Software Engineering and VIS

VIS Group

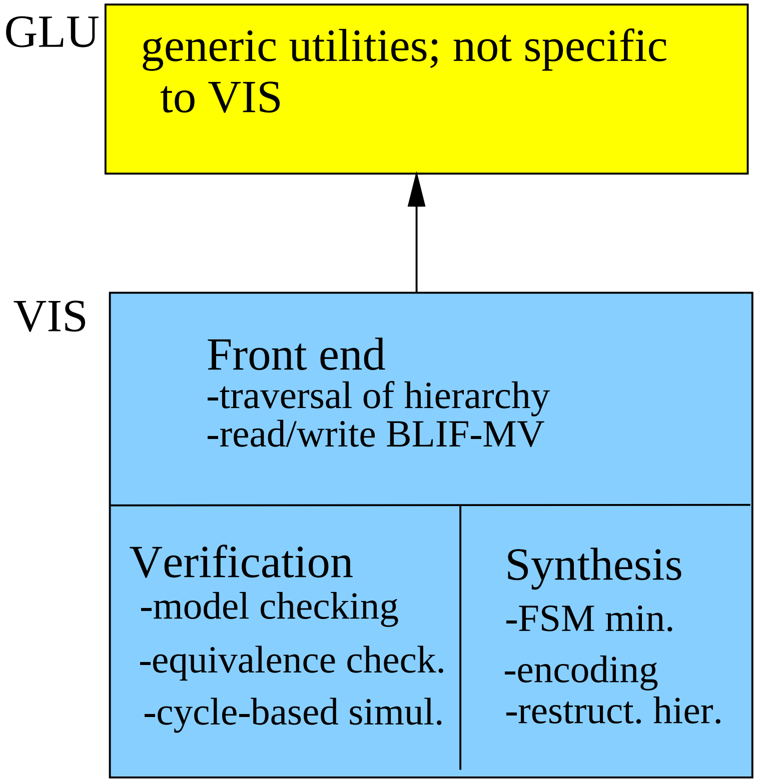
Presented by: Tom Shiple

May 10, 1996

Outline

1. Architecture
   - top-level organization
   - packages
2. Coding conventions
   - package organization
   - naming identifiers
3. Document extraction
   - `extdoc`
   - programmer documentation
   - command documentation

Top-level organization of VIS



What is a package?

- a group of source (.c) and header files (.h) in a subdirectory, centered around either a function or a data structure

