Computer-Aided Recoding for Multi-Core Systems

Rainer Dömer

doemer@uci.edu

With contributions by P. Chandraiah

Center for Embedded Computer Systems
University of California, Irvine

Outline

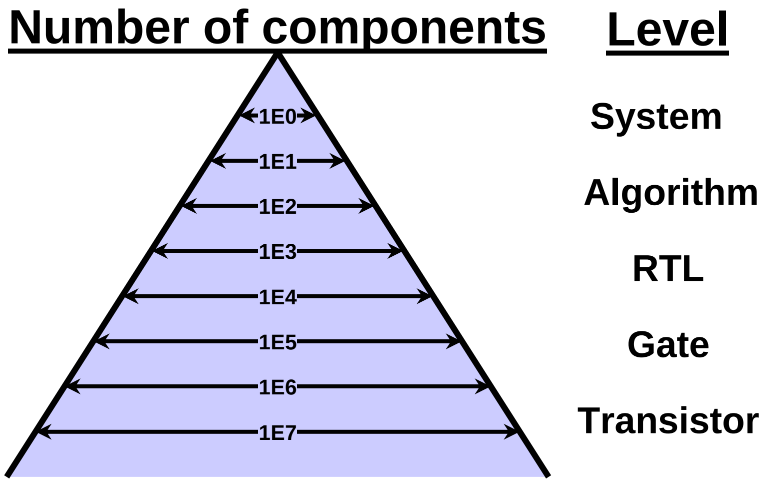
- Embedded System Design
- Computer-Aided Recoding
- Recoding Transformations
  - Creating structural hierarchy
  - Exposing potential parallelism
  - Creating explicit communication
- Interactive Source Recoder
- Experiments and Results
- Conclusions

Embedded System Design

- How can we overcome the productivity gap?

International Technology Roadmap for Semiconductors (ITRS) 2004:
higher-level abstraction and specification is the first promising solution

- System Level Design
  - Unified HW and SW design
  - Higher level of abstraction
    - Fewer, more complex components
    - Maintain system overview
      - Without overwhelming details
    - Compose a system of algorithms
  - System Level Design Languages
    - SpecC [Gajski et. al, 2000]
    - SystemC [Groetker et. al, 2002]



Number of components

1E0
1E1
1E2
1E3
1E4
1E5
1E6
1E7

Level

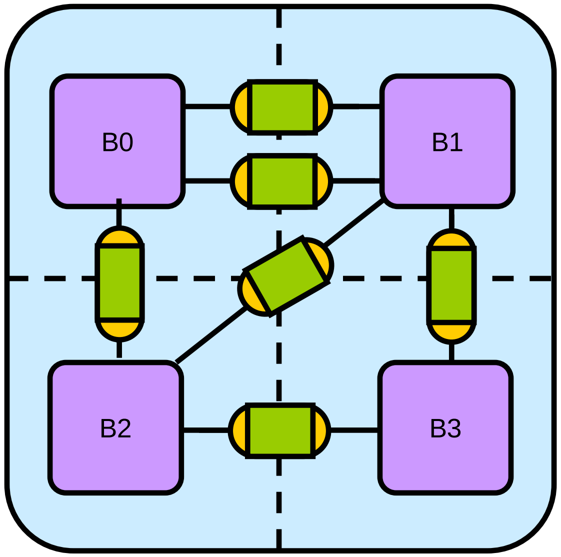
System
Algorithm
RTL
Gate
Transistor

Source: “System Design: A Practical Guide with SpecC”, 2001

Embedded System Design

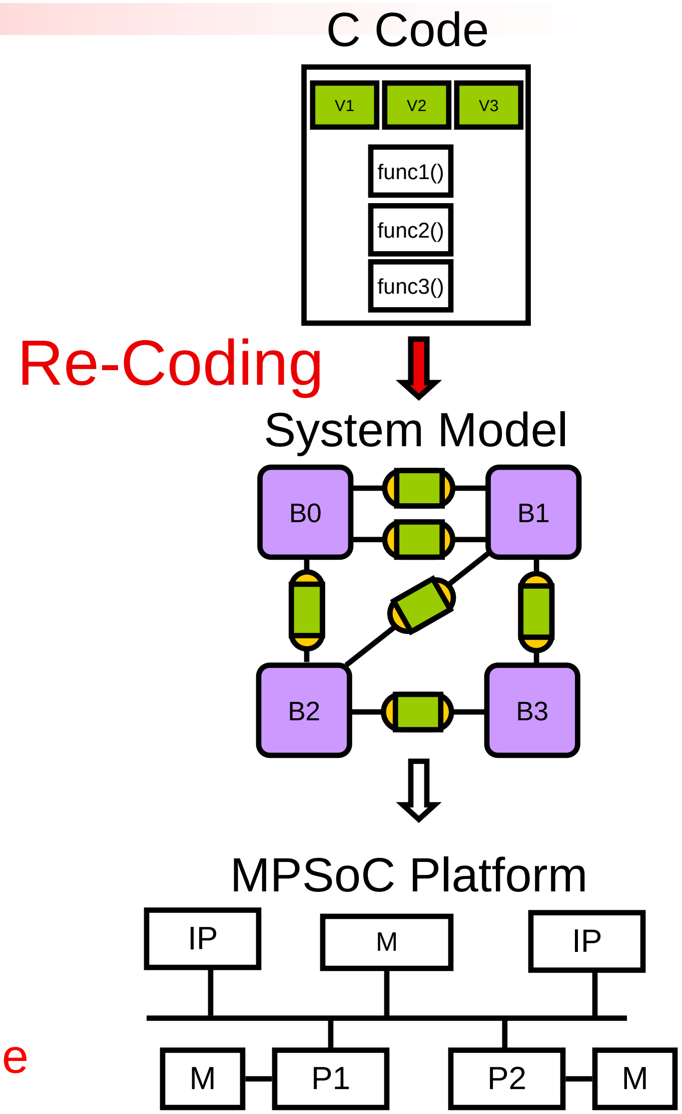
- System Level Modeling
  - Abstract description of a complete system
  - Hardware + Software
- Key Concepts in System Modeling
  - Explicit Structure
    - Block diagram structure
    - Connectivity through ports
  - Explicit Hierarchy
    - System composed of components
  - Explicit Concurrency
    - Potential for parallel execution
    - Potential for pipelined execution
  - Explicit Communication and Computation
    - Channels and Interfaces
    - Behaviors / Modules

