VISTA—User Interface, Task Level, and Tool Integration

Stefan Halama, *Member, IEEE,* Christoph Pichler, *Student Member, IEEE,* Gerhard Rieger, *Student Member, IEEE,* Gerhard Schrom, *Member, IEEE,* Thomas Simlinger, and Siegfried Selberherr, *Fellow, IEEE*

Abstract—**Succeeding an earlier paper [1] on the data level, the Viennese Integrated System for Technology CAD Applications (VISTA), an integration and development system for Technology CAD, is presented. Starting with a short overview of TCAD methodology and existing integrated systems, portability and comprehensibility are postulated as key considerations and an application-framework architecture is proposed. The design of VISTA's user interface and task level, presented next, adheres strictly to these guidelines. To enable the cooperation of independent simulation tools, an automatic triangulation-based service is provided by VISTA to resolve inconsistencies in wafer representations. A final example shows how three different simulators, integrated by the framework are used to simulate a planarized, trench-isolated 0.25 μm CMOS process.**

I. Introduction

A. TCAD Methodology

TCAD methodology consists of two parts. These are the *application* and the *development* of methods for the computer aided design of semiconductor technology. The development of these methods is driven by the dependence of TCAD on fabrication technology [2]. When seeking to improve the common lag of TCAD (with respect to current semiconductor technology) one must shift the attention from the commonly emphasized *application* aspects of TCAD methodology to its *development* aspects.

New capabilities are provided within a TCAD system by extending existing applications (which has often caused an unsound growth of code and has eventually lead to maintenance problems) by *developing* new tools, or by *integrating* additional, existing applications which implement the required functionality.

The time between the request for new simulation capabilities and their actual availability is comprised of several time factors, each corresponding to a phase of a typical TCAD tool development process:

- problem formulation;
- physical modeling;
- initial tool implementation;
- maintenance (improvement of robustness, porting, debugging);
- tool calibration;
- tool integration on the data and task level (which may require significant efforts to improve the inter-operability with other tools);
- user interface adaptions for ease of use; and
- acceptance, acquaintance, and learning phase of end users.

Apart from the time required for understanding and abstract modeling of physical problems, major causes for unforseeable delays are rather general software problems, which are not specific to TCAD.

From a historical perspective, the evolution of TCAD started with single applications (so-called *point tools*) that were implemented independently by different *tool developers*. Traditional point tools are process and device simulators which solve specific, isolated TCAD problems and which have been designed without envisioning the software environment in which they will be used. Later, these process and device simulation tools have been integrated with other services (like visualization, interactive design editing facilities, and a tool control level) to form *Technology CAD Systems*. An overview of existing TCAD systems is given in [3] and [4]. To support the *users* in the *application* of the methods provided, most of these TCAD systems are equipped with converter-based tool coupling, homogenizing user interfaces and TCAD-specific task level environments (often focused on tasks like optimization). Although a significant improvement of TCAD is perceptible, most of these *application*-oriented integrative efforts do not address the software-related difficulties which slow down the *creation process* and limit the responsiveness of TCAD.

B. TCAD Levels

A notion commonly found in TCAD and Electronic Design Automation ([5]) uses so-called *levels* which correspond to different views of integrated multitool systems. The *data level* is the home of the wafer (and sometimes process) representation, it provides the database for tool coupling. The *tool level* is where the simulation functions reside. The *task level* is a homogeneous control environment where operations and flows are defined and executed. The *presentation level* is the interface through which the user interacts with the TCAD system. The architectures of TCAD systems in use do not

Manuscript received December 6, 1994; revised May 1, 1995. The VISTA project has been sponsored by the research laboratories of Austrian Industries-AMS, Unterpremstätten, Austria; Digital Equipment, Hudson, MA; Siemens, Munich, FRG; and Sony, Atsugi, Japan, and by the "Forschungsförderungsfonds für die gewerbliche Wirtschaft," Project 2/285 and Project 2/299, as part of ADEQUAT (JESSI Project BT1B), ESPRIT Project 7236, and by ADEQUAT II (JESSI Project BT), ESPRIT Project 8002. This paper was recommended by Associate Editor S. G. Duvall.

The authors are with the Institute for Microelectronics, Technical University Vienna, A-1040 Vienna, Austria.

IEEE Log Number 9414151.

TABLE I
LIST OF TCAD SYSTEMS AND TOOL INTEGRATION APPROACHES, IN ALPHABETICAL ORDER OF INSTITUTIONS (INT.: INTERNAL, PROD.: PRODUCTION, IND.: INDUSTRIAL)

Name	Institution	Status	Data level	Task level	Presentation Level	References
MECCA	*AT&T*	int. prod.	awk/sed, C++	UNIX Shell, Tcl	Tk	[43; 44; 16]
PREDITOR pdFab	*CMU*	experimental, commercial	CDB/HCDB	Tcl	Tk, Motif	[45; 20; 46]
"Superviser"	*Hitachi*	int. prod.	Converter	"Superviser"		[47]
VATS	*IBM*	int. prod.	VATS/DB	WIZARD (Tcl)	WIZARD (Tk)	[17]
EASE	*Intel*	int. prod.	PIF predecessor, PIF derivative	UNIX Shell, FASST/TEL	Motif	[2; 48; 49]
CAFE	*MIT*	internal	BPIF/Gestalt	*MIT* PFR		[50] [51]
P&D Workbench	*NEC*	int. prod.		MEDLEY+ ESCORT	DAIJOBDA	[52]
UNISAS	*Oki*	int. prod.	GCOS	UNICOL		[53]
IDDE	*Philips UK*	int. prod.	ASCII PIF derivative	none	Apollo DIALOGUE	[54; 55]
PRIDE	*Philips USA*	int. prod.	ASCII PIF derivative	none	SunView	[56; 7]
SATURN	*Siemens*	int. prod.	SATURN	UNIX Shell	none	[57; 18]
MASTER	*Silvaco*	commercial	SSF	none	DeckBuild	[58]
STORM	*ESPRIT*	production	DAMSEL	none	STORM-UI IDAS	[59]
Alladin, SEWB	*Stanford*	ind. prototype	SWR			[60]
CAESAR	*TMA*	commercial	TIF	Module/Step	CAESAR	[61; 62]
ITS	*TI*	int. prod.	none	none	Motif	[63]
"System controller"	*Toshiba*	int. prod.	ASCII topography	interface programs	none	[64]
VISTA	*TU Vienna*	production	PAI/PIF	XLISP	Xvw + VUI	[65; 66; 67; 68]
PROSE	*UC Berkeley*	internal	BPIF, SWR	Tcl	Tk + VEM	[69; 9; 70]
SIMPL-IPX	*UC Berkeley*	internal	Converter (BTU)	none	SIMPL-DIX	[8; 71]

perfectly match this conception, but the correspondence is strong enough to be used as a basis for the characterization of existing implementations.

C. Existing TCAD Systems

Some of the existing Technology CAD systems have evolved from Electronic CAD (ECAD) or computer integrated manufacturing (CIM) systems. Hence the distinction between advanced ECAD,[1] extended CIM, and dedicated TCAD systems is sometimes not trivial. Most of the systems in use are not publicly available, and neither is the documentation of their architecture and implementation. Nevertheless, using the sparse published data, we have compiled a brief overview.

In Table I *internal* means that the system is not used outside the institution listed. *Production* means that the system is known to be *in use* somewhere. *Commercial* means that the system is commercially available. Blank fields indicate lack of reliable information.

Each of the systems listed exhibits a different architecture and emphasizes certain aspects of TCAD. Although remarkable achievements in terms of current TCAD application, most of the systems suffer from a common neglect of the development side of methodology.

D. Qualitative Requirements

The main purpose of the TCAD systems listed is the integration of multiple simulation tools. A detailed, comprehensive, and stable *a priori* specification of the functionality and services required for multitool integration (on the data, task, and presentation level) and for tool development is hardly possible, mainly owing to the strong dynamics of TCAD methodology. However, remembering the desired improvement of the methodology creation process, a more general, qualitative guideline for design and implementation of a TCAD system may be given. Among the many qualitative demands (such as orthogonal functionality, ease of use, customizability, configurability, robustness) we consider most important the following.

1) **Portability:** For general application software, but especially for sophisticated and expensive technical applications, like CAD systems which run on workstations, the need for operating system independence is widely acknowledged. However, in existing approaches this requirement is often disregarded for the sake of reduced implementation effort. Especially in TCAD, where a broad spectrum of workstations and mainframe platforms are in use, a high degree of operating-system independence is essential.
2) **Comprehensibility:** The intrinsic semantical complexity of TCAD indicates that special care will have to be exerted in order to create a system which is still understandable (for both users and programmers) with an acceptable expenditure of time and effort. Conceptual integrity [6], as the most important design consideration for controlling system complexity, should be fostered by favoring the generalization of existing concepts over the introduction of new ones. It is desirable that both design and implementation of all system components adhere to a few simple and consistent concepts. With point tools, neglecting conceptual integrity may just lead to superfluous complexity and to fragile software. An entire TCAD system lacking conceptual integrity, should its implementation succeed, will very likely be unusable.

Too often, functionality is overemphasized at the cost of software quality. A typical, unfortunate implementation method is to build large software systems from existing, optimal, but independently developed components. Despite the optimality of the constituents, the integrity of the sum is violated by the multitude of concepts introduced (most often this happens without notice).

[1]A review and classification of ECAD systems is given in [5].

E. The Application Framework Architecture

The *users* of TCAD methodology are fairly satisfied when the most frequently used TCAD point tools are coupled and are accessible via a homogeneous control environment. From the prevailing application-driven perspective of TCAD, it does not matter *how* this tool integration is achieved. Hence, in most cases of existing systems, the tools used have been tied together with human efforts only, as this has been the fastest approach toward the desired integrated system.

The major aim of a *TCAD framework*, on the other hand, is not just a static integration of a (limited) set of tools, or presently increased simulation capabilities, but rather a permanent improvement of the the production process of new TCAD methods. We firmly believe that only a system with a dedicated *application-framework architecture* can reduce effort and time required for tool integration, tool implementation, and maintenance.

An application-framework architecture divides the whole system into one framework and several applications. The framework contains all reusable, generic, and persistent parts, whereas the applications contain specialized and comparatively volatile functionality. For TCAD, typical applications are classical point tools which perform specific simulations, and the framework is comprised by all integrating and supporting software and technology-independent services like visualization, graphical editing of device structures, data level implementation, and task control environment. An improvement of the TCAD methodology creation process is expected from the reduction of application complexity due to the reuse of framework parts. Unfortunately the term *framework* is too often used in TCAD to denote an integrated system where, of course, applications can be added somehow, but which does not at all exhibit an application-framework architecture. The data/tool/task/presentation layering (which is independent of the application-framework architecture), on the other hand, is commonly seen in most TCAD systems.

F. Outline

We have implemented the *Viennese Integrated System for TCAD Applications* (VISTA), an integration and development system with a dedicated application-framework architecture. The data level of VISTA was presented in an earlier paper [1]. In the remainder of this article we will describe the architectures and implementations of the user interface (Section II) and the task level (Section III) of VISTA and discuss a typical challenge of tool integration, namely inconsistencies of wafer representations in different tools and how this problem is addressed by VISTA (Section IV). Using a planarized CMOS process as an example, the cooperation of independent applications within the framework is demonstrated (Section V). A final discussion (Section VI) summarizes essential points of this paper.

