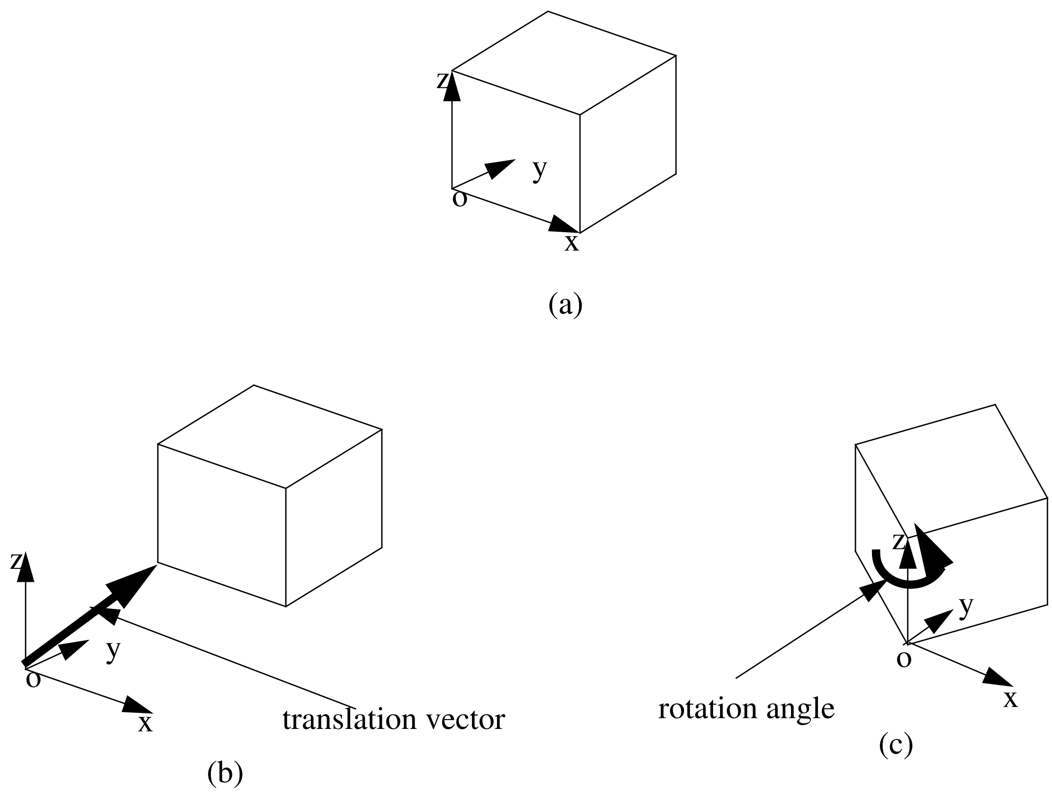


Figure 5.17 Illustration of the solid transformations: (a) A box. (b) After translation. (c) After rotation

5.4.3 Mesh Service

Mesh service is required by almost every numerical method based simulation tool. For 3D device structure, there are three commonly used meshes, 3D volume mesh, 2D cross section mesh and 3D surface mesh, as shown in Figure 5.18. Consistent meshes should be provided by the mesh service. In our implementation of the framework, the quadtree/ octree based unstructured meshes are supported.

5.5 Semiconductor Wafer Representation

5.5.1 SWR Overview

The Semiconductor Wafer Representation (SWR) is an information model for representing wafer state. A major difference between SWR and other information models for semiconductor device is the build-in geometry modeling capability for SWR [DUVA88, WONG90, WALK93, HALA93]. SWR has been adopted as the TCAD framework standard by CAD Framework Initiative, Inc. [TCAD92, TCAD93]. More information about the history of CFI and SWR can be found in [BARN92, CHIN92].

