STRUCTURED FUNCTIONAL MODELING IN SES/WORKBENCH

Bhargav P. Upender
barg@utrc.utc.com

Philip J. Koopman Jr.
koopman@utrc.utc.com

United Technologies Research Center
411 Silver Lane
East Hartford, CT 06108

Overview

Functional Modeling

- Definition & tool support
- Modeling goals

Approach & Techniques

- Structured modeling
- Generic function template
- Data dictionary
- Macro definitions

Summary

Modeling & Tools

Performance Modeling

- Captures statistical behavior of a system
- Abstract high-level model
- Done at later stages of system development

Functional Modeling

- Behavior in terms of functions and their interconnections
- Large detailed model
- Early stages of system development (function to resource mapping unknown)

Modeling tool

- We want a single tool to do both
- SES/Workbench is very good at performance modeling
- Using macros, we adapted workbench to do functional modeling

Functional Modeling

Functional modeling involves defining:

- Functions (data transformation) ..... *Split node*
- Dataflows (input/output of each function) ... *Categories*
- Parameters of the dataflow ..... *Unshared variables*
  (i.e. the information associated with flow)
- Interconnections between functions ..... *Transaction flows between split nodes*

Functional Modeling Goals

We want:

- Executable functional model for:
  - some level of completeness and correctness
- Model at a fine level for:
  - flexible partitioning and allocation to resources
- Scaleable model for:
  - iterative and detailed definition
- Structured model for:
  - ease of understanding
  - multiple developers
  - configuration control
  - modeling error detection

Structured Modeling Approach

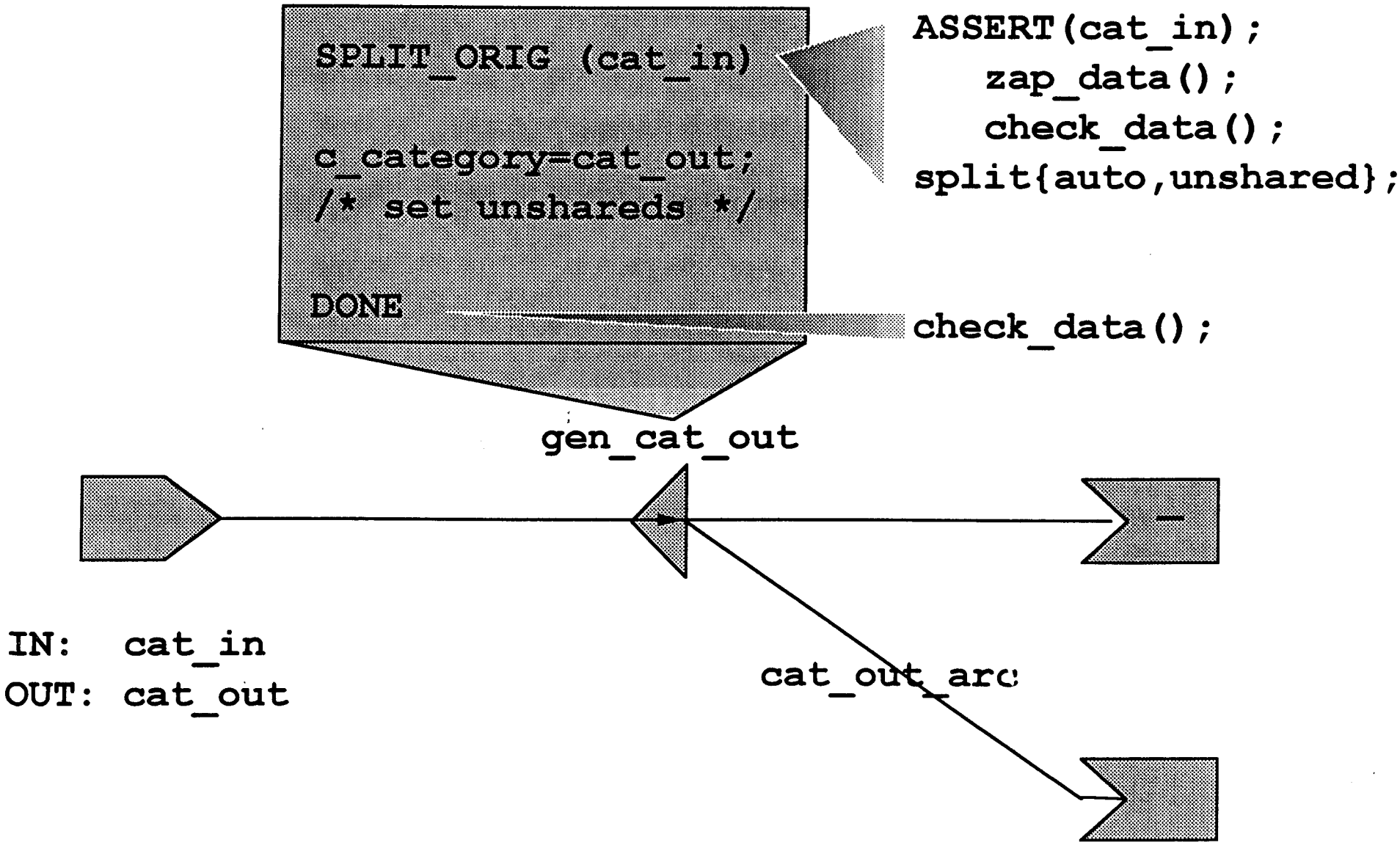
Generic function template: *for function definition*