System Model



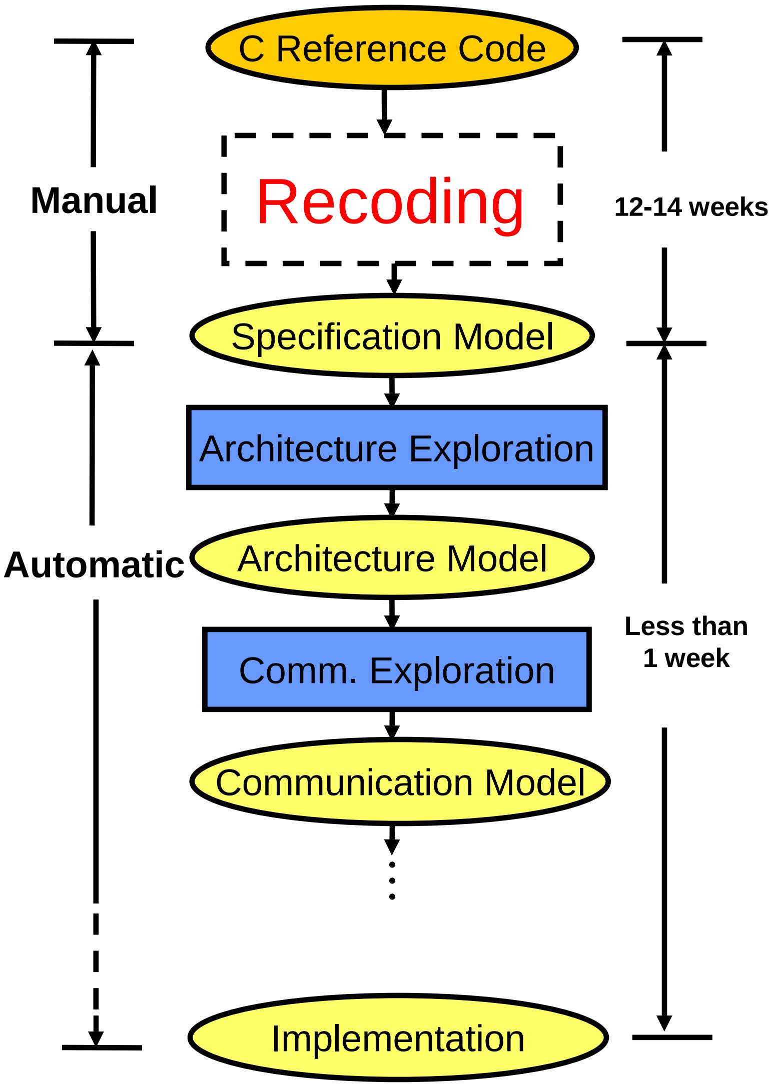
Computer-Aided Recoding

- Embedded System Design Flow
  - Input: System model
  - Output: MPSoC platform
- Actual Starting Point
  - C reference code
  - Flat, unstructured, sequential
  - Insufficient for system exploration
- Need: System Model
  - System-Level Description Language (SLDL)
  - Well-structured
    - Explicit computation, explicit communication
    - Potential parallelism explicitly exposed
  - Analyzable, synthesizable, verifiable
- Research: Automatic *Re-Coding*
  - How to get from flat and sequential C code to a flexible and parallel system model?



Motivation

- Extend of Automation
  - Refinement-based design flow
  - Automatic
    - Specification model down to implementation
    - Example: SCE (mostly automatic)
    - MP3 decoder: less than 1 week
  - Manual
    - C reference code to SpecC specification model
    - Source code transformations
    - MP3 decoder: 12-14 weeks!
- Automation Gap
  - 90% of overall design time is spent on re-coding!
- Proposal: Automatic Recoding



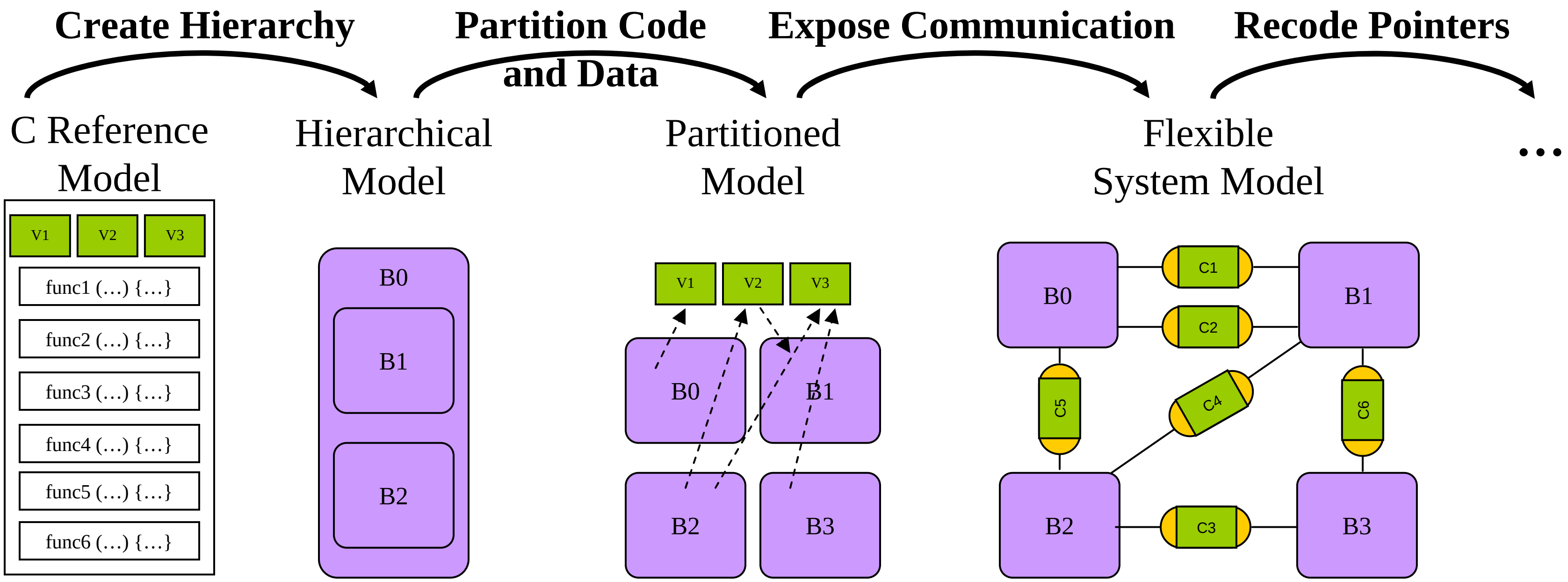
Source: ***System Design: A Practical Guide with SpecC***

Computer-Aided Recoding

- Complete Automation is Infeasible!
  - Today’s parallelizing compilers are largely ineffective
    - Heterogeneous architectures
    - Complexity of embedded applications
    - Hard problems (eliminating pointers, exposing parallelism, etc.)
  - Modeling requires understanding of the application
  - Recoding is not a monolithic transformation
    - Multiple transformations in application-specific order
- ➢ Interactive Approach
  - “Designer-in-the-loop”
  - Designer can utilize application knowledge
- *Designer-controlled* Transformations
  - Designer makes decisions
  - Tool automatically transforms the source code

Overcoming the Specification Gap

- Recoding Transformations
  - Creating structural hierarchy [ASPDAC’08]
  - Code and data partitioning [DAC’07]
  - Creating explicit communication [ASPDAC’07]
  - Recode pointers [ISSS/CODES’07]

