Module Interface Specification for a Parallel Mesh Generation Toolbox

Wen Yu

September 2008

Computing and Software

McMaster University

Contents

1 Introduction

One of the advantages of decomposing the system into modules is that each module can be developed independently. However, the secret and services of each module does not provide enough information for parallel coding. A document specifying the interface of each module, called the Module Interface Specification (MIS), is needed. An MIS of a particular module is not only used as a guide by the programmers that are responsible for coding this module, but also by programmers that will use this module. An MIS is abstract because it describes *what* the module will do, but not *how* to do it.

This MIS describes the services of the corresponding modules specified in the document "Module Guide for a Mesh Generator." A state machine MIS is used. Note that some of the modules have multiple projections. In this case, variables listed in section *state variables* give the format of all states for all of the created objects. The idea of multiple projection is also used in Bauer (1995). By using projections, the change of state variables is applicable to the particular object associated with this module. In this system, this particular kind of modules includes the module Vertex, Edge, Cell, and Mesh.

The rest of the document is organized as follows. Section 2 describes the MIS template used in this document. Section 3 copies the module hierarchy from *Module Guide* document for convenience. Section 4, Section 5, Section 6, Section 7, Section 8, Section 11, Section 12 give the MISs for the Vertex Module, Edge Module, Cell Module, Mesh Module, Service Module, Refining Module, and Coarsening Module, respectively.

2 Template

This section gives the template used in this document. This template is modified version of the MIS template presented in Ghezzi et al. (2003) and Hoffman and Strooper (1999). According to this template, each module is modeled as a finite state machine. It has a set of state variables, inputs, outputs, and transitions. In the case that an exception conditions become true, an exception is raised by the associated access program. If an access program has an output, then *Output* is specified. If an access program changes states variables, a *Transition* is specified. The inputs of the access program are listed as arguments.

The discrete mathematics notation used here follows that introduced by Gries and Schneider (1993). This notation is explained in the SRS. A dot notation is used in two cases. One is for referring to a field in a tuple, and the

other is for referencing the access program of a module.

The whole template is composed of four parts. First, the name of the module is given. Second, constants, data types, and access programs that are used by this module, but defined outside of this module, are listed. Third, the syntax of the interface is specified. Finally, the semantics of the interface is described. The template is described in the rest of this section.

2.1 Module Name

If "**(MP)**" is appended to the name of the module, it means that this module has multiple projections.

2.2 Uses

This section lists constants, data types, and access programs that are defined outside of this module. The format of each imported item is specified after each header.

2.2.1 Imported Constants

Uses ⟨ *module name* ⟩ **Imports** ⟨ *resource constants list* ⟩

2.2.2 Imported Data Types

Uses ⟨ *module name* ⟩ **Imports** ⟨ *resource data type list*⟩

2.2.3 Imported Access Programs

Uses ⟨ *module name* ⟩ **Imports** ⟨ *resource access program list*⟩

2.3 Interface Syntax

This section defines the syntax of the module interface. The interface indicates the services that the module provides. Other modules can only access this module through this interface. Other information inside the module is the secret that it hides from other modules. Changing the internal design of a module will not affect the way that other modules use this module. The format of each exported items is specified after each header.

2.3.1 Exported Constants

constant name : type of the constant

2.3.2 Exported Data Types

data type name := structure of the data type

2.3.3 Exported Access Programs

The exported access programs are listed in the tabular format shown below. In this software, exceptions are handled inside the access routine by displaying error messages and terminating the program.

Routine Name	Input	Output	Exceptions

2.4 Interface Semantics

The semantics of the interface is introduced in this section. The components of this section include state variables, state invariants, access program semantics, *etc.*

2.4.1 State Variables

This section lists the state variables in the format of *variable name*: *type*

2.4.2 Assumption

Any assumption about this module are specified here.

2.4.3 Invariant

Predicates that should always hold before and after each access routine in the module.

2.4.4 Access Program Semantics

This section includes possible exceptions, possible outputs, and possible transitions. The contents of this section should be as formal as possible. When necessary and appropriate, an English explanation is included to help readers understand the meaning of some the mathematical notations.

2.4.5 Local Functions

Functions used to facilitate the expression of the interface semantics.

2.4.6 Local Data Types

Data types used to facilitate the expression of the interface semantics.