II. User Interface

Independent of the type of application, users nowadays expect a comfortable presentation of all services provided by a computer system. As an accompanying effect of this trend, graphical user interfaces are judged rather by their visual appearance than by their practical utility or by their efficiency of assisting the programmer systematically in providing new functionality for the user.

In TCAD, the users' expectations include a *homogeneous* intuitive graphical user interface which covers the data level, the task level, and all integrated applications. A challenging discrepancy between the obvious demand for ease of use and the semantical complexity of TCAD tasks and tools may be anticipated.

In addition to the general need for *portability* and *comprehensibility*, a multitude of criteria apply specifically to the design of a TCAD user interface. The appearance should be extensively *configurable* to foster the acceptance of users already acquainted with a certain "look and feel" of an existing ECAD/CIM environment. The user interface must be *flexibile* enough to accommodate functional extensions for the unforeseeable variety of new TCAD capabilities. On the other hand, these extensions should not require changes in user interface principles and elements so that a casual user perceives a maximum *continuity*.

Despite claimed standards for window systems and presentation level toolkits, window system software seems to be prone to the same progress as workstation technology. For example, four rather differing window systems (VWS, DECwindows, Motif, and Windows NT) have been featured by Digital Equipment's Workstations within just five years (1989–1994). Although a reliable standard system would be highly welcomed by the application programming community, there is yet no stability in sight.

Several workstation-based systems can be found which feature the integration of multiple tools into a unified user interface, mostly based on the X Window system. PRIDE[7], based on SunView, exhibits a user interface and task level architecture which is strongly influenced by the preprocessing—computation—postprocessing task model of TCAD. SIMPL-IPX[8], based directly on Xlib, features a central interactive graphical editor which has menu-oriented facilities for running simulators. In both cases, implicit or explicit assumptions about the design cycle have had an impact on the (top-down) *design* of the software and restrict the design tasks which can be performed. A more flexible and extension-oriented user interface architecture has been accomplished in PROSE[9], which is mainly due to the consequent use of the generic Tcl interpreter[10] and Tk toolkit[11].

A. Architecture

A sufficiently standardized and reasonably well-established *de-facto* multiplatform basis used in many integrated TCAD systems is the X Window system [12], which we have chosen to build upon. The structure of the VISTA user interface is shown in Fig. 1. The bottom layer is the X Toolkit[13], an object-oriented subroutine library, designed to simplify the development of X Window applications. The X Toolkit defines methods for creating and using so-called *widgets*, which appear to the user as pop-up windows, scrollbars, text-editing areas, labels, buttons, etc. Basic functionality is provided by the

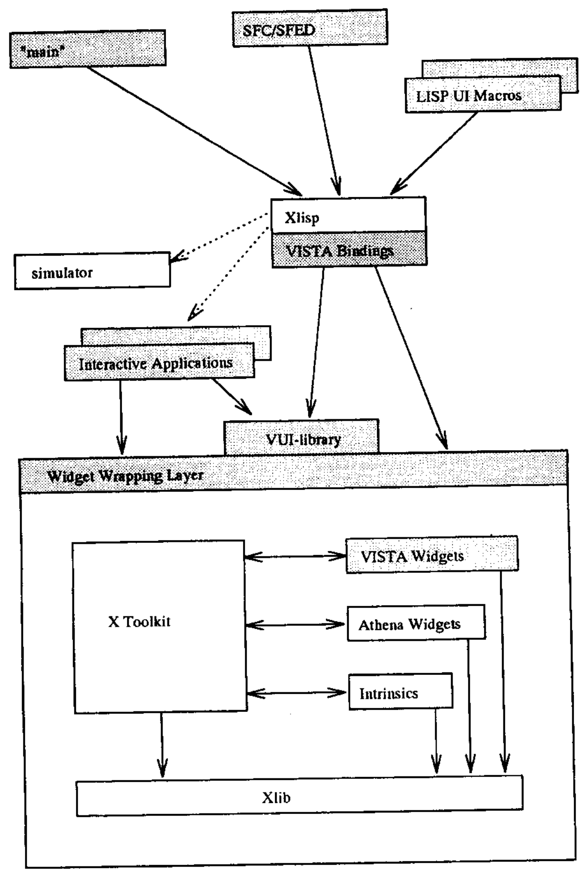


Fig. 1. Structure of the VISTA user interface and parts of the task level. Shaded boxes represent VISTA components and arrows indicate the sequence of function calls between different parts of the user interface.

generic *Athena widgets*, which are included in the MIT X11 distribution.

In addition, specialized VISTA widgets have been developed as TCAD-specific user interface elements. A widget-wrapping layer has been put on top of these widgets in order to achieve a certain widget-set independence. All widgets are created and modified via specific functions rather than via the generic interface of the X Toolkit. This should facilitate the potential migration of the user interface onto another widget set or onto any other platform similar to the X Toolkit, should the need arise.

The top layer of the VISTA user interface, the VUI (*VISTA User Interface*) library serves two purposes. It provides some often needed higher-level operations and contains most of the policy which is shared among VISTA applications. In other words, the VUI library takes care that all parts of VISTA look alike and behave similarly.

B. The VISTA Widget Set

The *Athena* widgets are a lean, generic, low-functionality widget set (they serve as an example for the usage of the X Toolkit and are part of the X release). They were originally not intended to be used in professional applications and thus do not meet the needs of a TCAD user interface, but they do provide the required basic functionality, they are highly portable, they are available on virtually every modern

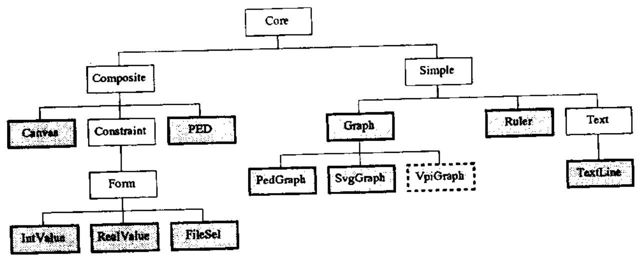


Fig. 2. The widget set used by VISTA. The VISTA extensions are shaded, the Intrinsics and Athena Widgets used for subclassing are blank.

workstation platform, and they are easy to comprehend. We have decided to use this concise and ductile widget set as base for our implementation because currently no well-established high-level standard toolkit can be found that would suffice for TCAD purposes without requiring the same significant extensions as the Athena widgets.

The VISTA widgets are subclassed from either X Toolkit Intrinsics or Athena widgets (Fig. 2). The *Canvas*, *PED*, *PedGraph*, and *Ruler* are parts of the interactive PIF Editor (PED), the *IntValue*, *RealValue*, *TextLine*, and *FileSel* are widgets for the specification of integer, real, string values, and files, respectively, and the *SVGraph* widget is a widget for displaying simple vector graphics plots.

A TCAD data level implementation would be incomplete without an accompanying interactive graphical editor for manipulating all geometric data stored in the binary PIF. The *PIF EDitor* (PED) is the front-end user interface for the interactive creation and modification of geometrical data in one, two and three spatial dimensions and of all attributes (like the material type) which define the device structures (see Fig. 14). The PED can work on all PIF files independent of their semantics. It is a generic tool for building a simulator input PIF file from scratch, for modifying existing wafer structures, and for visualizing geometric PIF information.

The PED uses the *Canvas*, *Ruler*, and *PedGraph* widgets and is implemented as a widget itself. This allows the use of multiple subwindows for editing one and the same device geometry, editing of several logical PIF files in one PED process and even using the PED as a component in "surrounding" applications. Thus, arbitrary additional menus or other widgets can be added without interfering with the PED itself. The PIF data is held in a memory-resident intermediate representation which is slightly extended with respect to the binary PIF to allow for efficient interactive manipulations. The top-level execution control of the PED is implemented as an extensible and configurable automaton which filters all user input and triggers appropriate actions.

The LISP interpreter which is used as the basis of VISTA's task level programming environment (Section III) is reused as extension language for the PED, allowing the addition of LISP-coded macros. The use of LISP as extension language for an interactive geometry editor has already proven to be a successful strategy for interactive geometrical CAD [14].

The X Toolkit and Athena widget set do not provide "classical" two-dimensional vector graphics capabilities, which are a firm requirement for any CAD discipline. To support simple platform-independent vector graphics output we have implemented a minimum-functionality vector graphics widget (*SVGraph* Fig. 2, Fig. 13) which is built directly on the generic Xlib and X Toolkit. The widget remembers all drawing commands and provides zoom and pan functions for the user, which henceforth, the programmer does not need to bother with. Callbacks can be utilized, for example, to digitize data points. This widget is used as an interactive interface to VISTA's visualization library.

C. The VUI (VISTA User Interface) Library

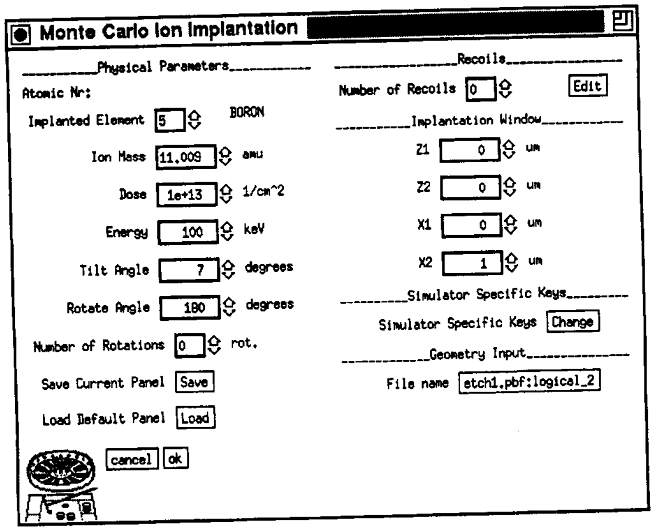
The VUI library contains functions which create often-used combinations of several widgets in one step, arrange them, and set up all required connections and callback functions. These widget aggregates behave as if they were single composite widgets and are indistinguishable from the user's point of view. This is similar to the OSF/Motif "Convenience Function" concept [15], and helps to maintain a unified layout and behavior for all VISTA applications.

The VUI library currently defines two types of main widget arrangements, the chat shells and the dialog shells. Chat shells are used for longer lasting user interactions and contain a standardized menu bar with pulldown menus. Dialog shells are for short (typically pop-up) inquiries and contain a button box on the bottom for alternative answers. These two types of widget arrangements are commonly found among most window systems and applications. Specialized dialogs like the selection of logical PIF files are supported by single VUI functions.

D. Presentation Level Interfaces

The presentation level must allow access to all data, tools, and tasks. Hence interfaces between the presentation level and data, tool, and task levels must exist. A common architectural feature found among many TCAD systems is the tight connection between user interface and task level environment.