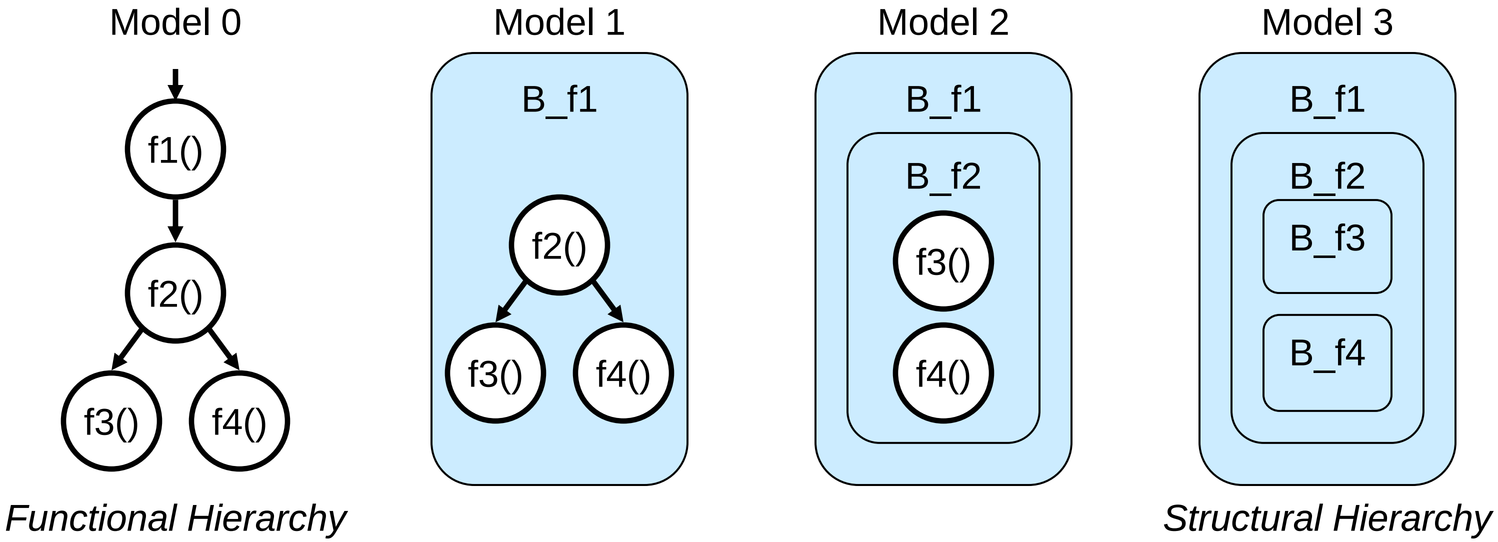


Create Hierarchy **Partition Code and Data** **Expose Communication** **Recode Pointers**

C Reference Model → Hierarchical Model → Partitioned Model → Flexible System Model → …

Creating Structural Hierarchy

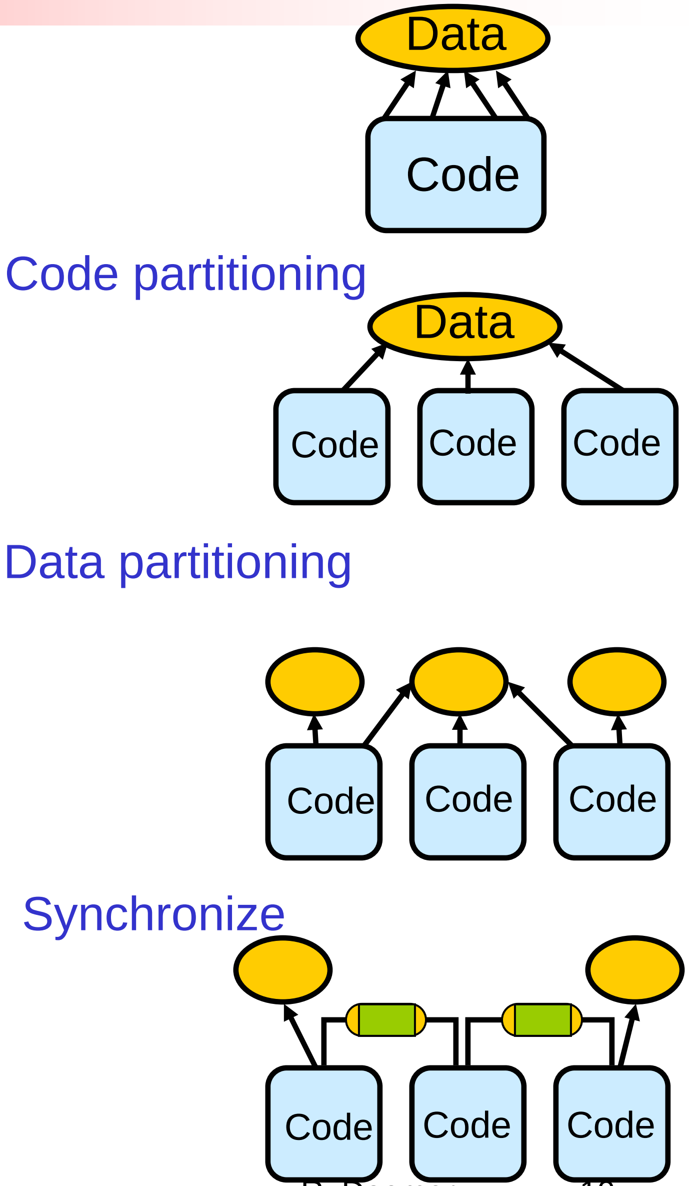
- Recoding
  - – Convert functional hierarchy into structural hierarchy
  - – Step-wise model transformation
  - – Hierarchical encapsulation
    - Utilize given function call tree
    - Convert each function into a behavior
    - Start with root (i.e. **`main()`** function)
    - Continue step by step down to leafs



Exposing Potential Parallelism

- Desirable model features
  - Enable parallel execution
  - Allow mapping to different PEs
- Recoding tasks
  - Partition code
  - Partition data
  - Synchronize dependents
- Recoding transformations
  1. Loop splitting
  2. Cumulative Access Type analysis
  3. Partitioning of vector dependents
  4. Synchronizing dependent variables

  ➢ [DAC'07, TCAD'08]



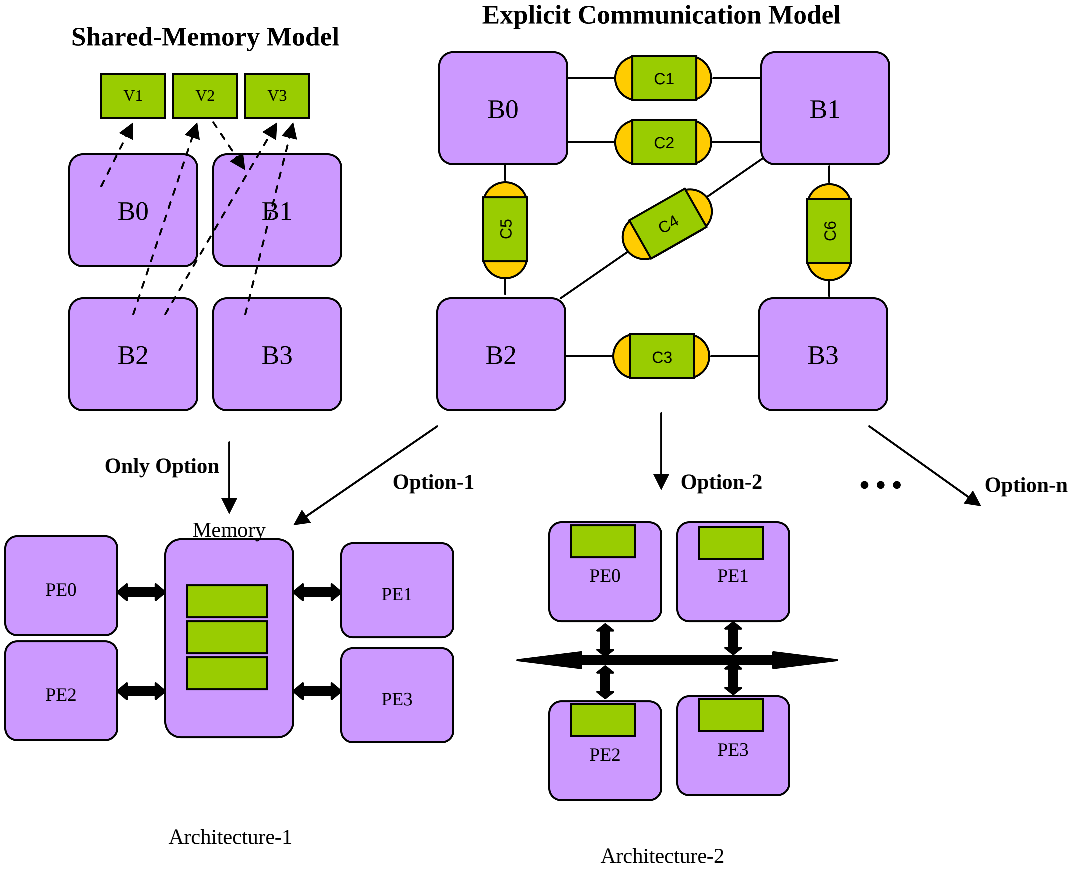
Exposing Communication

- Why create explicit communication?