2.4.7 Local Constants

Constants used to facilitate the expression of interface semantics.

2.4.8 Considerations

Other issues related to the MIS of this module, but not covered in the other parts of the document.

3 Module Decomposition

PMGT is decomposited into the modules listed in Table 1. Note that only the leaf modules are implemented. The *Virtual Memory Module*, *File Read/Write Module*, *Keyboard Input Module*, and *Screen Display Module* are implemented by the operating system and programming language libraries. More information on the modular decomposition of the PMGT can be found in the MG document.

4 MIS of Vertex Module

4.1 Module Name: Vertex (MP)

4.2 Uses

4.2.1 Imported Constants

None

4.2.2 Imported Data Types

None

Level 1	**Level 2**	**Level 3**	**Level 4**
Hardware-Hiding Module	Extended Computer Module	Virtual Memory Module	
		File Read/Write Module	
	Device Interface Module	Keyboard Input Module	
		Screen Display Module	
Behavior-Hiding Module	Input Format Module		
	Output Format Module		
	Service Module		
Software Decision Module	Mesh Data Module	Entity Module	Vertex Module
			Edge Module
			Cell Module
		Mesh Module	
	Algorithm Module	Refining Module	
		Coarsening Module	

Table 1: Module Hierarchy

4.2.3 Imported Access Programs

None

4.3 Interface Syntax

4.3.1 Exported constants

None

4.3.2 Exported Data Types

`VertexT` := tuple of $(x : \mathbb{R},\ y : \mathbb{R})$

4.3.3 Exported Access Programs

The exported access programs for the vertex module are listed in Table 2.

Routine Name	Input	Output	Exceptions
initVertex	$\mathbb{R}$, $\mathbb{R}$		
getVertex		`VertexT`	

Table 2: Exported Access Programs of the Vertex Module

4.4 Interface Semantics

4.4.1 State Variables

$x : \mathbb{R}$
$y : \mathbb{R}$

4.4.2 Invariant

None

4.4.3 Assumptions

initVertex() is called before any other access routine.

4.4.4 Access Program Semantics

4.4.4.1 initVertex($x1 : \mathbb{R},\ y1 : \mathbb{R}$)

- **Transition**
  $x := x1$
  $y := y1$

4.4.4.2 getVertex()

- **Output**
  (x, y)

4.4.5 Local Functions

None

4.4.6 Local Data Types

None

4.4.7 Local Constants

None

4.4.8 Considerations

None

5 MIS of Edge Module

5.1 Module Name: Edge (MP)

5.2 Uses

5.2.1 Imported Constants

None

5.2.2 Imported Data Types

Uses *Vertex Module* **Imports** `VertexT`

5.2.3 Imported Access Programs

None

5.3 Interface Syntax

5.3.1 Exported constants

None

5.3.2 Exported Data Types

`EdgeT` := set of `VertexT`

5.3.3 Exported Access Programs

The exported access programs for the Edge module are listed in Table 3.

Routine Name	Input	Output	Exceptions
initEdge	`VertexT, VertexT`		EqualVertices
getEdge		`EdgeT`	

Table 3: Exported Access Programs of the Edge Module

5.4 Interface Semantics

5.4.1 State Variables

e: set of `VertexT`

5.4.2 Invariant

$\#e = 2$

5.4.3 Assumptions

initEdge() is called before any other access routine.

5.4.4 Access Program Semantics

None

5.4.4.1 initEdge(*start*: `VertexT`, *end*: `VertexT`)

- **Exception**
  $start = end \implies$ EqualVertices
- **Transition**
  $e: = \{start, end\}$

5.4.4.2 getEdge()

- **Output**
  e

5.4.5 Local Functions

None

5.4.6 Local Data Types

None

5.4.7 Local Constants

None

5.4.8 Considerations

None

6 MIS of Cell Module

6.1 Module Name: Cell (MP)

6.2 Uses

6.2.1 Imported Constants

None

6.2.2 Imported Data Types

Uses *Vertex Module* **Imports** `VertexT`

6.2.3 Imported Access Programs

None

6.3 Interface Syntax

6.3.1 Exported constants

None

6.3.2 Exported Data Types

`CellT` := set of `VertexT`

6.3.3 Exported Access Programs

The exported access programs for the cell module are listed in Table 4.