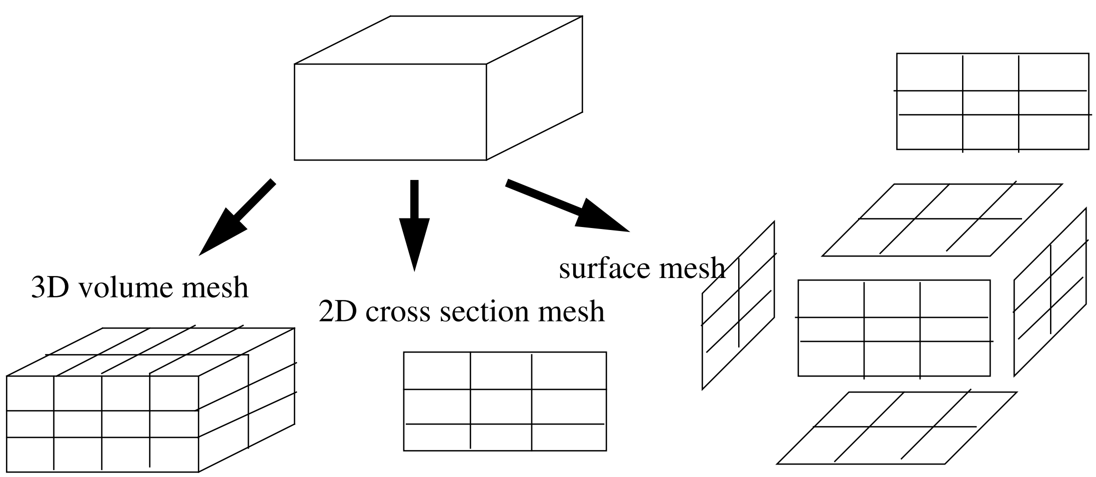


Figure 5.18 Three types of mesh for a 3D volume: volume mesh, 2D cross section mesh and surface mesh.

The SWR consists of two servers, the geometry server and the field server. The geometry server provides the geometry representation and their related operations and the field server provides the field representation and their related operations. It is the responsibility of the SWR server to maintain the data consistency between two representations. For the MOSFET example shown in Figure 5.19, the geometry consists of two volumes, the gate oxide *A* and the silicon substrate *B*. The impurity field is defined on a mesh and the rectilinear interpolation is used. A change in the geometry due to an etch also changes the domain of the corresponding impurity field.

The geometry represented in the geometry server is similar to the one discussed in Section 5.3.4 except that it supports the cellular cell concept, i.e. two volumes which share the same boundary face can be directly accessed through the method of the face. For the MOSFET example, the oxide volume *A* and the substrate volume *B* can be directly accessed from the boundary face $p_0p_1p_2p_3$. This capability is convenient for the lithography simulation [WANG93]. In addition to vertices, edges, loops, faces, shells and volumes, a geometry object called *cellcomplex* is provided. A cell complex is a collection