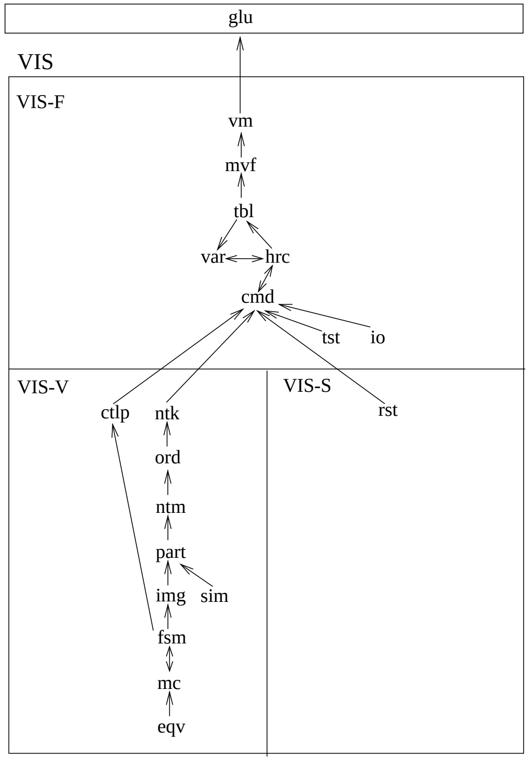
- examples

  1. ntk - network data structure

  2. img - image computation

  3. io - read/write BLIF, BLIF-MV

Packages in VIS



edge from pkg A to B denotes tha A depends on B

GLU Packages

- **`array`** — dynamically allocated arrays
- **`avl`** — priority trees
- **`bdd_cmu`** — David Long's BDD package
- **`bdd_port`** — UCB interface to **`bdd_cmu`**
- **`error`** — error message builder
- **`graph`** — directed graphs
- **`list`** — linked lists
- **`mdd`** — multi-valued decision diagrams
- **`mem`** — memory allocator for **`bdd_cmu`**
- **`sparse`** — sparse matrices
- **`st`** — hash tables
- **`util`** — system-dependent stuff; memory alloc.
- **`var_set`** — bit-arrays

VIS - Front end packages

- `vm` — Contains the `main()` function, and provides the compilation date, version number, and the location of the VIS library.
- `cmd` — The interactive command interface. Provides a global table to store values for user-settable variables (e.g., the value `autoexec`).
- `mvf` — A data structure to represent multi-valued input, multi-valued output functions; based on BDDs.
- `tbl` — A data structure to represent multi-valued relations (e.g., `.table` construct in BLIF-MV).
- `var` — A data structure to represent multi-valued variables, (e.g., `.mv` construct in BLIF-MV).
- `hrc` — Data structures to represent a hierarchical design, in particular the `.model`, `.subckt`, and `.latch` constructs in BLIF-MV.
- `io` — Routines to read and write BLIF-MV and BLIF files.
- `tst` — A package template that can be used as the starting point for the creation of new packages.

VIS - Verification packages

- `ntk` — A directed graph representation, where the vertices are gates, inputs and latches.
- `ord` — Routines to order the MDD variables of a network. Also provides an interface to dynamic ordering of variables.
- `ntm` — A routine to build the MVFs of an arbitrary region of a network.
- `part` — Routines to build an MVF representation of a network. In general, intermediate vertices can be introduced to control the size of the MVFs.
- `sim` — A cycle-based network simulator. Simulation is performed by evaluating the MVFs provided by the `part` package.
- `img` — Generic routines for performing forward and backward image computation. The routines work off the graph of MVFs provided by the `part` package.
- `fsm` — An abstraction of a network. The FSM does not actually store the next state functions of the FSM — these are provided by `part`. It does

store reachability information, fairness constraints-related information, and image computation information.

- `mc` — A fair CTL model checker and debugger for FSMs.

- `eqv` — Routines for performing combinational equivalence between regions of two networks, and for performing sequential equivalence between two FSMs.

- `ctlp` — A CTL parser.

VIS - Synthesis packages

- `rst` — Routines for restructuring the hierarchy.

Outline

1. Architecture
   - top-level organization
   - packages
2. **Coding conventions**
   - package organization
   - naming identifiers
3. Document extraction
   - `extdoc`
   - programmer documentation
   - command documentation

Organization of a package

- header file names are predetermined
- source file names have package prefix, followed by a descriptive term

Package Filenames

Name	Contains
`pkg.h`	Externally-visible functions, etc. Includes system headers only
`pkgInt.h`	Functions, etc., internal to the package. Includes `vm.h`, `pkgPort.h` if present.
`pkgPort.h`	`#defines`, etc., that hide system differences
`pkgUtil.c`	E.g., utility functions. `#include "pkgInt.h"`
`pkgIntUtil.c`	E.g., internal utility functions. `#include "pkgInt.h"`