In VISTA, the connection between user interface and task level environment (detailed in Section III-C) is implemented by a full-function VUI and widget wrapper programming interface (denoted "VISTA Bindings" in Fig. 1) for the XLISP interpreter (XLISP is the basis of VISTA's task level). Using these function bindings, the task level interpreter serves as implementation environment for the presentation level integration of all batch-mode tools that don't need user interaction and are accessed by the user indirectly via the task level. The "LISP UI Macros" in Fig. 1 are user interface code which is executed by the XLISP interpreter and, e.g., produces tool control panels, either for direct invocation or for editing tool parameters via the VISTA Simulation Flow Control (SFC) module. An example tool control panel is shown in Fig. 3. Most tool control panels are created from formal specifications of the tools by simple interface generators implemented in LISP. This relieves the application engineer from the need to use low-level X Toolkit programming to create new tool control panels. The separation of the user interface code from the (rather complex) tool itself is an important means to stabilize and unify the


Monte Carlo Ion Implantation

Physical Parameters	Recoils
Atomic Nr:	Number of Recoils 0 Edit
Implanted Element 5 BORON	Implantation Window
Ion Mass 11.009 amu	Z1 0 um
Dose 1e+13 1/cm^2	Z2 0 um
Energy 100 keV	X1 0 um
Tilt Angle 7 degrees	X2 1 um
Rotate Angle 180 degrees	Simulator Specific Keys
Number of Rotations 0 rot.	Simulator Specific Keys Change
	Geometry Input
Save Current Panel Save	File name etch1.pbf:logical_2
Load Default Panel Load	

Fig. 3. Widget aggregate (tool control panel) for editing the parameters of a PROMIS Monte Carlo ion implantation step.

user interface behavior and to relieve the application from the burden of the user interface implementation.

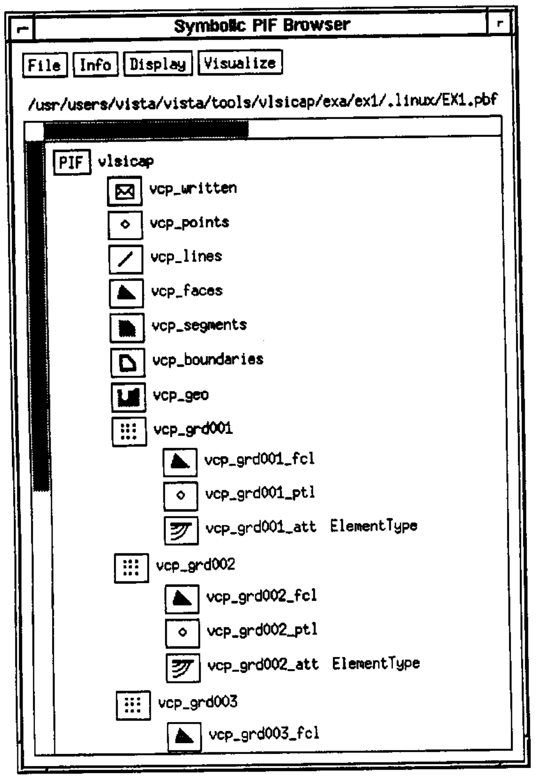
Interactive applications, like visualization clients or the PED are considered part of the presentation level and access the VUI library and widgets directly (the user interface is compiled and linked).

The presentation/data level interface of VISTA is essentially comprised by the PED, the PIF browser, and the visualization. The front-end data interface PED (described above) focuses at editing geometries contained in the data level and is implemented (in C) as a widget. The symbolic PIF browser (see Fig. 4) presents the logical structure and hierarchy of the PIF file in iconic form and allows the inquiry and selection of PIF objects. It is used as a generic intuitive facility in applications which require PIF object selection. The PIF browser is implemented as a LISP program which is executed by the task level shell.

VISTA's built-in visualization is the back-end interface between presentation level and data level. The visualization is organized as a modular library working on purely geometric data consisting of sets of primitive geometrical objects (so called "simplexes"). A major consideration for this design was to provide a sufficient spectrum of simple interfacing options to export visualizable data for other (preferred) visualization systems. The visualization of simulation data is performed by translating the PIF into a simplex representation, successively applying operations such as isosurface extraction, cutting, slicing, projection, coloring, and by displaying the resulting plot with the *SVGraph* widget. Results of this visualization procedure can be seen in Figs. 12 and 13.

III. Task Level

For a TCAD task level environment, proper design and choice of an implementation platform are not trivial, but become clearer when the proposed qualitative requirements are remembered. A UNIX- (or any other operating system) shell based solution does not fulfill the portability requirement, whereas the use of an integrating interactive master application



```
/usr/users/vista/vista/tools/vlsicap/exa/ex1/.linux/EX1.pbf
PIF vlsicap
      vcp_written
      vcp_points
      vcp_lines
      vcp_faces
      vcp_segments
      vcp_boundaries
      vcp_geo
      vcp_grd001
          vcp_grd001_fcl
          vcp_grd001_ptl
          vcp_grd001_att  ElementType
      vcp_grd002
          vcp_grd002_fcl
          vcp_grd002_ptl
          vcp_grd002_att  ElementType
      vcp_grd003
          vcp_grd003_fcl
```

Fig. 4. The symbolic PIF browser

(like a device editor), which is sometimes found in TCAD systems, does not offer enough flexibility for task definition and tool access.

Looking at existing systems in CAD areas, it seems obvious that an interpreter is the classical approach for the task level environment ([2], [5], [9], [14], [16]–[18]). Both single interactive actions and more complex flow control can be executed in an interpreting environment.

A conceivable, rather modern alternative is an entirely object-oriented concept where both CAD data and simulation tools are objects (as in [19]) and the task flow to be performed is deduced from their properties and relationships and by methods and rule inference from a design goal or inquiry. At a closer look, however, the two concepts are not mutually exclusive. In fact, the use of an interpreter is merely an implementation-oriented architectural choice which does not at all preclude the later introduction of object-oriented concepts.

Especially for Technology CAD, the immediately available programming language features offered by an interpreter are of great value for defining task level macros and arbitrarily complex control flows which involve the execution of several applications, without the need to construct complex object systems before any practical problems can be solved.

A. XLISP

Although there are many TCAD systems which successfully use Tcl/Tk [10], [11] as task level interpreter and user interface (see, e.g., [16], [17], [20], [21]), VISTA's task level environment is predominantly based on XLISP [22], a comprehensible and compact LISP interpreter. The highly portable C source code of this public-domain product is freely available. It can be extended and customized for TCAD purposes by both adding C-coded primitives and by loading LISP code at run time.

The appropriateness of this choice is also confirmed by several other remarkable, LISP-based implementations of task level environments in related fields. Examples are the well-known *GNU Emacs* [23] text editor which uses LISP as extension and top-level implementation language, the generic CAD system *AutoCAD* [14] which derives much of its success from third-party applications implemented in the SCHEME-like extension language *AutoLISP*, or *Winterp* [24] ("*W*idget *Interp*reter"), an experimental user interface prototyping environment which is part of the *MIT* X11 distribution. Other integrated CAD systems which allow for the definition of complex, data-driven control flows often use LISP as major implementation language [19], [25].

A required task-level (LISP) function can be implemented as

- a LISP coded function loaded at run time;
- a C coded function which is linked with the LISP interpreter; or
- an external application, implemented as a separate executable.

Whatever implementation method is used does not make a difference at the interpreter level, the operation (the task atom) is always presented as a single LISP function.

B. Task Level Interfaces

For all batch mode tools (these are applications that do not require user interaction) the task level interface and the *presentation integration* (according to [26] and [27], that is the provision of applications with a homogeneous user interface) are implemented within a single context. LISP functions are used to create a layer of *virtual applications* on the task level for one or more physical applications (executables). Fig. 5 shows two characteristic examples. The user interface for these virtual applications is implemented in the task level programming environment, leaving the physical applications entirely unaffected.

An example for a one-to-many mapping ("Application A" in Fig. 5) is a device simulator, where for every desired type of device characterization a function that performs the required simulation step and extracts the requested parameter is implemented ("Virtual Applications W" and "X" in Fig. 5). For process simulation the situation is inverted. Several physical applications are run consecutively under control of a single presentation (VISTA's *Simulation Flow Controller* [28]).

This kind of presentation level/tool level mapping is *only* possible with a dedicated task level that separates the applications from the presentation level. This is both an indispensible prerequisite for the integration of "foreign" tools (the source code often can not, or should not be changed), and a smart strategy to strengthen continuity and portability of the tool / user interface. Furthermore, the intermediate task level layer allows the user interface to be tailored to *design tasks* instead of *application peculiarities*.

Relationships between tools, user interface code, and task level interpreter can be seen in Fig. 1. The *XLISP* interpreter

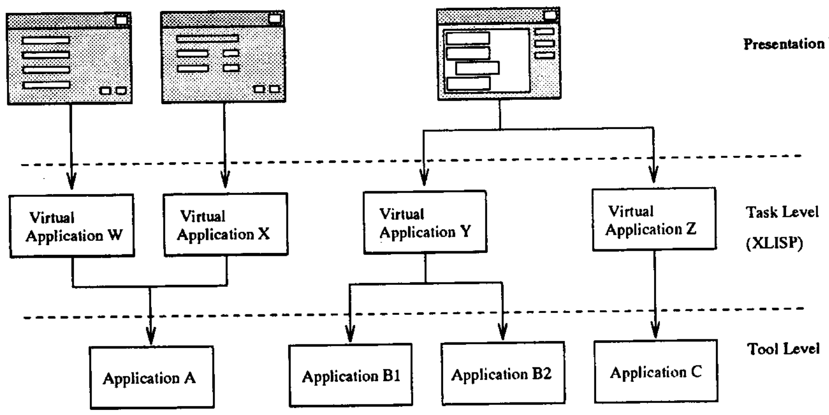


Fig. 5. A set of physical applications is mapped to a set of virtual applications on the task level.

```
(defun pbfm (&rest args)                          ; define function pbfm
  (let ((cmdline (apply #'vos::cmd-args args))) ; convert argument list
    (vos::run                                     ; run subprocess
      (vos::create-vospec :name "pbfm")           ;   symbolic executable "pbfm",
      cmdline                                     ;   argument list,
      :bg T)))                                    ;   as parallel subprocess
```

Fig. 6. LISP interface code defining the virtual application *pbfm* (the task level view of the PIF Binary File Manager).

```
(vui::create-editable-file            ; creates a popup file selection dialog
  vui::current-menu-button            ;   within a pulldown menu
  "PIF->PBF"                          ;   using this button label
  (vos::get-curdir)                   ;   starting in the current directory
  "*.pif"                             ;   symbolic wildcard
  xvw::ALLOW-EVERYTHING               ;   allowing change of directory and such
  #'(lambda (wdg cld cad)             ;   callback function
      (pbfm 'a 'k                     ;     runs pbfm with certain arguments
       (vui::get-file-value wdg)))    ;     on the selected file
  nil)                                ;     unused client data
```

Fig. 7. LISP user interface macro defining a file selection dialog for running the virtual application pbfm on a selected ASCII PIF file.

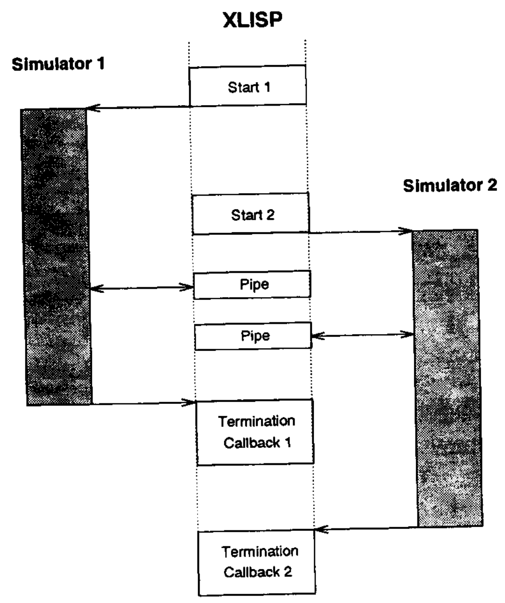


Fig. 8. Timing diagram of two (independent) simulations which are run in parallel under control of the XLISP interpreter.

runs the simulator as parallel subprocess. The definition of the respective virtual application(s) and graphical interfaces are contained in *LISP Macros* (entirely decoupled from the actual tool code), which are loaded into and interpreted by *XLISP*.