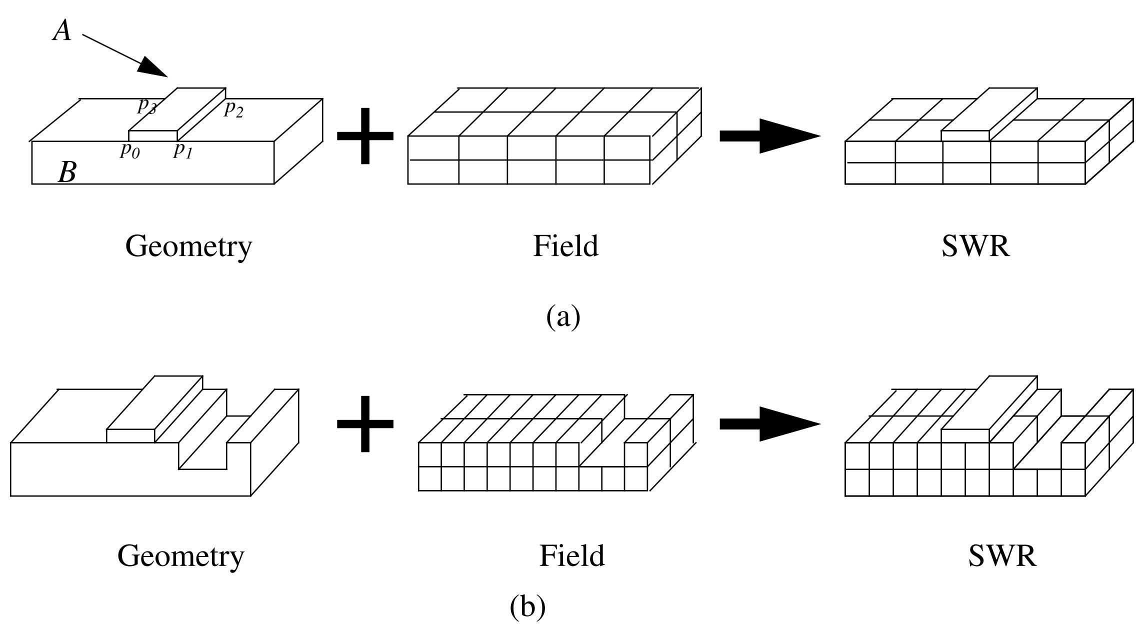


Figure 5.19 SWR for a MOSFET device: (a) Before the etch. (b) After the etch.

of volumes which constitute a partition of a region. In other words, a cell complex is a volume with internal boundaries. For the MOSFET example, the cell complex consists of two volumes *A* and *B*, which shares the internal boundary $p_0p_1p_2p_3$. Figure 5.20 depicts all the geometry objects and relationships supported by the geometry server.