Organization of a C source file

```
/**CFile****************************************************

  FileName    [required]

  PackageName [required]

  Synopsis    [required]

  Description [optional]

  SeeAlso     [optional]

  Author      [optional]

  Copyright   [Copyright (c) 1994-1996 The Regents
  of the Univ. of California.  All rights reserved.]

***********************************************************/

#include "Int.h"

static char rcsid[] = "$Id:"session1".ps"v"1.1.1.1""""1/"4/"6""1:"":"""vis"Exp"$";
USE(rcsid);


/*--------------------------------------------------------*/
/* Constant declarations                                  */
/*--------------------------------------------------------*/

/*--------------------------------------------------------*/
/* Type declarations                                      */
/*--------------------------------------------------------*/

/*--------------------------------------------------------*/
/* Structure declarations                                 */
/*--------------------------------------------------------*/

/*--------------------------------------------------------*/
/* Variable declarations                                  */
/*--------------------------------------------------------*/

/*--------------------------------------------------------*/
/* Macro declarations                                     */
/*--------------------------------------------------------*/

/**AutomaticStart******************************************/

/*--------------------------------------------------------*/
/* Static function prototypes                             */
/*--------------------------------------------------------*/

/**AutomaticEnd********************************************/


/*--------------------------------------------------------*/
/* Definition of exported functions                       */
/*--------------------------------------------------------*/

/*--------------------------------------------------------*/
/* Definition of internal functions                       */
/*--------------------------------------------------------*/

/*--------------------------------------------------------*/
/* Definition of static functions                         */
/*--------------------------------------------------------*/
```

Scoping of functions

- External: functions visible outside the package. Names are prefixed with "*package_*".
- Internal: functions not visible outside, but visible throughout the package. Names are prefixed with "*package*".
- Local: functions visible only within a single file. Names have no package prefix.

Naming of functions

The words in a function name are arranged in an English sentence-like order:

1. First word: Name of the object being operated on: the subject of the sentence (e.g., `Range`, `Network`).
2. Second word: The action being performed on the object: the verb of the sentence (e.g., `Read`, `Obtain`, `Alloc`).
3. Third and successive words: Modifiers: the object of the sentence (e.g., `States`, `RightChild`, `BddIdArrayFromMddId`)

E.g., `Ntk_NetworkReadName`

Naming of identifiers

Identifier Names

	External	Internal	Local
Functions, Macros	`Pkg_SubjVerbMod`	`PkgSubjVerbMod`	`SubjVerbMod`
Variables	`pkg_numThings`	`pkgNumThings`	`numThings`
Structures	`Pkg_Thing_t`	`PkgThing_t`	`Thing_t`
Enumerated types	`Pkg_ThingList_c`	`PkgThingList_c`	`ThingList_c`
Constants	`PKG_SOME_THING`	`PKGSOME_THING`	`SOME_THING`

Coding style

```
if (alreadyVisited) {
  return;   /* 2 spaces for indentation */
}           /* close brace on separate line */
else {
  depth = NtkNodeComputeDepth(node);
}
```

Outline

1. Architecture
   - top-level organization
   - packages
2. Coding conventions
   - package organization
   - naming identifiers
3. **Document extraction**
   - `extdoc`
   - programmer documentation
   - command documentation

Package Header

```
/**CHeaderFile*****************************************
  FileName    [ required ]
  PackageName [ required ]
  Synopsis    [ required ]
  Description [ optional ]
  SeeAlso     [ optional ]
  Author      [ optional ]
  Copyright   [ required ]
  Revision    [ required ]
******************************************************/
```

FSM package header, unformatted

```
/**CHeaderFile****************************************

  FileName    [fsm.h]

  PackageName [fsm]

  Synopsis [Finite state machine abstraction of a
  network.]

  Description [A finite state machine contains a
  pointer to a network, so it inherits all the
  .....
  default partition associated with the network.<p>

  The FSM maintains information relating to state
  reachability (initial states, reachable states, and
  .....
  convenient format.]

  Author [Shaker Sarwary and Tom Shiple and Rajeev
  Ranjan]

  Copyright [Copyright (c) 1994-1996 The Regents of
  the Univ. of California.  All rights reserved.]

  Revision    [$Id: fsm.h,v 1.31 1995/12/12 22:55:26]

******************************************************/
```


FSM package header, formatted

The fsm package

Finite state machine abstraction of a
network.