- Function=data transformation=*split node* : creates a new transaction with a different category
- Define macros for:
  - detecting modeling bugs
  - controlling access to unshared variables

Central router: *for defining interconnections*

- For flexible allocation of functions to resources
- Central instrumentation point for gathering communication workload
- Two approaches:
  - Static - fixed arcs
  - Dynamic - software routing

Generic Function Template



Macros for Reducing Model Errors

`ASSERT(cat_in)` - gives an error if unexpected category arrives at the function

`zap_data()` - sets unshared variables to illegal values if not needed for that category

`check_data()` - gives error if illegal value is detected in a required unshared variable

ASSERT(cat_in)

```
#define ASSERT(expected_cat)                                    \
  if(c_category != expected_cat)                                \
   { printf("wrong category entered \n"); }                     \
   else                                                         \
   { zap_data();                                                \
     check_data();                                              \
   }                                                            \
```


Programming Constructs for the Macros

Token Merging

```
#define FUNCTION (type,name,value) \
     type INIT_/**/name=value;
```

```
FUNCTION (int,count,5)     -->   int INIT_count=5;
```

For ANCI C: `type INIT_ ## name=value;`

Stringization

```
#define MAKESTRING(x) "x"
```

```
fprintf("error at variable: \n",MAKESTRING(count));
```

```
error at variable: count
```

Data Dictionary

"dict.h"

x,y,z are unshared variables

defines valid variables for each category

```
FIELD_DEF (int,x,0)
FIELD_DEF (double,y,0)
FIELD_DEF (long,z,0)

CAT(category1)
 FIELD(x)
 FIELD(y)

CAT(category2)
 FIELD(y)
 FIELD(z)
```


Declaring Variables Using Dictionary

```
unshared struct tr_type{
#define FIELD_DEF(type_,name_,value_)
    type_ name_;
#define CAT(cat_name)
#define FIELD(var_name)
#include "dict.h"
}tr;

#define FIELD_DEF(type_,name_,value_)    \
    type_ INIT_/**/name_=value_;
#define CAT(cat_name)
#define FIELD(var_name)
#include "dict.h"

#define FIELD_DEF(type_,name_,value_)    \
    type_ TEMP_/**/name_;
#define CAT(cat_name)
#define FIELD(var_name)
#include "dict.h"
```

```
unshared struct
tr_type{
int x;
double y;
long z;
}tr;


int INIT_x=0;
double INIT_y=0;
long INIT_z=0;



int TEMP_x;
double TEMP_y;
long TEMP_z;
```

zap_data()

```
void zap_data()
{
#define FIELD_DEF(type_,name_,value_)    \
 type_ temp_/**/name_=ILLEGAL_VALUE_/**/name_;
#define CAT(cat_name)
#define FIELD(name_)
#include "dict.h"    {
#define FIELD_DEF(type_,name_,value_)
#define CAT(cat_name) \
   }  if (c_category==cat_name) {
#define FIELD(name_)\
   temp_/**/name_=tr./**/name_;
#include "dict.h"     }
#define FIELD_DEF(type_,name_,value_) \
   tr./**/name_=temp_/**/name_;
#define CAT(cat_name)
#define FIELD(name_)
#include "dict.h"
}
```

```
void zap_data()
{
int temp_x = ILLEGAL_VALUE_x;
double temp_y = ILLEGAL_VALUE_y;
long temp_z = ILLEGAL_VALUE_z;
  {
} if (c_category==category1) {
 temp_x=tr.x;
 temp_y=tr.y;
} if (c_category==category2) {
 temp_y=tr.y;
 temp_z=tr.z;
   }
tr.x=temp_x;
tr.y=temp_y;
tr.z=temp_z;      }
```


check_data()

```
void check_data()
{ {
#define FIELD_DEF(type_,name_,value_)
#define CAT(cat_name) }  if (c_category == cat_name) {
#define FIELD(var_name)                          \
   if (tr./**/var_name == ILEEGAL_VALUE_/**/var_name) {   \
    incomplete_error(MAKESTRING(var_name));  }
#include "dict.h" } }
```

```
void check_data()
{  {} if (c_category == category1) {
 if (tr.x == ILEEGAL_VALUE_x) { incomplete_error("x"); }
 if (tr.y == ILEEGAL_VALUE_y) { incomplete_error("y"); }
} if (c_category == category2) {
 if (tr.y == ILEEGAL_VALUE_y) { incomplete_error("y"); }
 if (tr.z == ILEEGAL_VALUE_z) { incomplete_error("z"); } } }
```

Summary

Developed data dictionary and macros to:

- Provide controlled access to the unshared variables
- Run-time detection of improper transaction values
  (finds many modeling bugs)
- Ease configuration control

Developed a structure for functional model:

- Function template
  Reduces typing and clicking
  Enables modeler to focus on functional definition
- Central router
  Increases flexibility in partitioning and allocating

References

Harbison, Samuel P. and Steele, Guy L., *C: A Reference Manual,* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1987.

--- *describes token merging and stringization concepts*

Hatley, Derek J., and Pirbhai, Imtiaz A., *Strategies for Real-Time System Specification*, Dorset House Publishing, 353 West 12th Street, New York, 1988.

--- *example of a vending machine DFD*

Scientific and Engineering Software Inc. *SES/Workbench Reference Manual,* Release 2.1, Austin, Texas, February 1992.

--- *why not?*

Bhargav Upender

Fourth Annual SES User Group Meeting

PROCEEDINGS

Radisson Hotel
Austin, Texas

7-8 April 1994