Fig. 6 shows the LISP interface code which implements the tool level / task level interface for the PIF Binary File Manager (PBFM, [1]). Fig. 7 lists the LISP code which uses a VUI dialog creation function *vui::create-editable-file* to define a pop-up dialog which will call the function *pbfm* (a virtual application) when the user confirms the selection of a file matching the (system-independent) symbolic wildcard "**.pif*". All *vos::* functions are part of an operating system layer which is used to encapsulate system-dependent features and to allow fully system-independent programming.

C. Implementation Details

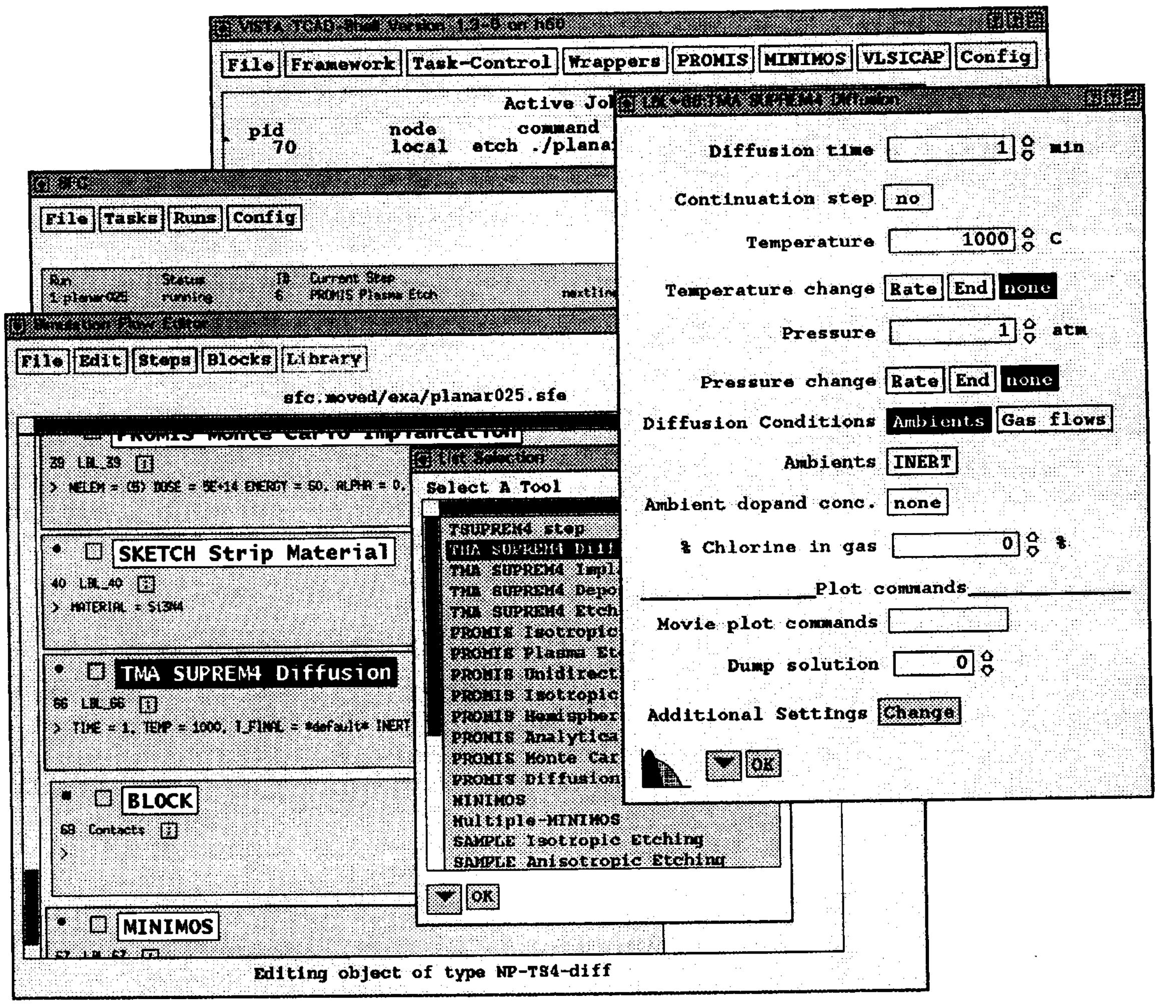
The object-oriented callback concept of the X Toolkit has been generalized in a very straightforward manner and successfully applied to those parts of the TCAD framework where a strict decoupling of functional modules and high flexibility of the control flow is desirable. It is obvious that this is of special value for a flexible task level implementation.

Events coming from the X Window system are passed to the XLISP interpreter. If a LISP expression has been associated with the activated widget at creation time, this expression is then evaluated by the interpreter and can be used to change parameter values, trigger the execution of a simulator or start the evaluation of a LISP program.

The same callback concept is also used for the control of simulator execution. If a simulation tool terminates, it signals the termination to the parent process, which again causes an associated callback expression to be evaluated. Callbacks can be triggered by the user interface, error handler, network layer, or by the termination of child processes.

Many computationally extensive design tasks, like statistical computations, exhibit intrinsic parallelism and can hence be effectively and easily parallelized using the callback technique. Whenever there is no data dependency (which is the case for, e.g., stationary multiple operating point analysis) the simulations can be done in parallel and "only" need to be distributed on different workstations or servers and synchronized at the end. Fig. 8 shows two (simulation) processes which are run in parallel. The termination of simulation process 1 causes the termination callback 1 to be executed and can be used to trigger the computation of the next step.

To preserve the simplicity of XLISP and in order to provide a homogeneous procedural interface and programing environ-



```
Active Jo
 pid        node     command
  70        local   etch ./plana

Run           Status     ID   Current Step
1 planar025   running    6    PROMIS Plasma Etch        nextli

sfc.moved/exa/planar025.sfe

PROMIS Monte Carlo Implantation
39 LBL_39
> NELEM = (5) DOSE = 5E+14 ENERGY = 50. ALPHA = 0.

SKETCH Strip Material
40 LBL_40
> MATERIAL = Si3N4

TMA SUPREM4 Diffusion
66 LBL_66
> TIME = 1. TEMP = 1000. I_FINAL = *default* INERT

BLOCK
68 Contacts
>

MINIMOS
67 LBL_67

Editing object of type NP-TS4-diff

Select A Tool
TSUPREM4 step
TMA SUPREM4 Diff
TMA SUPREM4 Impl
TMA SUPREM4 Depo
TMA SUPREM4 Etch
PROMIS Isotropic
PROMIS Plasma Et
PROMIS Unidirect
PROMIS Isotropic
PROMIS Hemispher
PROMIS Analytica
PROMIS Monte Car
PROMIS Diffusion
MINIMOS
Multiple-MINIMOS
SAMPLE Isotropic Etching
SAMPLE Anisotropic Etching

TMA SUPREM4 Diffusion
        Diffusion time  [     1]  min
     Continuation step  no
           Temperature  [  1000]  C
    Temperature change  Rate  End  none
              Pressure  [     1]  atm
       Pressure change  Rate  End  none
  Diffusion Conditions  Ambients  Gas flows
              Ambients  INERT
  Ambient dopand conc.  none
      % Chlorine in gas [     0]  %
____________Plot commands____________
   Movie plot commands  [      ]
         Dump solution  [     0]
   Additional Settings  Change
```

Fig. 9. Screen dump of a VISTA session with the simulation flow editor, the simulation flow control module and the VISTA main panel.

ment, we had to implement the X Window interface (*VISTA UI Bindings* in Fig. 1) for XLISP from scratch. As there are other C-coded parts of the framework which need to be accessible on the extension language level, a generic, automatic method, for linking given C functions with the XLISP interpreter has been implemented which creates the major part of the task level interface code. This so-called *Tool Abstraction Concept* (TAC) yields highly homogeneous and coherent code and relieves the provider of such functions from tedious and error-prone programming work.

In addition to the reuse in the PED, the XLISP interpreter is also used as basis for VISTA's integrated make utility *Vienna MAKE* (VMAKE). The TAC and other code generators which facilitate application and framework implementation are controlled by VMAKE. VMAKE parses source code to verify certain coding and documentation conventions (like, e.g., the proper use of VISTA's global error system) and to extract dependency information.

D. Simulation Flow Control

The *Simulation Flow Control* (SFC) module (presented in greater detail in [28]) of VISTA is a high-level utility of the task level responsible for the definition, management, and execution of simulation sequences. The main field of application is the reproduction of fabrication processes which usually involves several independent process simulators. The task flow is stored in a simulation flow description using symbolic names to call virtual applications. Process flows are assembled from single tasks or from predefined sequences comprised of several steps by means of an interactive graphical *Simulation Flow Editor* (SFED).

Fig. 9 shows a typical VISTA session where the SFED is used to edit the process flow. The parameters of the last step of an example simulation flow presented in Section V (using a TSUPREM-4 wrapper) is being edited. The parts of the user interface shown are all created and executed by the task level shell.

The SFC stores, upon request, the results of intermediate steps for later analysis. The task functionality of the SFC include dependency analysis, interfaces for process parameter variation, and lot splits. The SFC is entirely coded in LISP (cf. "SFC/SFED" in Fig. 1) uses the callback concept for simulator synchronization, and is executed by the XLISP interpreter.

IV. Tool Integration

The main motivation for TCAD systems is the integration of different simulators. It is of course always possible to integrate another tool into an existing set of tools, the decisive criterion to assess the support offered by the framework is the initial effort and the maintenance effort involved in the integration. Another criterion for a proper application framework architecture is whether adaptions of already integrated tools are required when other, new tools are integrated or changed.

A. Generic Services, Tool Focus, and Inter-Operability

From a software point of view, a significant amount of TCAD methodology is technology-independent. This generic functionality must be provided as part of a framework, usually in form of libraries and applications. The scope of these services ranges from integrated CASE tools for application development, scientific visualization, interactive geometry editing facilities, to more traditional, "enabling TCAD methodology" like grid generation, interpolation, and many more.

There are many situations in TCAD, especially when coupling different simulators, where interesting semantic problems occur on the data level. Among the most intriguing challenges are potential inconsistencies between the geometry of the simulation domain and one or more grids with attributes defined on them, altogether describing the current state of a wafer subdomain.

When simulators are allowed to use their own wafer state abstraction (which must be done to let the tools focus on their specific task and to facilitate the integration of foreign tools), they will produce and affect only the data relevant to the specific problem they are modeling. Hence, certain inconsistencies are **inevitable**, and are not a consequence of misbehaving applications or of an insufficient data representation or architectural concept.

In addition to a standardized representation [1], additional functionality is required to map the different tool-specific wafer models to a common, unabridged but concise inter-tool wafer view, in order to bridge the gaps between simulators. The *framework* must provide these means for *resolving* conflicts and inconsistencies so that simulators can work together constructively without having to care for potential grid- and geometry-related conflicts they may create.