By Shaker Sarwary and Tom Shiple and Rajeev Ranjan

- Information for programmers
- Information for developers sorted by function
- Information for developers sorted by file

A finite state machine contains a pointer to a network, so it inherits all the information that is contained in the network, such as node names and MDD ids. An FSM has a partition associated with it; the output and next state functions are derived from the partition. This partition does not necessarily have to be the default partition associated with the network.

The FSM maintains information relating to state reachability (initial states, reachable states, and the partition of reachable states into "BFS shells"), and fairness information (fair states, fairness constraints, and generic debugging information). Also, the FSM stores the MDD ids of present state, next state, and input variables, in a convenient format.

Function header

`Synopsis`	required	A one-line synopsis of the function
`Description`	optional	A longer description of the function: its parameters, its return type, and how it is computed. This should be self-contained.
`SideEffects`	required	A description of any side effects (e.g., modification of global variables). If there are none, include an empty string: "[]".
`SeeAlso`	optional	A space-separated list of the names of related functions, files, packages, etc.
`CommandName`	optional	The name of the command that invokes this function. Omit when the function is not a command.
`CommandSynopsis`	optional	A one-sentence synopsis of the command.
`CommandArguments`	optional	The command-line arguments recognized or required by the command.
`CommandDescript.`	optional	A full description of the command.

Function header, source code

```
/**Function********************************************

  Synopsis [Computes the set of reachable states of an
  FSM.]

  Description [Computes the set of reachable states of
  an FSM.  If this set has already been computed, then
  just returns the result of the previous computation.
  The calling application is responsible for freeing
  the returned MDD in all cases.  If verbosityLevel is
  greater than 1, then information is printed after
  every k steps of reachability, where k is defined by
  the global flag "reach_print_step" (the default for
  k is 1). ]

  SideEffects [The result of the reachable state
  computation is stored with the FSM.]

  SeeAlso [Fsm_FsmComputeInitialStates]

*****************************************************/
mdd_t *
Fsm_FsmComputeReachableStates(Fsm_Fsm_t *fsm,
  int verbosityLevel, int printStep)
{
....
```

Function header, formatted

```
mdd_t *
Fsm_FsmComputeReachableStates(
  Fsm_Fsm_t * fsm,
  int  verbosityLevel,
  int  printStep
)
```

Computes the set of reachable states of an FSM. If this set has already been computed, then just returns the result of the previous computation. The calling application is responsible for freeing the returned MDD in all cases. If verbosityLevel is greater than 1, then information is printed after every k steps of reachability, where k is defined by the global flag "reach_print_step" (the default for k is 1).

Side Effects The result of the reachable state computation is stored with the FSM.

See Also Fsm_FsmComputeInitialStates

Function header, source code

```
/**Function********************************************************

  Synopsis    [The top-level routine for cd.]

  CommandName [cd]

  CommandSynopsis [change the current node]

  CommandArguments [ \[-h\]
  &lt;child_instance_name&gt; or &lt;..&gt;]

  CommandDescription [Changes the current node in the
  existing hierarchy. The argument should be either
  the name of a child node or .. meaning the parent
  node. Note that cd ../foo is not allowed.<p>

  Command options:<p>
  <dl>
  <dt> -h
  <dd> Print the command usage.
  </dl>
  ]

******************************************************************
static int
CommandCd(Hrc_Manager_t **hmgr, int argc, char **argv)
```

Function header, formatted

```
cd - change the current node


cd [-h] <child_instance_name> or <..>

Changes the current node in the existing
hierarchy. The argument should be either the name of a
child node or .. meaning the parent node. Note that cd
../foo is not allowed.

Command options:

-h
     Print the command usage.


Last updated on 960412 14h09
```


Conclusions

- Clean architecture, organized around packages
- Strong coding conventions, to make code uniform
- Automatic documentation extraction, to facilitate development