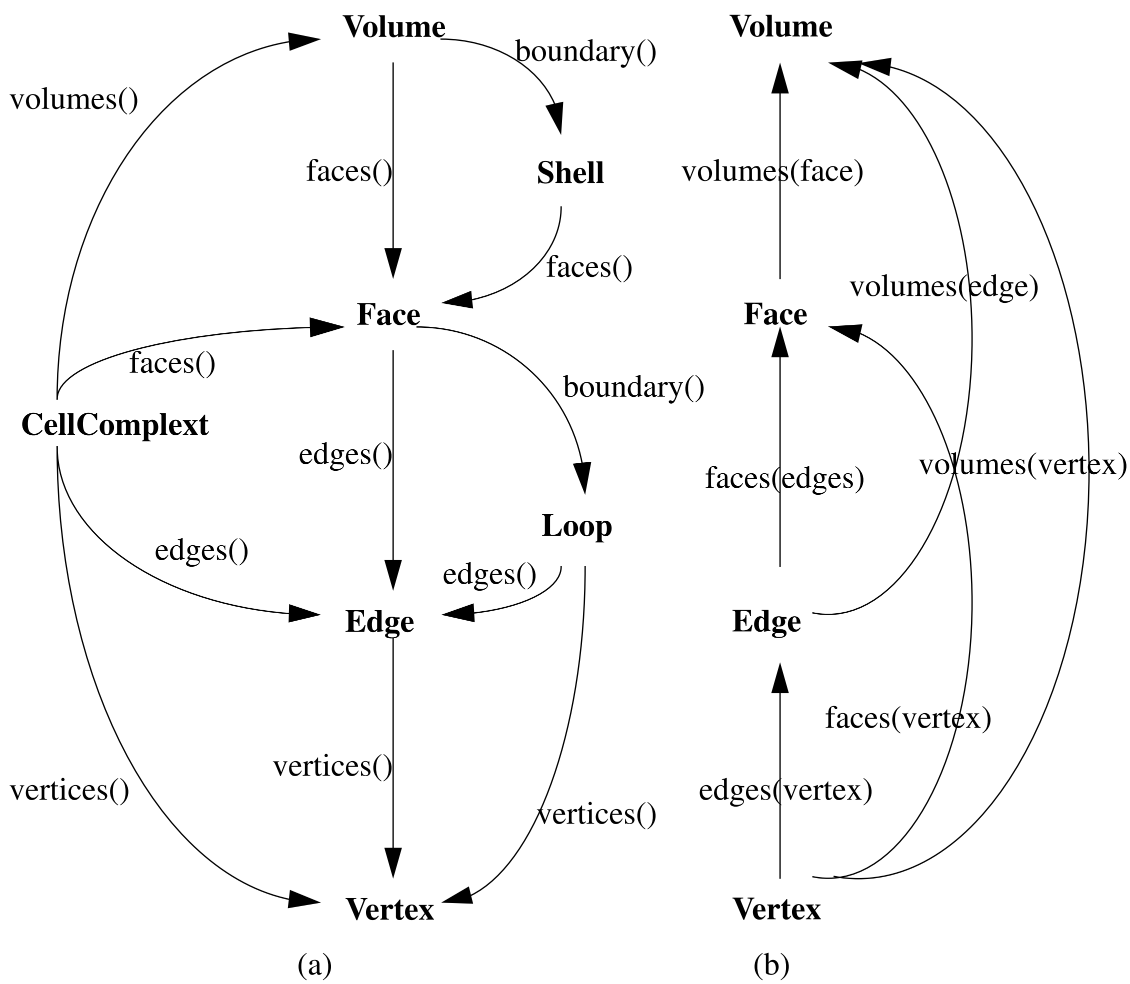


Figure 5.20 The SWR geometry structure: (a) Methods to traverse between swrCells. (b) Incident queries on cells of a cell complex. (Courtesy of CFI's SWR PI Document)

In addition to the geometry operations of merge, subtract and intersection, the geometry server supports two other boolean operations, *inset* and *glue*. A glue operation is a merge without removing any internal boundaries, as illustrated in Figure 5.21. An inset operation is a combination of an intersect followed by a subtract and then by a glue.

The field server supports the field representation which consists of mesh, data and shape function. For SWR1.0, only tensor mesh and triangle mesh in 2D are supported[TCAD93]. In our initial prototype implementation, the field server was not included. Further information on the SWR field server can be found in [CHIN92, TCAD93].

5.5.2 SWR3D - an ACIS Based Prototype Implementation

ACIS is an object-oriented geometry modeler from Spatial Technology Inc [SPAT91]. We have built a layer, also called husk, on top of ACIS to implement the SWR3D information model. Some implementation issues are presented in this subsection.

Similar to the SWR, ACIS uses a boundary based geometry representation for 3D geometry objects. However, a topology-geometry dual representation is adopted by ACIS to support non-polygonal objects, i.e. non-planar surface and non-linear edges. This duality makes the future enhancement to support non-polygonal geometry possible with a penalty of a significant increase in the computation time for the geometry modeling

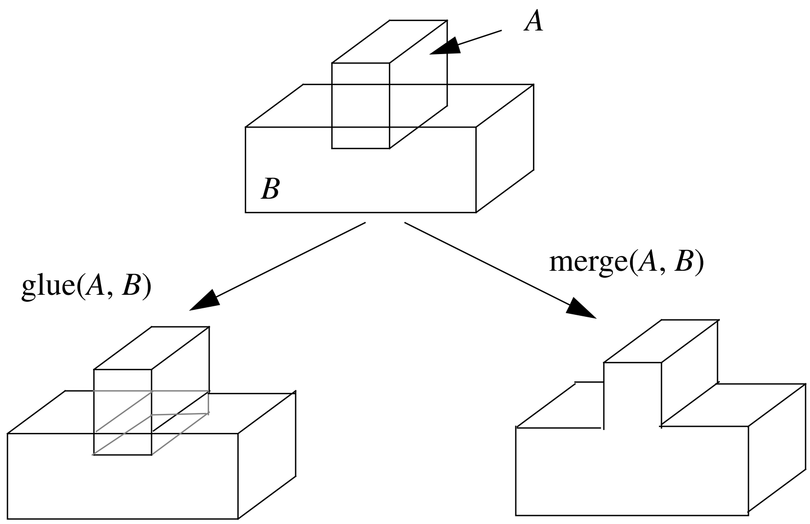


Figure 5.21 Difference between a glue and a merge operation.

operations. Figure 5.22 depicts the geometry and topological objects with their relationships supported by ACIS. One major difference between the SWR specification and ACIS is that the current ACIS does not support cellular complex cells. A *body* is a collection of *lumps*, where the term lump is used by ACIS to represent a volume.