Routine Name	Input	Output	Exceptions
initCell	`VertexT, VertexT, VertexT`		EqualVertices
getCell		`CellT`	

Table 4: Exported Access Programs of the Cell Module

6.4 Interface Semantics

6.4.1 State Variables

c: set of `VertexT`

6.4.2 Invariant

$\#c = 3$

6.4.3 Assumptions

initCell() is called before any other access routine.

6.4.4 Access Program Semantics

None

6.4.4.1 initCell($v1$: `VertexT`, $v2$: `VertexT`, $v3$: `VertexT`)

- **Exception**
  $v1 = v2 \vee v2 = v3 \vee v3 = v1 \implies$ EqualVertices
- **Transition**
  $c := \{v1, v2, v3\}$

6.4.4.2 getCell()

- **Output**
  c

6.4.5 Local Functions

None

6.4.6 Local Data Types

None

6.4.7 Local Constants

None

6.4.8 Considerations

None

7 MIS of Mesh Module

7.1 Module Name: Mesh (MP)

7.2 Uses

7.2.1 Imported Constants

None

7.2.2 Imported Data Types

Uses *Vertex Module* **Imports** `VertexT`
Uses *Edge Module* **Imports** `EdgeT`
Uses *Cell Module* **Imports** `CellT`

7.2.3 Imported Access Programs

None

7.3 Interface Syntax

7.3.1 Exported constants

None

7.3.2 Exported Data Types

`MeshT` := set of `CellT`

7.3.3 Exported Access Programs

The exported access programs for the mesh module are listed in Table 5.

Routine Name	Input	Output	Exceptions
initMesh			
getMesh		`MeshT`	
numOfCells		$\mathbb{N}$	
addCell	`CellT`		CellExist
deleteCell	`CellT`		CellNotExist
onEdge	`VertexT, EdgeT`	$\mathbb{B}$	
belongToCell	`EdgeT , CellT`	$\mathbb{B}$	
inside	`VertexT , CellT`	$\mathbb{B}$	
vertices		set of `VertexT`	
edges		set of `EdgeT`	
boundaryEdges		set of `EdgeT`	
boundaryVertices		set of `VertexT`	

Table 5: Exported Access Programs of the Mesh Module

7.4 Interface Semantics

7.4.1 State Variables

m: set of `CellT`

7.4.2 Invariant

$\#m \geq 0$

7.4.3 Assumptions

initCell() is called before any other access routine.

7.4.4 Access Program Semantics

7.4.4.1 initMesh()

- **Transition**
  $m := \emptyset$

7.4.4.2 getMesh()

- **Output**
  m

7.4.4.3 numOfCells()

- **Output**
  $\#m$

7.4.4.4 addCell(c: `CellT`)

- **Exception**
  $c \in m \Longrightarrow$ CellExist
- **Transition**
  $m := m \cup \{c\}$

7.4.4.5 deleteCell(c: `CellT`)

- **Exception**
  $c \notin m \Longrightarrow$ CellNotExist
- **Transition**
  $m := m \backslash \{c\}$

7.4.4.6 onEdge(v: `VertexT`, e: `EdgeT`)

- **Description**
  Returns true if a vertex v is on the line segment between two vertices (exclusive) of the edge e.
- **Output**
  $\exists v1, v2$: `VertexT` |
  $v1 \in e \wedge v2 \in e \wedge v1 \neq v2 \wedge v \neq v1 \wedge v \neq v2$:
  $(v1.x < v.x \leq v2.x \wedge$
  $(v.y - v1.y)/(v.x - v1.x) = (v2.y - v1.y)/(v2.x - v1.x))$

7.4.4.7 belongToCell(e: `EdgeT`, c: `CellT`)

- **Description**
  Returns true if an edge e belongs to a cell c.
- **Output**
  $\forall v$: `VertexT` | $v \in e : v \in c$

7.4.4.8 inside(v: `VertexT`, c: `CellT`)

- **Description**
  Returns true if a vertex v is inside a cell c. (The algorithm is adopt from Franklin (Last Access: January, 2006).)