- Quality of Communication Exploration
  - Number of explorations
  - Extent of automation
  - Time

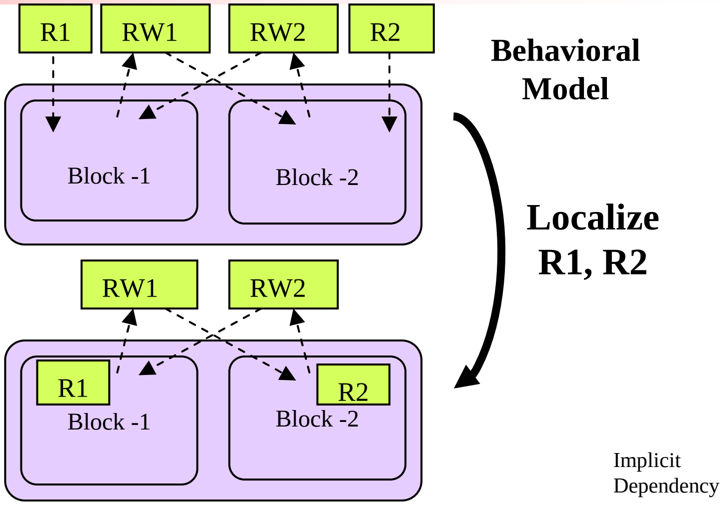
- Shared-Memory Model
  - Global variables limit the number of possible automatic explorations

- Explicit Communication Model
  - Enables automatic exploration of more design alternatives



Exposing Communication: 1. Localize

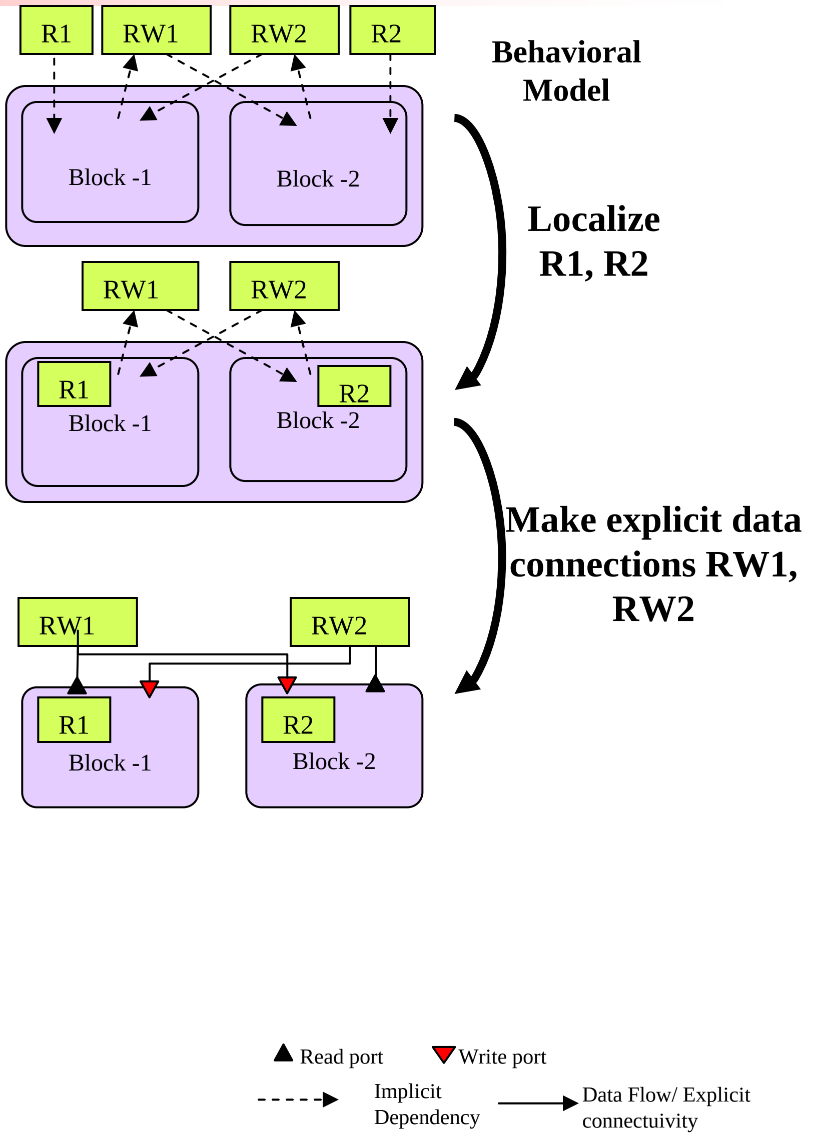
- Localize global variables to partitions
  - To enable multiple explorations
- Procedure
  - Find the global variable
  - Determine the functions and behaviors accessing it
  - If only one behavior is accessing it, migrate the variable into this behavior



Exposing Communication: 2. Expose

- Localize global variables to common parent and provide explicit access
  - Simplifies subsequent analysis of models

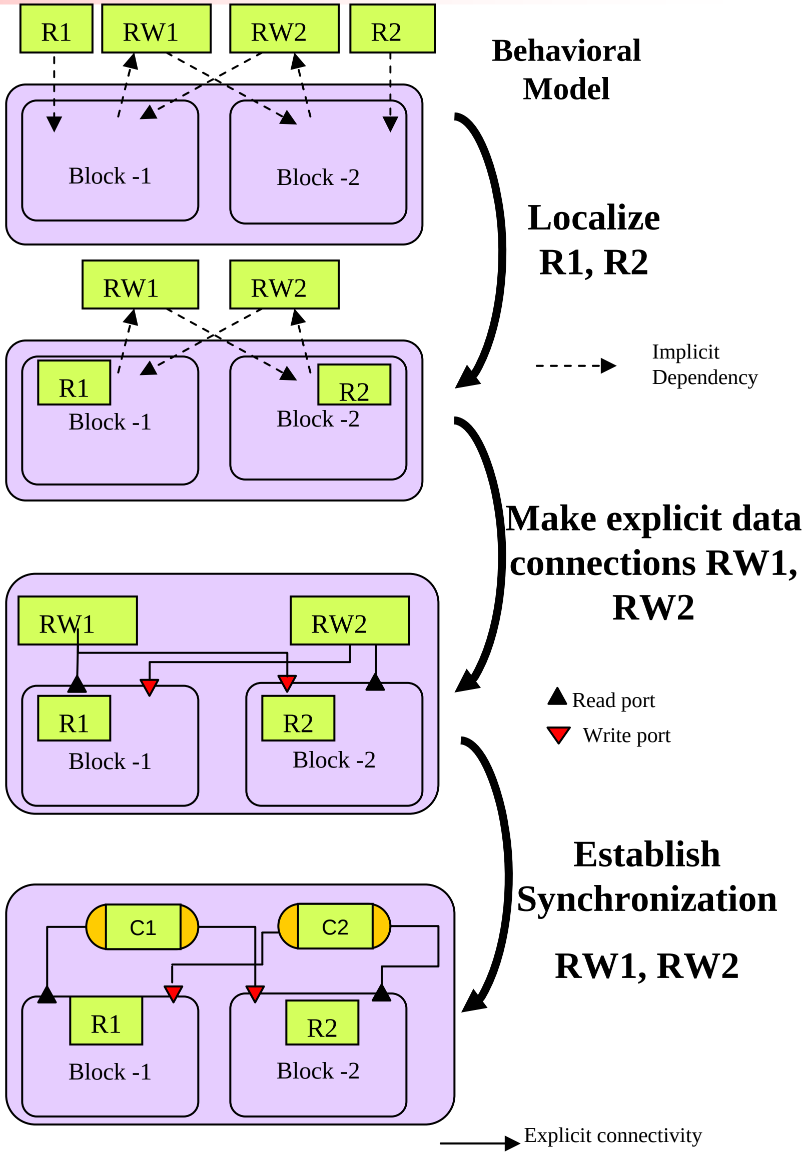
- Procedure
  - Find the global variable
  - Determine the functions and behaviors accessing it
  - If multiple behaviors are accessing it, find the lowest common parent
  - Migrate the variable to the parent
  - Provide access to the variable by recursively inserting ports in behaviors