B. Some Common Problems

- **Problem 1.** Tool A produces a result on a nongeometry-conforming grid and tool B needs a geometry-conforming grid as input.
- **Problem 2.** Tool A produces a result on a single grid that spans multiple segments (with different materials) and tool B needs one grid for each segment.
- **Problem 3.** Tool A creates a result attribute that must be merged with an existing attribute which describes the previous wafer state (e.g., additional Boron doping by ion implantation) to produce a new, valid wafer state. Should the old or the new grid be used to represent the superposition of the attributes? This choice of the target grid is nontrivial. Especially when different spatial regions are affected, a grid-merge is desirable.
- **Problem 4.** (A variation of problem 3) Tool A creates a result attribute that must be merged with the wafer state on a grid type different from the wafer state grid.
- **Problem 5.** Tool A alters the geometry of the wafer state, but does not update the attributes and grid defined on the altered geometry.
- **Problem 6.** The wafer state is defined for a much larger area (and so are grids and attributes) than the subdomain that shall be simulated by tool A. A subgeometry (a single segment or a rectangular subdomain) is fairly easily constructed, but the grid and attributes on this subdomain may be required as input for the tool.

There are *many* more grid-related problems and conflicts that do arise when multiple state-of-the-art simulation tools are used to simulate practical device fabrication steps. Some of these problems can be solved by interpolation services. The problems 1–6 listed above, however, can not be solved satisfactorily by interpolation alone. Moreover, the continued interpolation before and after each simulation step is a dangerous sink of accuracy and should be avoided when feasible alternatives exist.

C. The VORONOI Re-gridding and Interpolation Service

The set of different problems listed above can be summarized in the following generic problem statement.

Given a geometry consisting of multiple segments, a set of grids of different types with or without spatial overlap which do not necessarily conform to the geometry, and for every grid one or more attributes of different types defined on it, create a set of segment-conforming grids, one grid for every segment, with one attribute for every attribute type that is (re-)constructible for that segment. Grid points should be reused to avoid unnecessary interpolation.

To solve this generic problem rigorously, a framework tool (VORONOI [29]) has been implemented. The generic re-gridding problem can be substructured in several smaller (also very generic) problems which lead to fairly independent functional modules that perform operations on common data.

The key idea to overcome any of the grid/geometry consistency problems rigorously is to treat grid and geometry points equally by merging the set of geometry points and the set of grid points and by finally *(re-)triangulating* the resulting *point cloud* using a *constrained Delaunay triangulation.*

The basic re-gridding algorithm is as follows.

1) Read all geometry points, edges, and segments. Read all grid points, but ignore the mesh.
2) **Refine** geometry edges by inserting additional points, so that the resulting, refined geometry edges are Delaunay edges and will be produced by the subsequent triangulation.
3) Remove all edge information. Perform a **Delaunay Triangulation** of the resulting point cloud.
4) **Partition** the resulting triangular grid into segment-conforming partial grids.
5) Interpolate missing attribute values.

Note that by adding only boundary refinement points to the input grid point cloud, any excess interpolation inside the segments is avoided. Due to the grid merge and refinement step some of the target grid points have unknown attribute values. In the **Output** step, these missing values are constructed

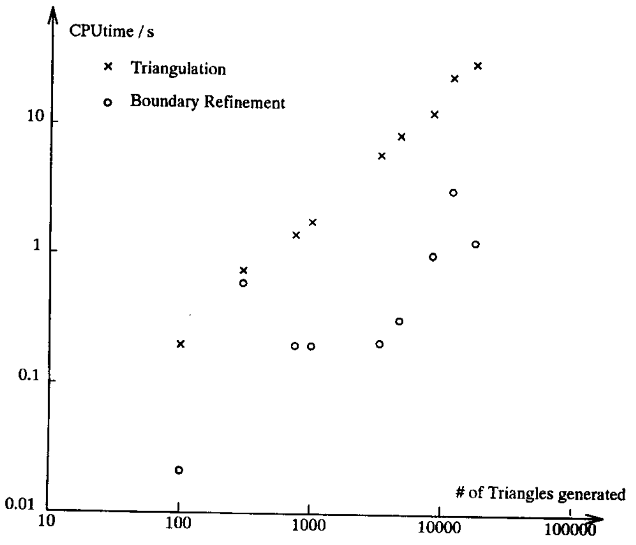


Fig. 10. CPU time measured for boundary refinement and triangulation as a function of the number of created triangles.

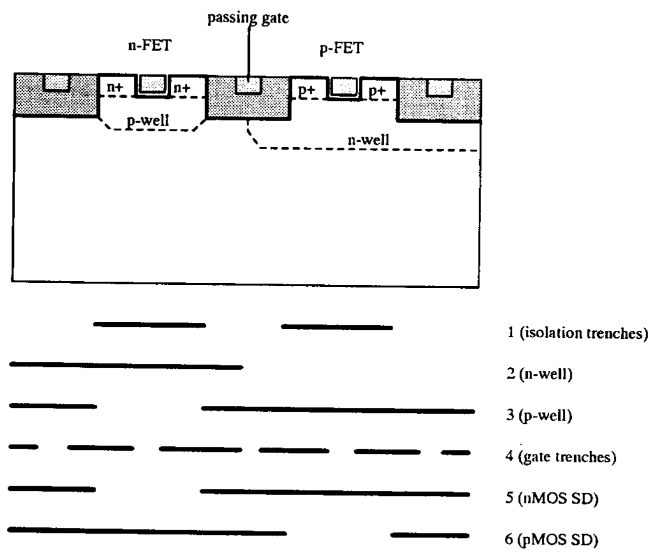


Fig. 11. Final CMOS structure and symbolic mask information

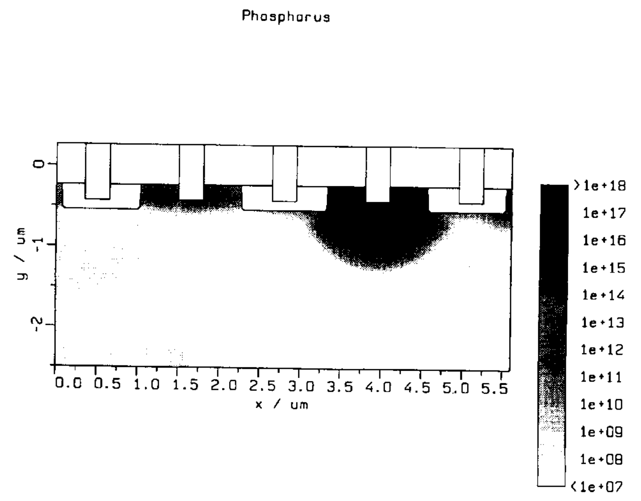


Fig. 12. N-well doping profile after etching the trench for the planarized poly gate (result of step 24).

from all known attribute values by solving either the Laplace equation or the biharmonic equation on the merged grid, where all grid points with known values serve as Dirichlet "boundary" condition.

For the actual Delaunay **Triangulation** step, a divide-and-conquer algorithm[30] has been implemented, extended by the robust treatment of important special cases, like regular (tensor product) subgrids.

All search and point location operations performed by VORONOI during the **Boundary Refinement** and **Triangulation** step are based on a so-called *bucket point quadtree*[31], which is responsible for the efficiency $O(N \log(N))$ of all critical steps (where N is the total number of final vertices).

Fig. 10 shows the measured performance of the computationally expensive steps boundary refinement and triangulation. Realistic input data with a variety of geometries and input grid point clouds have been used. CPU time measurements have been performed on a PC 486 (66 MHz, 16 MBytes RAM), Linux, GNU C compiler. In all practical cases, the total CPU time for re-triangulation is approximately one to two orders of magnitude less than the CPU time for numerical simulation steps such as etching, deposition, diffusion, Monte Carlo implantation, or oxidation.

V. Tools and Framework Applied

A major challenge of multitool TCAD lies in the robust reproduction of increasingly complex manufacturing processes. This challenge is particularily prevalent in fabrication processes with a tight interaction between structuring and doping techniques, like planarization and trench isolation processes. A fully planarized, trench-isolated 0.25 μ m CMOS has been presented by Wen *et al.* [32], [33] and is used here to demonstrate the coupling of several specialized process simulation tools by means of the VISTA framework. These point tools have been developed independently, focus on different simulation problems and use entirely different abstractions and internal representations of the wafer state.

The Monte Carlo simulation module of PROMIS [34], [35] is used for the simulation of all ion implantation steps. It is a traditional FORTRAN simulator and has been integrated into VISTA directly using the PIF Application Interface (PAI) [36]. It reads only the device geometry and produces the resulting doping profiles on a single, nongeometry-conforming tensor product grid which covers the bounding box of the entire device geometry.

TSUPREM-4 [37] is used to simulate the diffusion steps only. It has been integrated by means of a wrapper which converts the wafer state from VISTA PIF to TIF (TMA's Technology Interchange Format) and vice versa. TSUPREM-4 requires and produces doping information on several boundary-conforming triangular grids, one for each geometry segment.

The simulator ETCH [37], newly developed using VISTA as implementation basis, utilizing high-level libraries [38], is used for the simulation of etching and deposition steps. It reads only the geometry and produces a new, modified geometry. As ETCH uses a cellular data model to perform purely geometric

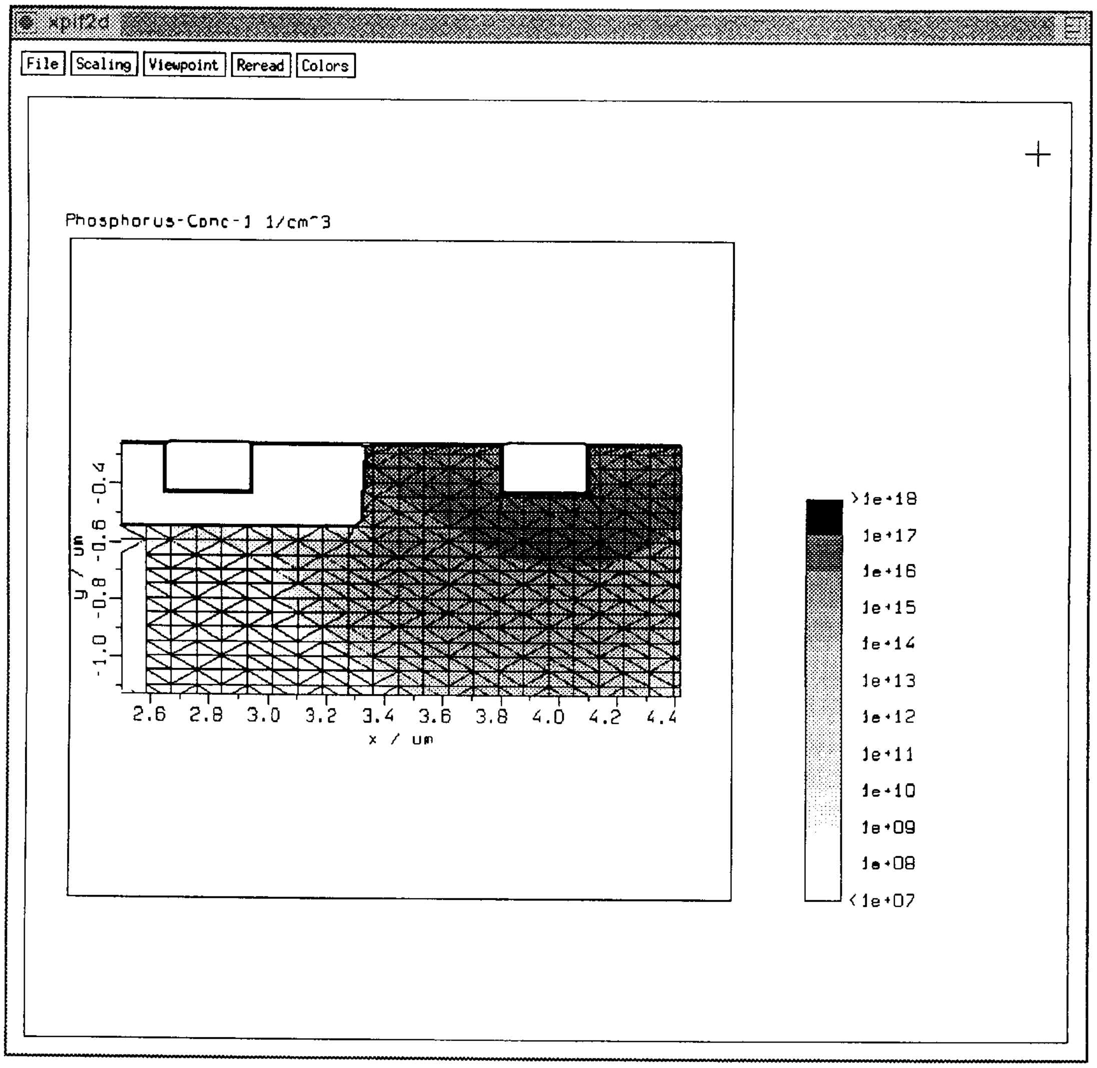


Fig. 13. xpif2d showing a detail of the n-well and a nearby passing gate structure (result of step 28).

operations, it can not change the wafer state mesh accordingly. Hence, after etching and deposition steps, the (old) wafer state mesh does not match with the new geometry produced.