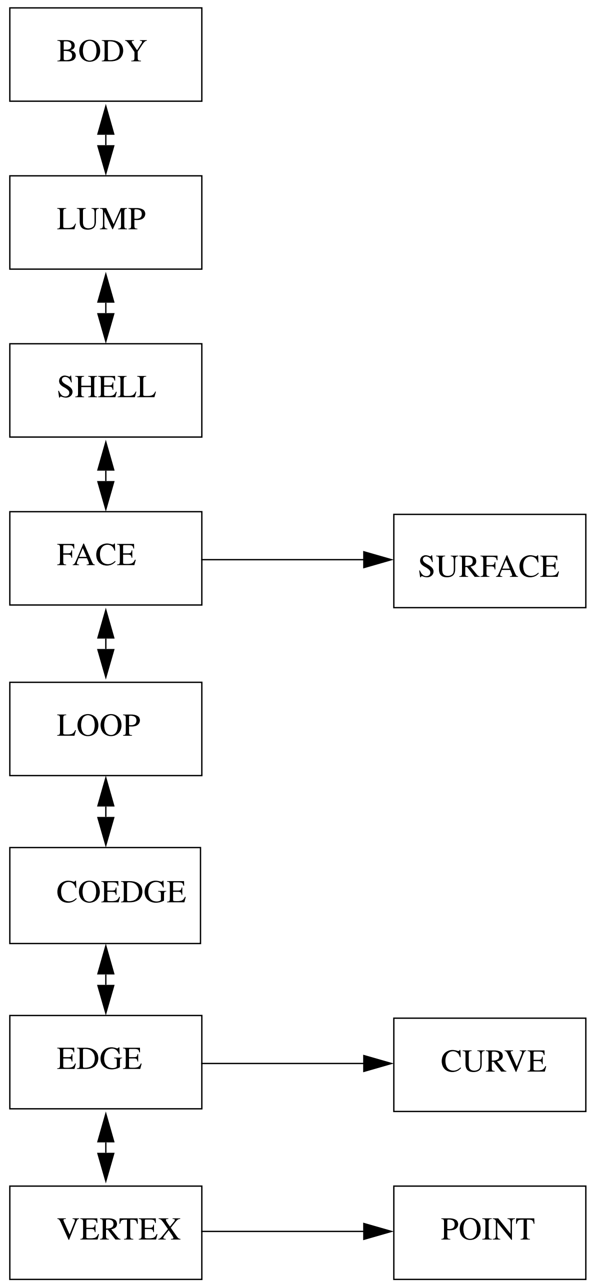


Figure 5.22 Topological and geometry objects supported by ACIS.

All query and geometry operations required by the SWR can be implemented using ACIS interface methods except for the inset operation due to the lack of the support of the cellular cell.

A useful feature provided by ACIS is the capability to support attribution propagation for the geometry operations. This is illustrated by the following example as shown in Figure 5.23. The material attribute oxide is attached to the volume *A*. The etch process can be implemented with subtract operation. After the etch, we would like the material attribute oxide to be attached to volumes A_1 and A_2, i.e. the material attribute oxide should be preserved after the subtraction is performed.

In ACIS, each geometry operation can be customized for dealing with attribution propagation through call back function schemes. When a new geometry object is created, four call-back functions are registered, which are DELETE, MERGE, READ and WRITE. These functions are executed when geometry operations are performed. This enables different propagation policies to be applied to different geometry objects.

5.6 Summary

In this chapter, we discussed issues related to the development of a TCAD framework. An information model based framework with services is presented as the architecture for the infrastructure to facilitate TCAD application program development. A list of services are identified which should be supported at the framework level. We presented the prototype implementation of the TCAD development based on the Semiconductor Wafer Representation.

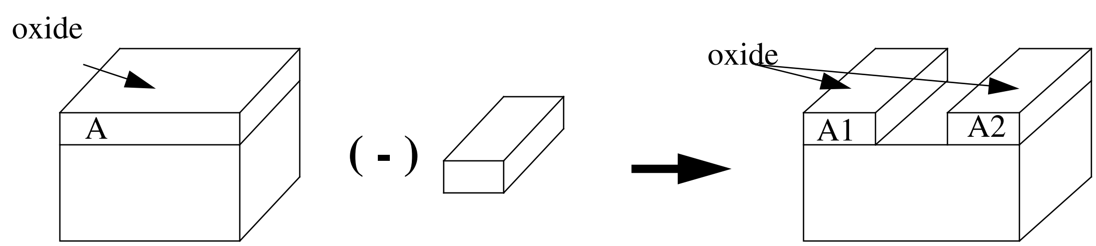


Figure 5.23 An example of the attribution propagation.