Exposing Communication: 3. Synchronize

- Use message passing channels instead of variables
  - Defines synchronization scheme
  - Guides exploration tools

- Procedure
  - Create a typed synchronization channel
  - Replace the ports corresponding to the original variable with the channel interface type
  - Modify each access to the variable to call the appropriate interface function of the channel
    - read() / receive()
    - write() / send()



Behavioral Model

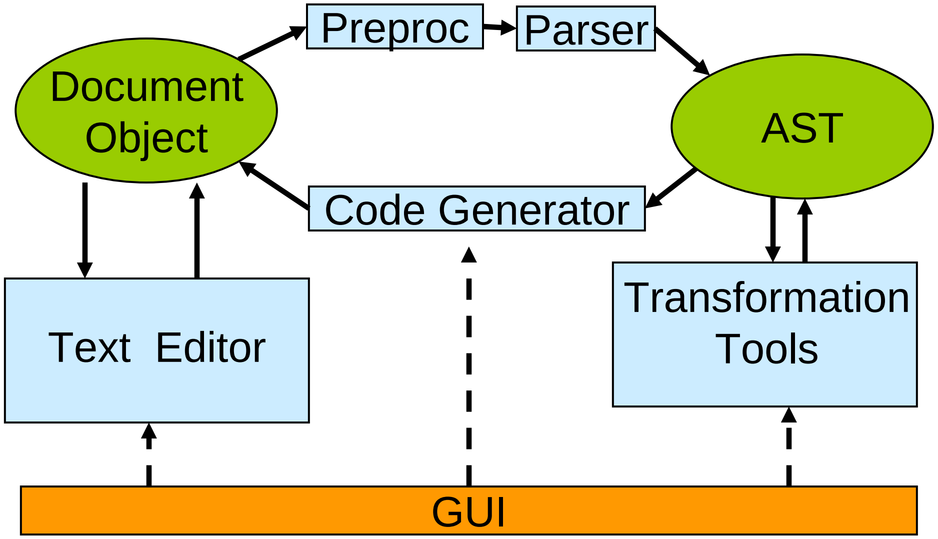
Localize R1, R2

Make explicit data connections RW1, RW2

Establish Synchronization RW1, RW2

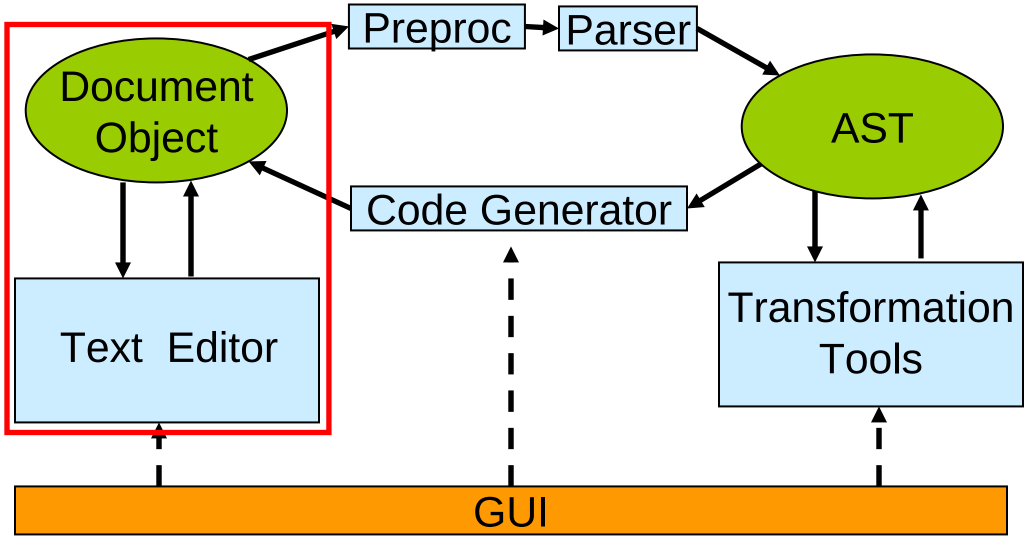
Interactive Source Recoder

- Implementation
  - Integrated Development Environment (IDE)
- *Cute* tool is a union of
  - Text editor
  - Abstract Syntax Tree (AST)
  - Parser
  - Transformations
  - Code generator



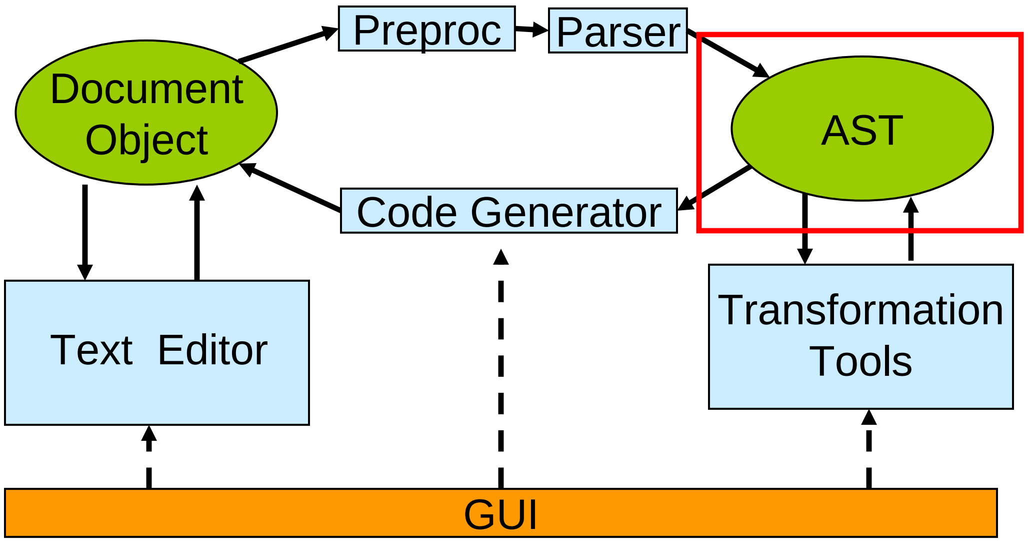
Interactive Source Recoder

- Text editor
  - Interface to the designer
  - Basic and advanced source-code editing
    - C/C++/SpecC
  - Document object
    - Based on Andrew text editor [8]