Due to the current lack of an integrated lithography simulator, the auxiliary tool SKETCH is used to define simple manhattan wafer geometries as an unphysical emulation of spin-on, exposure, and mask strip steps.

The VORONOI re-gridding and interpolation service is used by the simulation flow control (SFC) module to achieve and maintain consistency between grid and geometry, and to convert the structural and doping information between tool-specific and wafer-state compliant forms.

Table II shows the simulation steps executed by the SFC. This "virtual process" is a simplification of the real fabrication process. The virtuality of the simulation enables us to, e.g., "expose and develop" a nitride mask directly (step 10). The desired final device structure is shown in Fig. 11, together with the 1-dimensional (cross-sectional) mask information used in steps 3, 11, 16, 22, 30, and 35.

Under control of the SFC, VORONOI is automatically called after each step which modifies the geometry only without consistently adapting the grid and after each step that adds a new grid to the current wafer state. In the practical application, VORONOI is run even more often in order to maintain a consistent wafer state. Compared to the effort of the actual simulation steps, the CPU time required for re-triangulation and interpolation is negligible.

Fig. 12 shows the phosphorus doping of the n-well after the trenches for the MOS gates and for the passing gates have been etched. This etch step has removed part of the geometry at the wafer surface. The doping in the void has been automatically removed by VORONOI and the grid has been made geometry-conforming to produce a consistent wafer state after completion of step 24.

Fig. 13 is a screen dump of the xpif2d visualization client showing the final planarized geometrical structure of the nMOS transistor along with the phosphorus concentration (after step 28). The visualization uses the vector graphics widget of the VISTA user interface described earlier.

The PED (PIF editor) is used to visualize and analyze the triangular grid created by VORONOI and the geometry of the

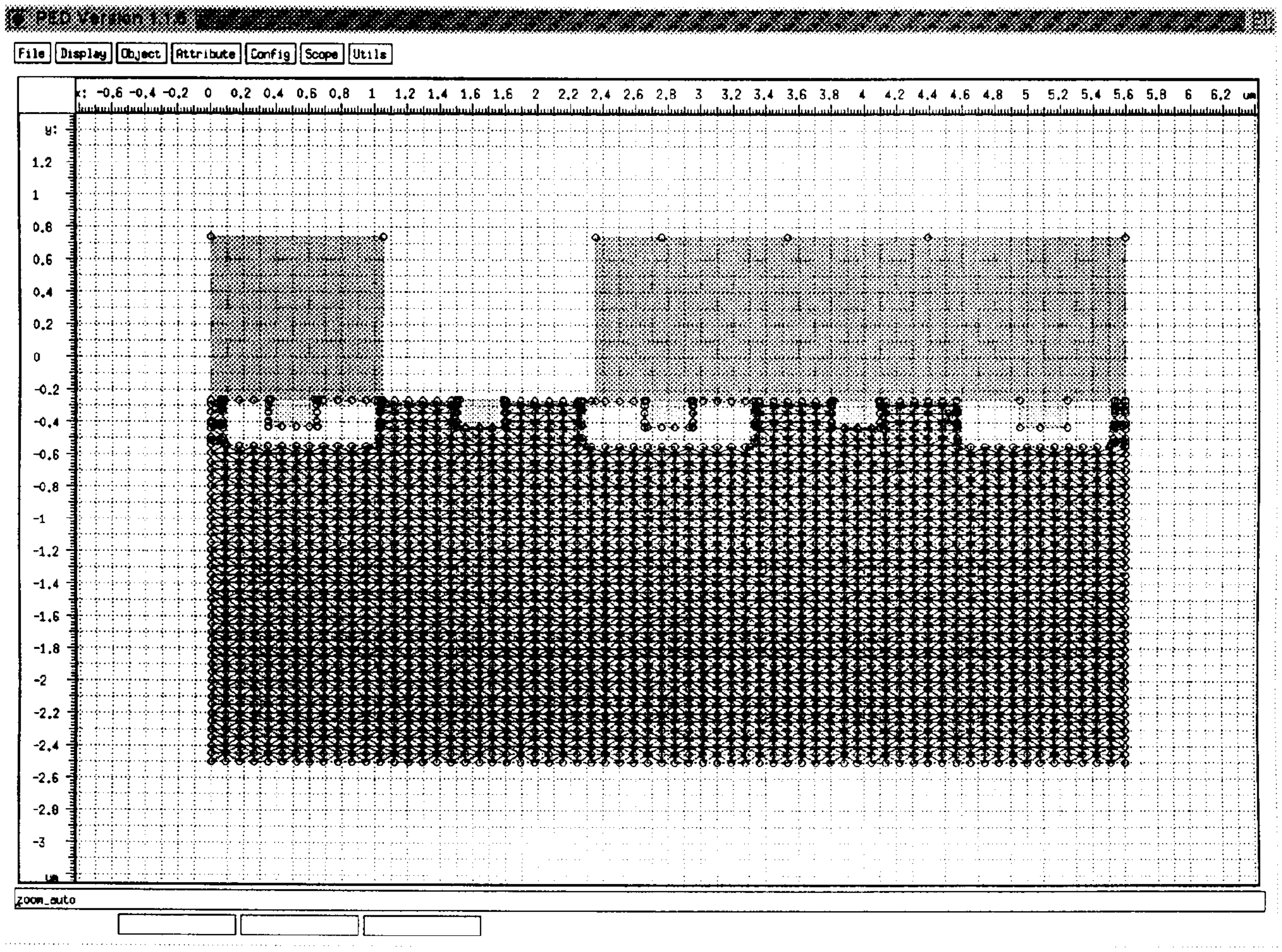

PED Version 1.1.0
File
Display
Object
Attribute
Config
Scope
Utils
x: -0.6 -0.4 -0.2 0 0.2 0.4 0.6 0.8 1 1.2 1.4 1.6 1.8 2 2.2 2.4 2.6 2.8 3 3.2 3.4 3.6 3.8 4 4.2 4.4 4.6 4.8 5 5.2 5.4 5.6 5.8 6 6.2 um
y: 1.2 1 0.8 0.6 0.4 0.2 0 -0.2 -0.4 -0.6 -0.8 -1 -1.2 -1.4 -1.6 -1.8 -2 -2.2 -2.4 -2.6 -2.8 -3 um
zoom_auto

Fig. 14. PED showing the entire CMOS structure with the grid on the silicon segment, triangulated by VORONOI (result of step 31).

CMOS structure after step 31. A screen dump is shown in Fig. 14.

VI. Discussion

A. The Application Framework Architecture

It is only due to the dedicated application-framework architecture that the simulation tools integrated into a TCAD system can still focus on a specific technological problem and even use entirely different methods and internal problem representations. The rapid evolution of these simulators is driven by fabrication technology. The *generic*, technology-independent functionality of the framework helps to bridge the gaps between the simulators and provides the device and process engineer with a more stable, homogeneous representation of simulation capabilities.

Whereas the users of an integrated TCAD system will not necessarily notice architectural shotcomings of the software, both tool developer and tool integrators will experience the retarding effect of the (prevailing) application-driven TCAD system approach. It is hence vital for the TCAD framework that it addresses properly not only the application of methodology, but also the entire creation process of TCAD methodology. This definitely includes support for tool integration and tool development. Due to its generic functionality and indispensable rigorousness such a TCAD framework is inevitably large and complex. This functionality-induced complexity can only be counteracted by qualitative demands, like conceptual integrity.

B. VISTA

With VISTA we have put emphasis on these qualitative demands for all design and implementation decisions. Platform independence has been one of the major design requirements for VISTA. Table III shows a list of currently supported computer architectures and operating systems. The only requirement for porting VISTA to another modern computer platform is the existence of close-to-ANSI C and FORTRAN compilers, and the X Toolkit. Common operating system-dependent features like pipes are wrapped in an operating system layer.

The data level [1], [38] (including high level services such as VORONOI [29], the user interface, and the task level [4] altogether present a considerable amount of technology-independent functionality. To make this system more comprehensibe for methodology creators (tool developers and tool integrators), we have tried to maintain conceptual integrity by favoring the generalization of existing concepts over the introduction of new ones.