- **Output**
  $\exists v1, v2, v3$: `VertexT` |
  $v1 \in c \wedge v2 \in c \wedge v3 \in c \wedge v1 \neq v2 \wedge v2 \neq v3 \wedge v3 \neq v1$:
  $((v.y - v1.y) * (v2.x - v1.x) - (v.x - v1.x) * (v2.y - v1.y)) *$
  $((v.y - v2.y) * (v3.x - v2.x) - (v.x - v2.x) * (v3.y - v2.y)) > 0 \wedge$
  $((v.y - v2.y) * (v3.x - v2.x) - (v.x - v2.x) * (v3.y - v2.y)) *$
  $((v.y - v3.y) * (v1.x - v3.x) - (v.x - v3.x) * (v1.y - v3.y)) > 0$

7.4.4.9 vertices()

- **Description**
  Returns the set of all the vertices of the mesh.

- **Output**
  $\{v$: `VertexT` $\mid (\forall c$: `CellT` $\mid c \in m : v \in c) : v\}$

7.4.4.10 edges()

- **Description**
  Returns the set of all the edges of the mesh

- **Output**
  $\{v1, v2$: `VertexT` $\mid (\forall c$: `CellT` $\mid c \in m : v1 \in c \wedge v2 \in c \wedge v1 \neq v2) :$
  $\{v1, v2\}\}$

7.4.4.11 boundaryEdges()

- **Description**
  Returns a set of boundary edges of the mesh

- **Output**
  $\{b$: `EdgeT` $\mid b \in$ Edges() $\wedge$
  (#$\{c$: `CellT` $\mid c \in m \wedge$ belongToCell*(b, c): c*$\}$=*1):b*$\}$

7.4.4.12 boundaryVertices()

- **Description**
  Returns a set of boundary vertices of the mesh.
- **Output**
  $\{v$: `VertexT` $\mid v \in$ boundaryEdges*(): v*$\}$

7.4.5 Local Functions

7.4.6 Local Data Types

None

7.4.7 Local Constants

None

7.4.8 Considerations

None

8 MIS of Service Module

8.1 Module Name: Service

8.2 Uses

8.2.1 Imported Constants

None

8.2.2 Imported Data Types

Uses *Vertex Module* **Imports** `VertexT`
Uses *Edge Module* **Imports** `EdgeT`
Uses *Cell Module* **Imports** `CellT`
Uses *Mesh Module* **Imports** `MeshT`

8.2.3 Imported Access Programs

Uses *Mesh Module* **Imports** *onEdge(), inside(), vertices(), edges(), boundaryEdges(), boundaryVertices()*

8.3 Interface Syntax

8.3.1 Exported constants

None

8.3.2 Exported Data Types

`InstructionT` := {`REFINE, COARSEN, NOCHANGE`}
`CellInstructionT`:= tuple of (*cell*: `CellT`, *instr*: `InstructionT`)
`RCinstructionT` := tuple of
(*rORc*: `InstructionT`, *cInstru*: set of `CellInstructionT`)

8.3.3 Exported Access Programs

The exported access programs for the services module are listed in Table 6.

Routine Name	Input	Output	Exceptions
isValidMesh	`MeshT`	$\mathbb{B}$	
coveringUp	`MeshT` $\times$ `MeshT`	$\mathbb{B}$	

Table 6: Exported Access Programs of the Services Module

8.4 Interface Semantics

8.4.1 State Variables

None

8.4.2 Invariant

None

8.4.3 Assumptions

None

8.4.4 Access Program Semantics

8.4.4.1 isValidMesh(m: `MeshT`)

- **Description**
  Returns true if cells of the mesh are bounded, conformal, and non overlapping.

- **Output**
  Bounded(m) $\wedge$ Conformal(m) $\wedge$ NoInteriorIntersect(m)

8.4.4.2 coveringUp($m1$:`MeshT`, $m2$: `MeshT`)

- **Description**
  Returns false if any boundary vertex of one mesh is not on a boundary edge of another mesh. Otherwise, return true.