Interactive Source Recoder

- Text editor
- Abstract Syntax Tree
  - Captures the structure of the design model
  - Used by transformation tools
  - Complete coverage
    - C and SLDLs
    - Correspondence with document object
  - Can re-generate code in its original form



Preproc

Parser

Document Object

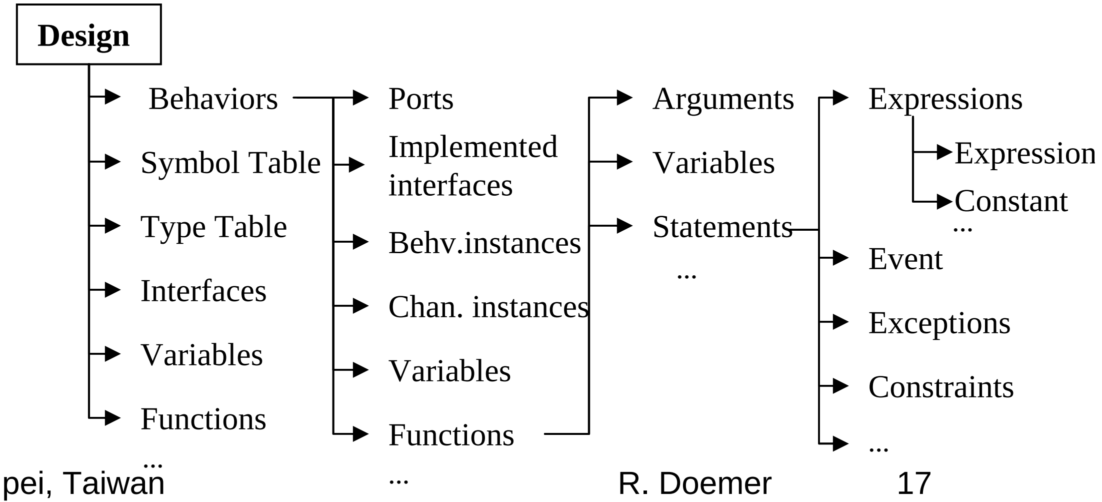
AST

Code Generator

Text Editor

Transformation Tools

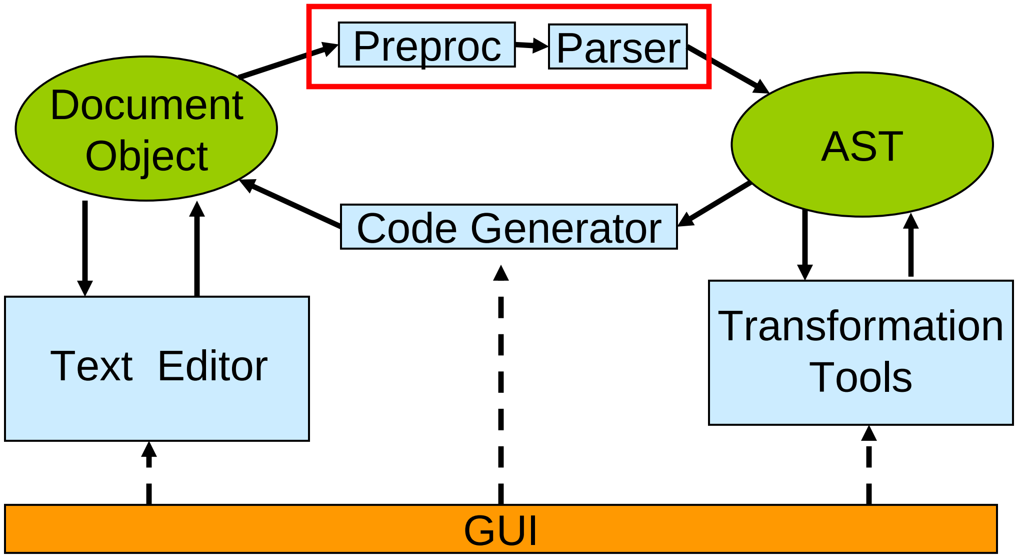
GUI



- **Design**
  - Behaviors
    - Ports
    - Implemented interfaces
    - Behv.instances
    - Chan. instances
    - Variables
    - Functions
      - Arguments
      - Variables
      - Statements
        - Expressions
          - Expression
          - Constant
          - ...
        - Event
        - Exceptions
        - Constraints
        - ...
      - ...
    - ...
  - Symbol Table
  - Type Table
  - Interfaces
  - Variables
  - Functions
  - ...

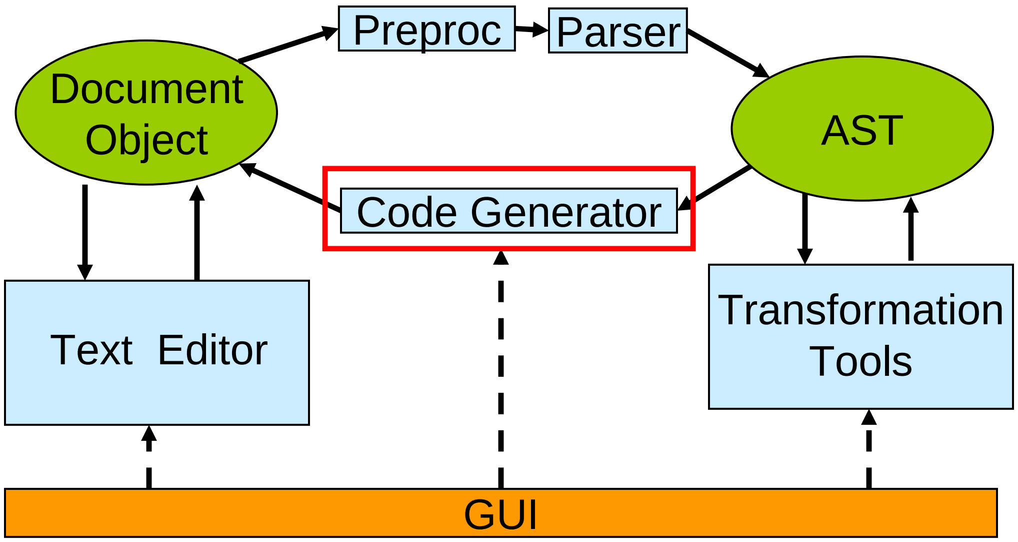
Interactive Source Recoder

- Text editor
- Abstract Syntax Tree
- Preprocessor and Parser
  - Build AST from text
  - Keep AST in synch
  - Complement the editor
    - Color coding
    - Syntax high-lighting