To achieve the framework functionality provided by VISTA, it would have been much easier and faster to glue together

TABLE II
PROCESS SIMULATION FLOW FOR THE $0.25\mu m$. FULLY PLANARIZED, SHALLOW TRENCH ISOLATED CMOS PROCESS

step	simulator	SFC operation	parameters
1	SKETCH	create-subdomain	"Si", $5.6 \times 2.5\mu m$
2	SKETCH	spin-on	"Resist", $0.2\mu m$
3	SKETCH	expose	Mask 1
4	SKETCH	strip-material	"Exposed"
5	ETCH	plasma-etch	misc. etch rates, 320 seconds
6	SKETCH	strip-material	"Resist"
7	ETCH	iso-depo	"SiO2", 60 seconds at $125nm/s$
8	SKETCH	spin-on	"Resist", $0.25\mu m$
9	ETCH	iso-etch	misc. etch rates, 330 seconds
10	SKETCH	spin-on	"Si3N4"[2], $1\mu m$
11	SKETCH	expose	Mask 2
12	SKETCH	strip-material	"Exposed"
13	PROMIS	monte-carlo-implant	"Phosphorus", $5 \cdot 10^{11}/cm^2, 60keV$
14	SKETCH	strip-material	"Si3N4"
15	SKETCH	spin-on	"Si3N4"[3], $1\mu m$
16	SKETCH	expose	Mask 3
17	SKETCH	strip-material	"Exposed"
18	PROMIS	monte-carlo-implant	"Boron", $5 \cdot 10^{11}/cm^2, 60keV$
19	SKETCH	strip-material	"Si3N4"
20	TSUPREM-4	diffusion	inert, 3.5 minutes at 1000°C
21	SKETCH	spin-on	"Resist" $0.5\mu m$
22	SKETCH	expose	Mask 4
23	SKETCH	strip-material	"Exposed"
24	ETCH	plasma-etch	misc. etch rates, 200 seconds
25	SKETCH	strip-material	"Resist"
26	ETCH	iso-depo	"SiO2", 60 seconds at $1.1nm/s$ [4]
27	SKETCH	spin-on	"Poly", $0.25\mu m$ [5]
28	ETCH	iso-etch	misc. etch rates, 310 seconds
29	SKETCH	spin-on	"Si3N4", $1\mu m$
30	SKETCH	expose	Mask 5
31	SKETCH	strip-material	"Exposed"
32	PROMIS	monte-carlo-implant	"Arsenic", $2.5 \cdot 10^{14}/cm^2, 33keV$
33	SKETCH	strip-material	"Si3N4"
34	SKETCH	spin-on	"Si3N4", $1\mu m$
35	SKETCH	expose	Mask 6
36	SKETCH	strip-material	"Exposed"
37	PROMIS	monte-carlo-implant	"Boron", $2.5 \cdot 10^{14}/cm^2, 30keV$
38	SKETCH	strip-material	"Si3N4"
39	TSUPREM-4	diffusion	inert, 1 minute at 970°C

TABLE III
SUPPORTED HARDWARE PLATFORMS AND OPERATING SYSTEMS

computer	operating system
DEC AXP 3000,7600	Open VMS 1.5
DEC AXP 3000	OSF 1.3
Apollo DN10000	Apollo DOMAIN 10.3
Decstation 3000,5000	Ultrix 4.2, Ultrix 4.3
HP/Apollo 9000/700	HP/UX 8.05, HP/UX 9.0
IBM RS6000	AIX 3.1, AIX 3.2
PC 386, PC 486	Interactive Unix 3.0
PC 386, PC 486	Linux 0.99pl13
Sparc Station	SunOS 4.1
VAX, VAXstation	VAX/VMS 5.5

well-established and near-optimal partial solutions. We could, e.g, have used GNU Make [39] as configuration management utility, Tcl [10] for the task level, and C++ for an object-oriented data level interface. Besides exposing the system to uncontrollable version changes and potential portability problems, the resulting system would use several incoherent concepts where a single concept would have sufficed. For example, we have initially choosen LISP as task level language as it conforms to the PIF. This conceptual conformity opens up potential features such as the storage of task level objects in the PIF database or the direct manipulation of PIF data on the task level. The XLISP interpreter which forms the basis of the task level has been reused for the PED's extension language and for the implementation of VISTA's integrated make utility VMAKE, instead of integrating another ready-to-use component. We have decided to build directly on the X Toolkit and Athena widgets not only for portability reasons, but also because the callback concept can be generalized and reused for the control flow on the task level and in many other places.

One may contend that TCAD has reached a software complexity where only a thorough understanding and a consequent support of the methodology creation process can mitigate a latent crisis. We believe that the architecture and facilities of VISTA are a small step into that direction. We hope that VISTA as development and integration *framework* and as TCAD system will help to improve the reuse of expensive engineering work and facilitate and speed up the creation of TCAD methodology.

REFERENCES

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

[2] J. Mar, "Technology CAD at Intel," in *Technology CAD Systems*, F. Fasching, S. Halama, and S. Selberherr, Eds. New York: Springer-Verlag, pp. 63–74.

[3] F. Fasching, S. Halama, and S. Selberherr, Eds., *Technology CAD Systems*. New York: Springer-Verlag, 1993, pp. 63–74.

[4] S. Halama, F. Fasching, C. Fischer, H. Kosina, E. Leitner, P. Lindorfer, Ch. Pichler, H. Pimingstorfer, H. Puchner, G. Rieger, G. Schrom, T. Simlinger, M. Stiftinger, H. Stippel, E. Strasser, W. Tuppa, K.Wimmer, and S. Selberherr, "The Viennese Integrated System for Technology CAD Applications," *Microelectron. J.*, vol. 26, no. 2/3, pp. 137–158, Mar. 1995.

[5] S. Kleinfeldt, M. Guiney, J. K. Miller, and M. Barnes. "Design methodology management," *Proc. IEEE*, vol. 82, pp. 231–250, Feb. 1994.

[6] F. P. Brooks. *The Mythical Man-Month: Essays on Software Engineering*. Reading, MA: Addison-Wesley, 1982.

[7] M. R. Simpson, "PRIDE: An integrated design environment for semiconductor device simulation," *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 1163–1174, Sept. 1991.

[8] E. W. Scheckler, A. S. Wong, R. H. Wang, G. Chin, J. R. Camagna, A. R. Neureuther, and R. W. Dutton, "A utility-based integrated system for process simulation," *IEEE Trans. Computer-Aided Design*, vol. 11, no. 7, pp. 911–920, 1992.

[9] A. S. Wong, "Technology computer-aided design frameworks and the PROSE implementation," Ph.D. dissertation, EECS Dep., Univ. of California, Berkeley, 1992.

[10] J. K. Ousterhout, "Tcl: An embeddable command language," in *1990 Winter USENIX Conf. Proc.*, 1990, pp. 133–146.

[11] ——, "An X11 toolkit based on the Tcl language," in *1991 Winter USENIX Conf. Proc.*, 1991, pp. 105–115.

[12] R. W. Scheifler, J. Gettys, and R. Newman, *X Window System: C Library and Protocol Reference*. Bedford, MA: Digital, 1988.

[13] P. J. Asente and R. R. Swick, *X Window System Toolkit, The Complete Programmer's Guide and Specification*. Bedford, MA: Digital, 1990.

[14] Autodesk AG, *AUTOCAD Release 11 Reference Manual*, Publication AC11RM.E1, 1990.

[15] *OSF/Motif Programmer's Guide, Release 1.1*. Englewood Cliffs, NJ: Prentice-Hall, 1991.

[16] P. Lloyd, C. C. McAndrew, M. J. McLennan, S. Nassif, K. Singhal, Ku. Singhal, P. M. Zeitzoff, M. N. Darwish, K. Haruta, J. L. Lentz, H. Vuong, M. R. Pinto, C. S. Rafferty, and I. C. Kizilyalli, "Technology CAD at AT&T," in *Technology CAD Systems*, F. Fasching, S. Halama, and S. Selberherr, Eds. New York: Springer-Verlag, 1993, pp. 1–24.

[17] R. W. Knepper, J. B. Johnson, S. Furkay, J. Slinkman, X. Tian, E. M. Buturla, R. Young, G. Fiorenza, R. Logan, Y. S. Huang, R. R. O'Brien, C. S. Murthy, P. C. Murley, J. Peng, H. H. K. Tang, G. R. Srinivasan, M. M. Pelella, D. A. Sunderland, J. Mandelman, D. Lieber, E. Farrell, and M. Kurasic, "Technology CAD at IBM," in *Technology CAD Systems*, F. Fasching, S. Halama, and S. Selberherr, Eds. New York: Springer-Verlag, 1993, pp. 25–62.

[18] W. Jacobs, "The SATURN technology CAD system," in *Technology CAD Systems*, F. Fasching, S. Halama, and S. Selberherr, Eds. New York: Springer-Verlag, 1993, pp. 147–162.