- **Output**
  $\forall v1, v2$: `VertexT` |
  $v1 \in$ boundaryVertice($m1$) $\wedge$ $v2 \in$ boundaryVertices($m2$) :
  ($\exists\, b1, b2$: `EdgeT` | $b1 \in$ boundaryEdges($m1$) $\wedge$ $b2 \in$ boundaryEdges($m2$):
  (onEdge($v1, b2$) $\vee$ $v1 \in b2$) $\wedge$ (onEdge($v2, b1$) $\vee$ $v2 \in b1$))

8.4.5 Local Functions

- ValidEdge: `EdgeT` $\rightarrow \mathbb{B}$
  ValidEdge(e: `EdgeT`) $\equiv \#e = 2$

- Area: `CellT` $\rightarrow \mathbb{R}$
  Area(c: `CellT`) $\equiv \Sigma v1, v2, v3$: `VertexT` | $v1 \in c \wedge v2 \in c \wedge v3 \in c$
  $\wedge\, v1 \neq v2 \wedge v2 \neq v3 \wedge v3 \neq v1$:
  $\frac{1}{12} * |v1.x * v2.y - v2.x * v1.y +$
  $v2.x * v3.y - v3.x * v2.y +$
  $v1.x * v3.y - v3.x * v1.y|$

- ValidCell: `CellT` $\rightarrow \mathbb{B}$
  ValidCell(c: `CellT`) $\equiv \#c = 3 \wedge$ Area(c) ≥ 0

- Bounded: `MeshT` $\rightarrow \mathbb{B}$
  Bounded(m: `MeshT`) $\equiv \forall v$: `VertexT` | $v \in$ boundaryVertices(m):
  ($\#\{e$: `EdgeT` | $e \in$ boundaryEdge(m) $\wedge v \in e : e\} = 2$)

- Conformal: `MeshT` $\rightarrow \mathbb{B}$
  Conformal(m: `MeshT`) $\equiv \forall c1, c2$: `CellT` | $c1 \in m \wedge c2 \in m \wedge c1 \neq c2$:
  ($\exists e$: `EdgeT` | $e \in$ edges(m) : ($\exists v$: `VertexT` | $v \in$ vertices(m) :
  ($c1 \cap c2 = e \vee c1 \cap c2 = v \vee c1 \cap c2 = \emptyset$) $\wedge$ ($\neg$ onEdge($v,\ e$))))

- NoInteriorIntersect: `MeshT` $\rightarrow \mathbb{B}$
  NoInteriorIntersect(m: `MeshT`) $\equiv \forall c1, c2$: `CellT` |
  $c1 \in m \wedge c2 \in m \wedge c1 \neq c2 : (\forall v$: `VertexT` | inside($v$, $c1$) : $\neg$ inside(v, $c2$))

8.4.6 Local Data Types

None

8.4.7 Local Constants

None

8.4.8 Considerations

None

9 MIS of Input Format Module

9.1 Module Name: Input Format

9.2 Uses

9.2.1 Imported Constants

None

9.2.2 Imported Data Types

Uses *Embedding Application* **Imports** `InputFormatT`

9.2.3 Imported Access Programs

None

9.3 Interface Syntax

9.3.1 Exported constants

None

9.3.2 Exported Data Types

None

9.3.3 Exported Access Programs

The exported access programs for the input format module are listed in Table 7.

Routine Name	Input	Output	Exceptions
convertInput	`InputFormatT`	`MeshT`	

Table 7: Exported Access Programs of the Input Format Module

9.4 Interface Semantics

9.4.1 State Variables

None

9.4.2 Invariant

None

9.4.3 Assumptions

None

9.4.4 Access Program Semantics

9.4.4.1 convertInput(m: `InputFormatT`)

- **Output**
  m' such that
  m' is of type `MeshT` and m and m' are equivalent.

9.4.5 Local Functions

None

9.4.6 Local Data Types

None

9.4.7 Local Constants

None

9.4.8 Considerations

- Semantics of access programs in this module heavily depend on the format of the input mesh. At this stage, this information is missing. Unknown data types `InputFormatT` is used to represent the data structures of input mesh. English is used to describe the semantics.

10 MIS of Output Format Module

10.1 Module Name: Output Format

10.2 Uses

10.2.1 Imported Constants

None

10.2.2 Imported Data Types

Uses *Embedding Application* **Imports** `OutputFormatT`

10.2.3 Imported Access Programs

None

10.3 Interface Syntax

10.3.1 Exported constants

None

10.3.2 Exported Data Types

None

10.3.3 Exported Access Programs

The exported access programs for the output format module are listed in Table 8.

Routine Name	Input	Output	Exceptions
convertOutput	`MeshT`	`OutputFormatT`	

Table 8: Exported Access Programs of the Output Format Module

10.4 Interface Semantics

10.4.1 State Variables

None

10.4.2 Invariant

None

10.4.3 Assumptions

None

10.4.4 Access Program Semantics

10.4.4.1 convertOutput(m: `MeshT`)

- **Output**
  m' such that
  m' is of type `OutputFormatT` and m and m' are equivalent.