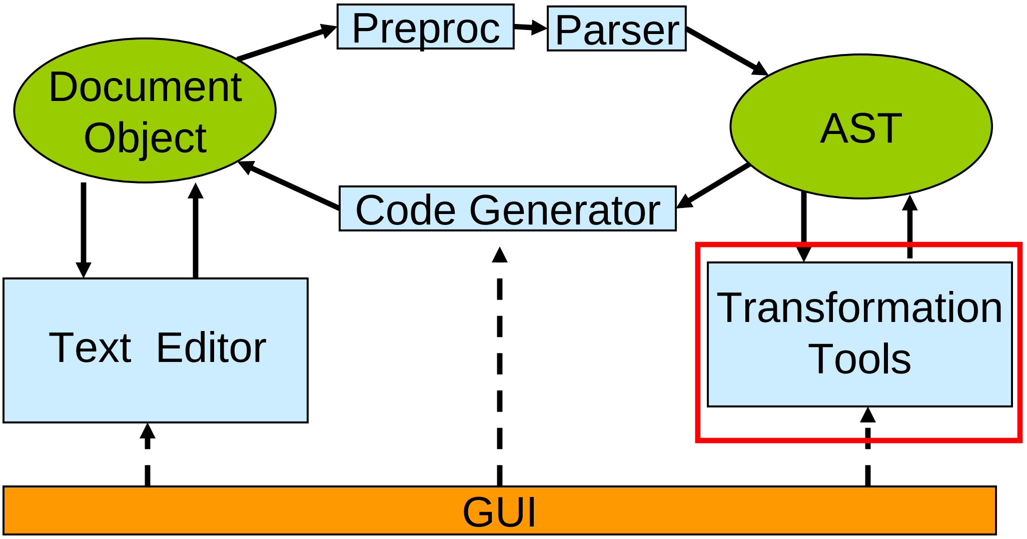
Interactive Source Recoder

- Text editor
- Abstract Syntax Tree
- Preprocessor and Parser
- Code Generator
  - Generates SLDL source code after transformations
  - Keeps text in synch



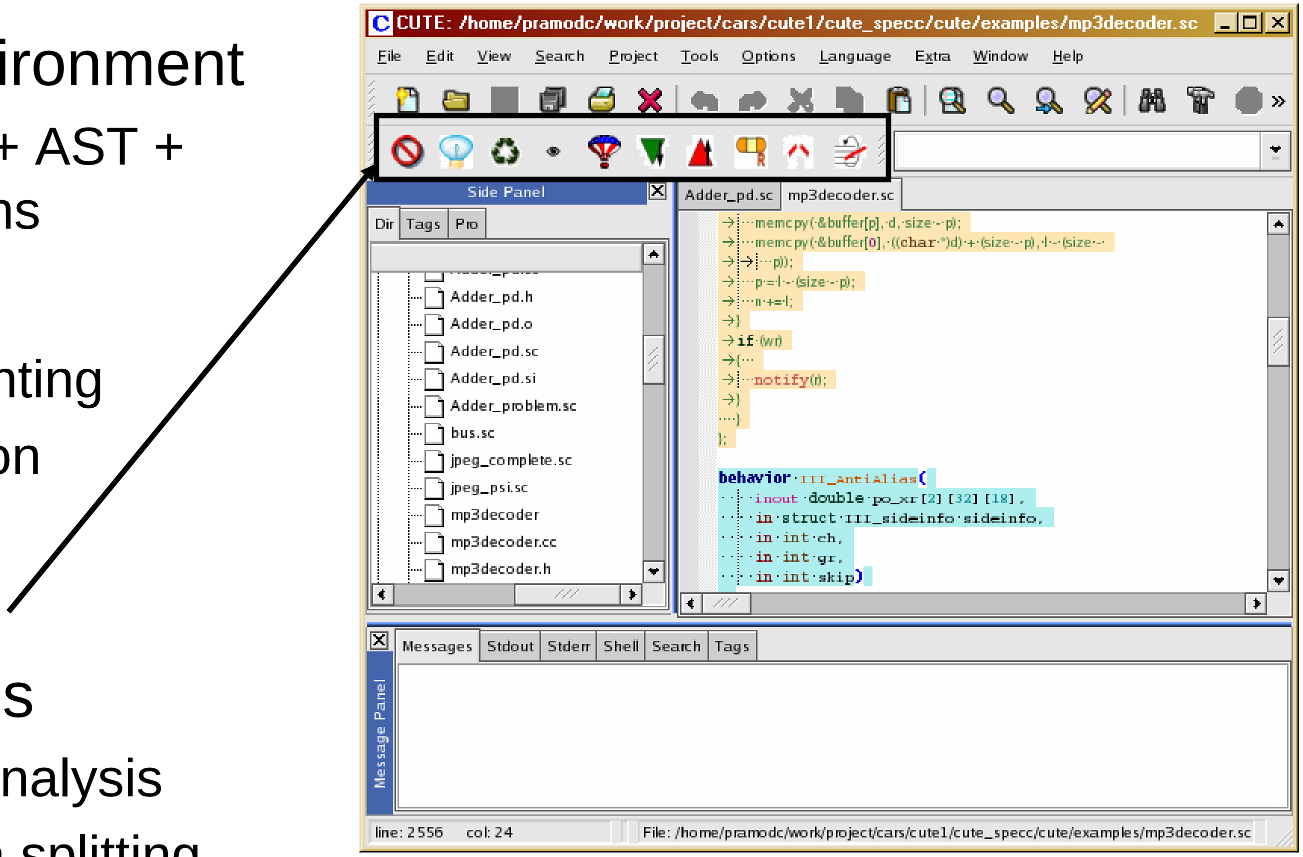
Interactive Source Recoder

- Text editor
- Abstract Syntax Tree
- Preprocessor and Parser
- Code Generator
- Transformation tools
  - Recoding transformations
    - Code partitioning
    - Create structural hierarchy
  - Data transformations
    - Variable re-scoping
    - Data structure partitioning
  - Analysis
    - Dependency analysis
    - Pointer analysis



Interactive Source Recoder

- Interactive Environment
  - Scintilla + QT + AST + Transformations
- Basic editing
  - Syntax highlighting
  - Auto-completion
  - …
- Recoding Transformations
  - Dependency analysis
  - Code and data splitting
  - Variable re-scoping
  - Port insertion
  - …



CUTE: /home/pramodc/work/project/cars/cute1/cute_specc/cute/examples/mp3decoder.sc

```
	   memcpy( &buffer[p], d, size - p);
	   memcpy( &buffer[0], ((char *)d) + (size - p), l - (size -
		   p));
	   p = l - (size - p);
	   n += l;
	}
	if (wr)
	{
	   notify(r);
	}
    }
};

behavior III_AntiAlias(
     inout double po_xr[2][32][18],
     in struct III_sideinfo sideinfo,
     in int ch,
     in int gr,
     in int skip)
```

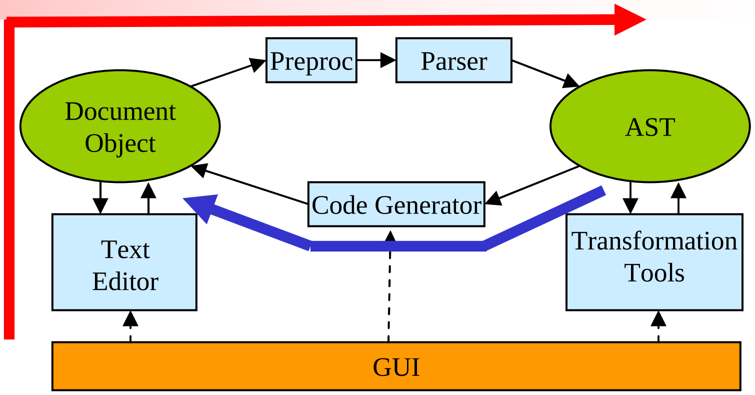
line: 2556 col: 24 File: /home/pramodc/work/project/cars/cute1/cute_specc/cute/examples/mp3decoder.sc