[19] J. Daniell and S. W. Director, "An object oriented approach to CAD tool control," *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 698–713, June 1991.
[20] D. M. H. Walker, J. K. Kibarian, Ch. S. Kellen, and A. J. Strojwas, "A TCAD framework for development and manufacturing," in *Technology CAD Systems*, F. Fasching, S. Halama, and S. Selberherr, Eds. New York: Springer-Verlag, 1993, pp. 83–112.
[21] A. Neureuther, R. Wang, and J. Helmsen, "Perspective on TCAD integration at Berkeley," in *Technology CAD Systems*, F. Fasching, S. Halama, and S. Selberherr, Eds. New York: Springer-Verlag, 1993, pp. 75–82.
[22] D. M. Betz, *XLISP: An Object-Oriented Lisp, Version 2.1*, 1989.
[23] R. Stallman, *GNU Emacs Manual*, Oct. 1986.
[24] N. Mayer, "WINTERP: An object-oriented rapid prototyping, development and delivery environment for building user-customizable applications with the OSF/Motif UI Toolkit," Hewlett-Packard Laboratories, Palo Alto, CA, Tech. Rep., 1991.
[25] K. Funakoshi and K. Mizuno, "A rule-based VLSI process flow validation system with macroscopic process simulation," *IEEE Trans. Semicond. Manufact.*, vol. 3, no. 4, pp. 239–246, Nov. 1990.
[26] A. I. Wasserman, "Tool integration in software engineering environments," in *Software Engineering Environments*, no. 467 in *Lecture Notes in Computer Science*. New York: Springer-Verlag, pp. 137–149.
[27] I. Thomas and B. A. Nejmeh, "Definitions of tool integration for environments," *IEEE Software*, vol. 9, no. 2, pp. 29–35, Mar. 1992.
[28] Ch. Pichler and S. Selberherr, "Process flow representation within the VISTA framework," in *Simulation of Semiconductor Devices and Processes*, vol. 5, S. Selberherr, H. Stippel, and E. Strasser, Eds.. New York: Springer-Verlag, 1993, pp. 25–28.
[29] S. Halama, *The Viennese Integrated System for Technology CAD Applications—Architecture and Critical Software Components*, vol. 64 of *Dissertationen der Technischen Universität Wien*. Vienna, Austria: Österreichischer Kunst- und Kulturverlag, 1994.
[30] F. P. Preparata and M. I. Shamos, *Computational Geometry*. New York: Springer-Verlag, 1985.
[31] H. Samet, *The Design and Analysis of Spatial Data Structures*. Reading, MA: Addison-Wesley, 1990.
[32] D. S. Wen, W. H. Chang, Y. Lii, A. C. Megdanis, P. McFarland, and G. Bronner, "A fully planarized 0.25 μm CMOS technology," in *Proc. Symp. VLSI Technol.*, 1991, pp. 83–84.
[33] B. Davari, C. W. Koburger, R. Schulz, J. D. Warnock, T. Furukawa, W. Jost, W. G. Schwittek, J. K. DeBrosse, M. L. Kerbaugh, and J. L. Mauer, "A new planarization technique, using a combination of RIE and chemical mechanical polish (CMP)," in *Int. Electron Devices Meeting*, 1989, pp. 61–64.
[34] G. Hobler, S. Halama, W. Wimmer, S. Selberherr, and H. Pötzl, "RTA-simulations with the 2-D process simulator PROMIS," in *Proc. Workshop Numerical Modeling Processes Devices Integrated Circuits NUPAD III*, Honolulu, HI, 1990, pp. 13–14.
[35] H. Stippel, "Simulation der ionen-implantation," Dr.technicae dissertation, Dep. Elec. Eng., Technische Universität Wien, Vienna, Austria, 1993.
[36] Technology Modeling Associates, Inc., *TMA TSUPREM-4, Two-Dimensional Process Simulation Program Version 6*, Dec. 1993.
[37] E. Strasser and S. Selberherr, "A general simulation method for etching and deposition processes," in *Simulation of Semiconductor Devices and Processes*, vol. 5, S. Selberherr, H. Stippel, and E. Strasser, Eds. New York: Springer-Verlag, 1993, pp. 357–360.
[38] F. Fasching, "The Viennese integrated system for technology CAD applications- Data level design and implementation," Dr.technicae dissertation, Dep. Elec. Eng., Technische Universität Wien, Vienna, Austria, 1994.
[39] R. Stallman and R. McGrath, *GNU make, Version 3.63*, Jan. 1993.
[40] W. Fichtner and D. Aemmer, Eds., *Simulation of Semiconductor Devices and Processes* , vol. 4. Hartung-Gorre: Konstanz, 1991.
[41] S. Selberherr, H. Stippel, and E. Strasser, Eds., *Simulation of Semiconductor Devices and Processes*, vol. 5. New York: Springer-Verlag, 1993.
[42] *Proc. Workshop Numerical Modeling Processes Devices Integrated Circuits NUPAD III*, Honolulu, HI, 1990.
[43] P. Lloyd, E. J. Prendergast, and K. Shinghal, "Technology CAD for competitive products," in G. Baccarani and M. Rudan, Eds., *Simulation of Semiconductor Devices and Processes*, vol. 3. Bologna: Tecnoprint, 1988, pp. 111–126.
[44] P. Lloyd, H. K. Dirks, E. J. Prendergast, and K. Singhal, "Technology CAD for competitive products," *IEEE Trans. Computer-Aided Design*, vol. 9, no. 11, pp. 1209–1216, 1990.
[45] D. M. H. Walker, C. S. Kellen, and A. J. Strojwas, "The PREDITOR process editor and statistical simulator," in *Proc. VPAD*, 1991, pp. 120–121.
[46] D. M. H. Walker, Ch. S. Kellen, D. M. Svoboda, and A. J. Strojwas, "The CDB/HCDB semiconductor wafer representation server," *IEEE Trans. Computer-Aided Design*, vol. 12, pp. 283–295, Feb. 1993.
[47] H. Matsuo, H. Masuda, S. Yamamoto, and T. Toyabe, "A supervised process and device simulation for statistical VLSI design," *Proc. Workshop Numerical Modeling Processes Devices Integrated Circuits NUPAD III*, Honolulu, HI, 1990, pp. 59–60.
[48] S. G. Duvall, "An interchange format for process and device simulation," *IEEE Trans. Computer-Aided Design*, vol. 7, pp. 741–754, July 1988.
[49] J. Mar, K. Bhargavan, S. G. Duvall, R. Firestone, D. J. Lucey, S. N. Nandgaonkar, S. Wu, K. S. Yu, and F. Zarbakhsh, "EASE—An application-based CAD system for process design," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 1032–1038, June 1987.
[50] D. S. Boning, M. L. Heytens, and A. S. Wong, "The intertool profile interchange format: An object-oriented approach," *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 1150–1156, Sept. 1991.
[51] D. S. Boning, M. B. McIlrath, P. Penfield Jr., and E. M. Sachs, "A general semiconductor process modeling framework," *IEEE Trans. Semicond. Manufact.*, vol. 5, pp. 266–280, Nov. 1992.
[52] N. Tanabe, "Technology CAD at NEC," in *Technology CAD Systems*, F. Fasching, S. Halama, and S. Selberherr, Eds. New York: Springer-Verlag, 1993, pp. 237–254.
[53] K. Nishi and J. Ueda, "Technology CAD at OKI," in *Technology CAD Systems*, F. Fasching, S. Halama, and S. Selberherr, Eds. New York: Springer-Verlag, 1993, pp. 255–274.
[54] P. A. Gough, M. K. Johnson, P. Walker, and H. Hermans, "An integrated device design environment for semiconductors," *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 808–821, June 1991.
[55] P. A. Gough, "An integrated design environment for semiconductors," in *Technology CAD Systems*, F. Fasching, S. Halama, and S. Selberherr, Eds. New York: Springer-Verlag, 1993, pp. 131–146.
[56] M. R. Simpson, "PRIDE: An integrated design environment for semiconductor device simulation," in *Proc. NUPAD III*, Honolulu, HI, 1990, pp. 57–58.
[57] H. Jacobs, W. Hänsch, F. Hofmann, W. Jacobs, M. Paffrath, E. Rank, K. Steger, and U. Weinert," SATURN—A device engineer's tool for optimizing MOSFET performance and lifetime," in *Proc. Workshop Numerical Modeling Processes Devices Integrated Circuits NUPAD III*, Honolulu, HI, 1990, pp. 55–56.
[58] P. J. Hopper and P. A. Blakey, "The MASTER framework," in *Technology CAD Systems*, F. Fasching, S. Halama, and S. Selberherr, Eds. New York: Springer-Verlag, 1993, pp. 275–292.
[59] J. Lorenz, C. Hill, H. Jaouen, C. Lombardi, C. Lyden, K. De Meyer, J. Pelka, A. Poncet, M. Rudan, and S. Solmi, "The STORM Technology CAD System," in *Technology CAD Systems*, F. Fasching, S. Halama, and S. Selberherr, Eds. New York: Springer-Verlag, 1993, pp. 163–196.
[60] R. W. Dutton and R. J. G. Goossens, "Technology CAD at Stanford University: Physics, algorithms, software, and applications," in *Technology CAD Systems*, F. Fasching, S. Halama, and S. Selberherr, Eds. New York: Springer-Verlag, 1993, pp. 113–130.
[61] V. Axelrad, Y. Granik, and R. Jewell, "CAESAR: The virtual IC factory as an integrated TCAD user environment," in *Technology CAD Systems*, F. Fasching, S. Halama, and S. Selberherr, Eds. New York: Springer-Verlag, 1993, pp. 293–307.
[62] V. Axelrad, "CAESAR 1.1 Released," Technology Modeling Associates, Inc., Palo Alto, CA, Tech. Rep. 1, 1994.
[63] K. S. V. Gopalarao, P. K. Mozumder, and D. S. Boning, "An integrated technology CAD system for process and device designers," *IEEE Trans. VLSI Syst.*, vol. 1, pp. 482–490, Dec. 1993.
[64] K. Kato, N. Shigyo, T. Wada, S. Onga, M. Konaka, and K. Taniguchi, "A supervised simulation system for process and device design based on a geometrical data interface," *IEEE Trans. Electron Devices*, vol. ED-34, pp. 2049–2058, Oct. 1987.
[65] H. Pimingstorfer, S. Halama, S. Selberherr, K. Wimmer, and P. Verhas, "A technology CAD shell," in *Simulation of Semiconductor Devices and Processes*, vol. 4, W. Fichtner and D. Aemmer, Eds. Konstanz: Hartung-Gorre, 1991, pp. 409–416.
[66] F. Fasching, C. Fischer, S. Halama, H. Pimingstorfer, H. Read, S. Selberherr, H. Stippel, W. Tuppa, P. Verhas, and K. Wimmer, "A new open techology CAD system," in M. Ilegems and M. Dutoit, Eds. *21st European Solid-State Device Res. Conf.*—ESSDERC'91, vol. 15 of *Microelectronic Engineering*. Amsterdam: Elsevier, 1991, pp. 217–220.
[67] S. Halama, F. Fasching, H. Pimingstorfer, W. Tuppa, and S. Selberherr, "Consistent user interface and task level architecture of a TCAD

system," in *Workshop Numerical Modeling Processes and Devices for Integrated Circuits NUPAD IV*, Seattle, WA, 1992, pp. 237–242.

[68] S. Halama, F. Fasching, C. Fischer, H. Kosina, E. Leitner, Ch. Pichler, H. Pimingstorfer, H. Puchner, G. Rieger, G. Schrom, T. Simlinger, M. Stiftinger, H. Stippel, E. Strasser, W. Tuppa, K. Wimmer, and S. Selberherr, "The Viennese integrated system for technology CAD applications," in *Technology CAD Systems*, F. Fasching, S. Halama, and S. Selberherr, Eds. New York: Springer-Verlag, 1993, pp. 197–236.

[69] A. S. Wong, D. S. Boning, M. L. Heytens, and A. R. Neureuther, "The intertool profile interchange format," in *Proc. NUPAD III* Honolulu, HI, 1990, pp. 61–62.

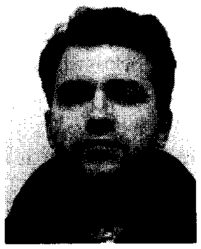
[70] A. S. Wong and A. R. Neureuther, "The intertool profile interchange format: A technology CAD environment approach," *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 1157–1162, Sept. 1991.

[71] R. H. Wang, A. Gabara, and A. R. Neureuther, "BTU—Berkeley Topography Utilities for linking topography and impurity profile simulations," in *Proc. NUPAD IV* 1992, pp. 237–242.

Stefan Halama (S'89–M'93) was born in Vienna, Austria, in 1964. He studied communications engineering at the Technical University of Vienna where he graduated with the degree of "Diplomingenieur" in 1989. He was then with the "Institut für Mikroelektronik" on VISTA as Researcher, where he finished the doctorate degree in electrical engineering with the dissertation "The Viennese Integrated System for Technology CAD Applications—Architecture and Critical Software Components" in 1994.

Since 1987, he has worked as Independent Software Engineer in computer aided design. His current work is focused on grid generation and its application in simulation, tool integration, surveying, and geographic information systems.

Dr. Halama is a member of the Society of Industrial and Applied Mathematics (1992) and the Association for Computing Machinery (1994).

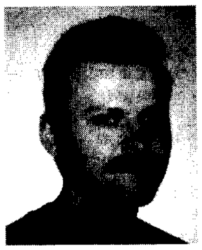


Christoph Pichler (S'93) was born in Vienna, Austria, in 1966. He studied electrical engineering at the Technical University of Vienna, where he received the degree of "Diplomingenieur" in 1991.

During his studies, he held several summer trainee positions at electrical engineering companies in Austria and Switzerland. In November 1991, he joined the Engineering Design Research Center at Carnegie Mellon University in Pittsburgh, PA, where he worked on rapid prototyping and automation in manufacturing. He joined the "Institut für Mikroelektronik" in November 1992, where he is currently working towards the doctoral degree. His scientific interests include engineering design, semiconductor technology, and system integration.

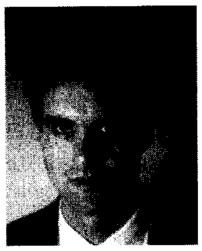
Gerhard Rieger (S'91) was born in Vienna, Austria, in 1963. He studied electrical engineering at the Technical University of Vienna, where he received the degree of "Diplomingenieur" in 1991.

He joined the "Institut für Mikroelektronik" in December 1991, where he is currently working towards the doctoral degree. His current research interests include tool integration, user interfaces, and programming languages.



Gerhard Schrom (S'90–M'91) was born in Mödling, Austria, in 1963. He studied electrical engineering at the Technical University of Vienna, where he received the degree of "Diplomingenieur" in March 1992. During his study he was working on software development projects in the CAD field.

He joined the "Institut für Mikroelektronik" in April 1992, where he is currently working towards the doctoral degree. His research interests include device and circuit simulation, circuit design and synthesis, signal and image processing, and TCAD framework aspects.



Thomas Simlinger was born in Mödling, Austria, in 1963. He studied communication engineering at the Technical University of Vienna, where he received the degree of "Diplomingenieur" in 1992.

He joined the "Institut für Mikroelektronik" in October 1992, where he is currently working towards the doctoral degree. His scientific interests include algorithms and data models, device modeling, and physical aspects in general.

Siegfried Selberherr (M'79–SM'84–F'93) for a photograph and biography, see p. 1114 of the September 1995 issue of this TRANSACTIONS.