10.4.5 Local Functions

None

10.4.6 Local Data Types

None

10.4.7 Local Constants

None

10.4.8 Considerations

- Semantics of access programs in this module heavily depend on the format of the requirements of the output. At this stage, this information is missing. Unknown data type `OutputFormatT` are used to represent the data structures of input and output. English is used to describe the semantics.

11 MIS of Refining Module

11.1 Module Name: Refining

11.2 Uses

11.2.1 Imported Constants

None

11.2.2 Imported Data Types

Uses *Mesh Module* **Imports** `MeshT`
Uses *Service Module* **Imports**
`InstructionT, CellInstructionT, RCinstructionT`

11.2.3 Imported Access Programs

Uses *Service Module* **Imports** *isValidMesh(), coveringUp()*

11.3 Interface Syntax

11.3.1 Exported constants

None

11.3.2 Exported Data Types

None

11.3.3 Exported Access Programs

The exported access programs for the refining module are listed in Table 9.

Routine Name	Input	Output	Exceptions
refining	`MeshT, RCinstructionT`	`MeshT`	

Table 9: Exported Access Programs of the Refining Module

11.4 Interface Semantics

11.4.1 State Variables

None

11.4.2 Invariant

None

11.4.3 Assumptions

isValidMesh(m) and $i.rORc =$ `REFINE`
for input m: `MeshT` and i: `RCinstructionT`

11.4.4 Access Program Semantics

11.4.4.1 refining(m: `MeshT`, i: `RCinstructionT`)

- **Output**
  m'
  such that
  ValidMesh(m) $\wedge$ ValidMesh(m') $\wedge$ CoveringUp(m', m) $\wedge \#m' \geq \#m$

11.4.5 Local Functions

None

11.4.6 Local Data Types

None

11.4.7 Local Constants

None

11.4.8 Considerations

None

12 MIS of Coarsening Module

12.1 Module Name: Coarsening

12.2 Uses

12.2.1 Imported Constants

None

12.2.2 Imported Data Types

Uses *Mesh Module* **Imports** `MeshT`
Uses *Service Module* **Imports**
`InstructionT, CellInstructionT, RCinstructionT`

12.2.3 Imported Access Programs

Uses *Service Module* **Imports** *isValidMesh(), coveringUp()*

12.3 Interface Syntax

12.3.1 Exported constants

None

12.3.2 Exported Data Types

None

12.3.3 Exported Access Programs

The exported access programs for the coarsening module are listed in Table 10.

Routine Name	Input	Output	Exceptions
coarsening	`MeshT, RCinstructionT`	`MeshT`	

Table 10: Exported Access Programs of the Coarsening Module

12.4 Interface Semantics

12.4.1 State Variables

None

12.4.2 Invariant

None

12.4.3 Assumptions

isValidMesh(m) and $i.rORc =$ `COARSEN`
for input m: `MeshT` and i: `RCinstructionT`

12.4.4 Access Program Semantics

12.4.4.1 coarsening(m: `MeshT`)

- **Output**
  m'
  such that
  ValidMesh(m) $\wedge$ ValidMesh(m') $\wedge$ CoveringUp(m', m) $\wedge \#m' \leq \#m$

12.4.5 Local Functions

None

12.4.6 Local Data Types

None

12.4.7 Local Constants

None

12.4.8 Considerations

None

References

Brian Bauer. Documenting complicated programs. Technical Report CRL Report 316, Department of Computing and Software, McMaster University, 1995.

W. Randolph Franklin. Pnpoly - point inclusion in polygon test, Last Access: January, 2006. URL `http://www.ecse.rpi.edu/Homepages/wrf/Research/Short_Notes/pnpoly.html`.

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Funcamentals of software Engineering*. Pearson Education, Inc., Upper Saddle River, New Jersey 07458, 2003.

David Gries and Fred B. Schneider. *A Logical Approach to Discrete Math*. Springer-Verlag New Yourk, Inc., 1993.

Daniel Hoffman and Paul Strooper. *Software Design, Automated Testing and Maintenance*. International Thomson Computer Press, 1999.