Experiments and Results

- We have conducted various sets of experiments
- Goals
  - Responsiveness of the “compiler in the editor”
  - Estimated Productivity Gains
    - Extrapolation based on the number of lines of code changed
  - Measured Productivity Gains
    - Class of graduate students
- Design examples
  - GSM Vocoder (voice codec in mobile phones)
  - MP3 Decoder (audio decoder, e.g. iPod)
    - Fixed-point version
    - Floating-point version
  - JPEG Encoder (image encoder, e.g. digital camera)
  - …

Experiments and Results: Responsiveness

- Why measure Responsiveness ?
  - To check feasibility
- Responsiveness
  - Response to designer actions
  - Time to synch AST
    - On editing
  - Time to synch Editor
    - On transformation
  - Depends on the size of the AST
- Design examples
  - JPEG, MP3, GSM
  - << 1 sec (on a 3 GHz Linux PC)
  - File I/O overhead (20%)



Operation	Simple	JPEG	MP3	GSM
Lines of code	174	1642	7086	7492
Objects in AST	1073	5338	31763	26009
Synch AST	**0.15 secs**	**0.19 secs**	**0.68 secs**	**0.55 secs**
Synch Editor	**0.16 secs**	**0.20 secs**	**0.73 secs**	**0.59 secs**

Experiments and Results

- Productivity Gain
  - Creating structural hierarchy
    - Manually
      - estimation
    - Automatically
      - measured
- Results
  - Manual time
    - weeks
  - Recoding time
    - minutes

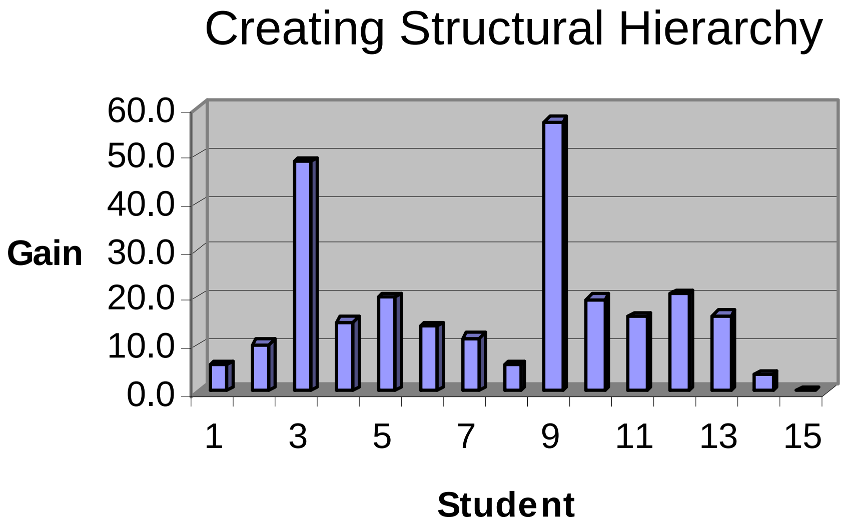
Properties	JPEG	Float-MP3	Fix-MP3	GSM
Lines of C code	1K	3K	10K	10K
C Functions	32	30	67	163
Lines of SpecC code	1.6K	7K	13K	7K
Behaviors created	28	43	54	70
Re-Coding time	≈ 30 mins	≈ 35 mins	≈ 40 mins	≈ 50 mins
Manual time	1.5 weeks	3 weeks	2 weeks	4 weeks
Productivity gain	120	205	120	192

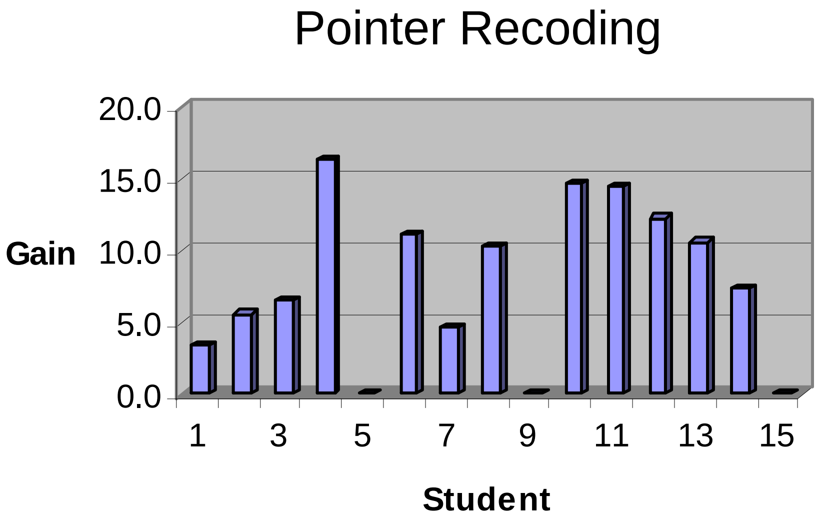
[ASPDAC’08]

- Significant productivity gains!

Experiments and Results: Productivity

- Measured Productivity Gains
  - Class of 15 graduate students
  - Recode an MP3 design example
    - Manually (given detailed instructions)
    - Automatically (using the Source Recoder)
- Results





- Productivity factors vary, but show significant gains!

Conclusions

- Embedded System Design
  - Start from higher level of abstraction
  - Need flexible system models in SLDL
- Motivation
  - Automation gap between C reference and SLDL system models
  - 90% of the overall design time spent on “coding” and “re-coding”
  - Need for design automation
- Problem
  - Complete automation is difficult
- Approach
  - *Computer-Aided Recoding* using Source Recoder
  - Designer-in-the-loop
- Results
  - Significant productivity gains
- Future work
  - Research and develop more transformations
  - Improve interactive graphical environment

References

- [ASPDAC'07] P. Chandraiah, J. Peng, R. Dömer, *"Creating Explicit Communication in SoC Models Using Interactive Re-Coding"*, Proceedings of the Asia and South Pacific Design Automation Conference 2007, Yokohama, Japan, January 2007.
- [IESS'07] P. Chandraiah, R. Dömer, *"An Interactive Model Re-Coder for Efficient SoC Specification"*, Proceedings of the International Embedded Systems Symposium, "Embedded System Design: Topics, Techniques and Trends" (ed. A. Rettberg, M. Zanella, R. Dömer, A. Gerstlauer, F. Rammig), Springer, Irvine, California, May 2007.
- [DAC'07] P. Chandraiah, R. Dömer, *"Designer-Controlled Generation of Parallel and Flexible Heterogeneous MPSoC Specification"*, Proceedings of the Design Automation Conference 2007, San Diego, California, June 2007.
- [ISSS+CODES'07] P. Chandraiah, R. Dömer, *"Pointer Re-coding for Creating Definitive MPSoC Models"*, Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis, Salzburg, Austria, September 2007.
- [ASPDAC'08] P. Chandraiah, R. Dömer, *"Automatic Re-coding of Reference Code into Structured and Analyzable SoC Models"*, Proceedings of the Asia and South Pacific Design Automation Conference 2008, Seoul, Korea, January 2008.
- [TCAD'08] P. Chandraiah, R. Dömer, *"Code and Data Structure Partitioning for Parallel and Flexible MPSoC Specification Using Designer-Controlled Re-Coding"*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 27, no. 6, pp. 1078-1090, June 2008.
- [DATE'09] R. Leupers, A. Vajda, M. Bekooij, S. Ha, R. Dömer, A. Nohl, *"Programming MPSoC Platforms: Road Works Ahead!"*, Proceedings of Design Automation and Test in Europe, Nice, France, April 2009.
- [TECS'10] P. Chandraiah, R. Dömer, *"Computer-Aided Recoding to Create Structured and Analyzable System Models"*, ACM Transactions on Embedded Computing Systems, accepted for publication December 21, 2009.