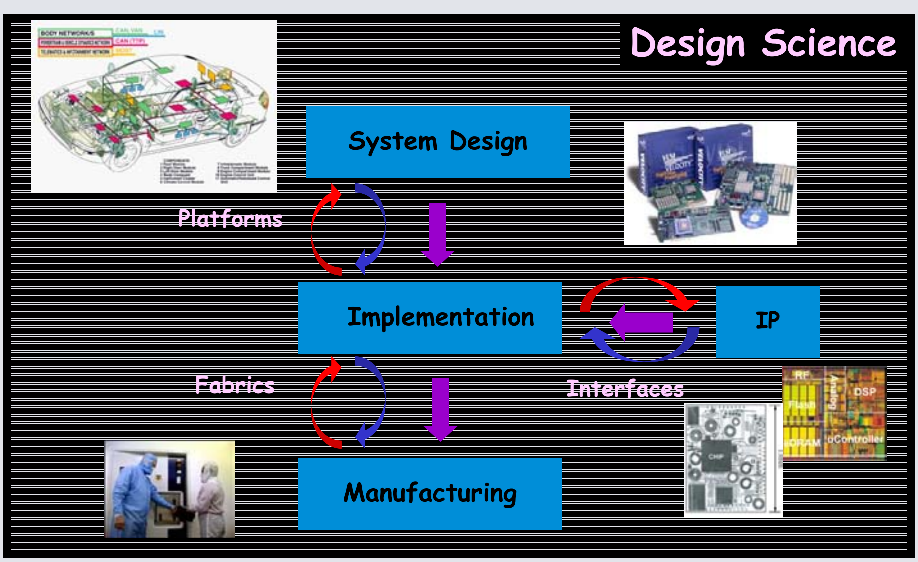
Design of Embedded Systems: Methodologies, Tools and Applications

Alberto Sangiovanni-Vincentelli
Dept. of EECS
University of California
Berkeley

UC Berkeley: Chess
Vanderbilt University: ISIS
University of Memphis: MSI

Foundations of Hybrid and Embedded Software Systems

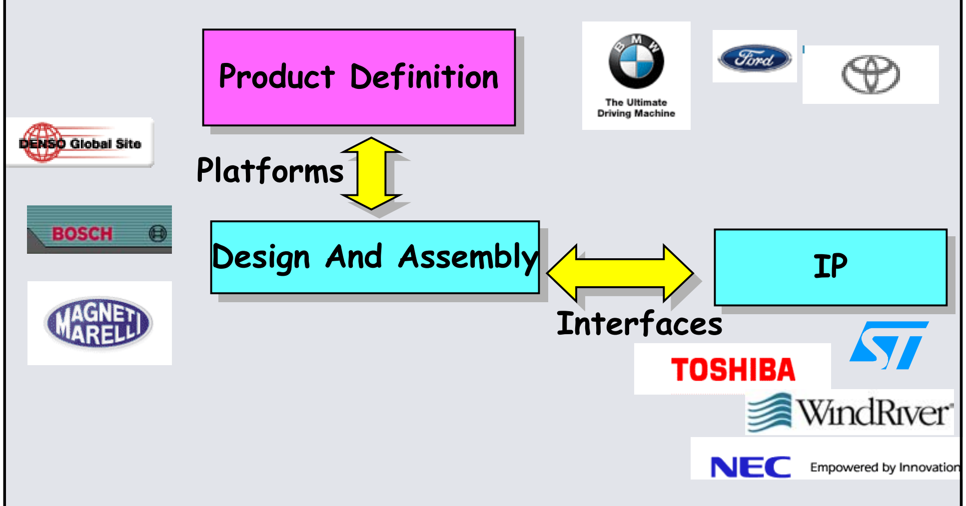
Disaggregation:
Electronic Systems Design Chain



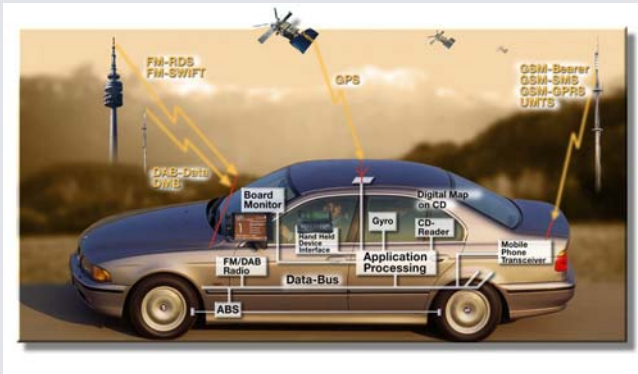
Outline

- Automotive Applications
- Distributed System Design Methodology and Flow
- Platform-based Design
- UAV Control Example
- Metropolis

The Automotive Electronic Design Chain

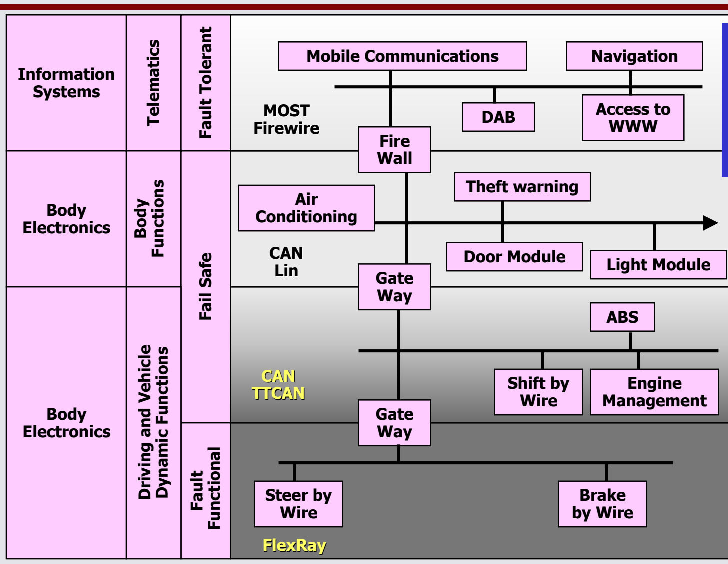


Automotive Supply Chain: Car Manufacturers



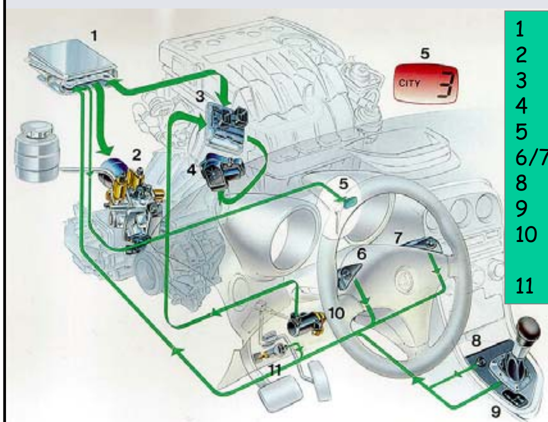
- Product Specification & Architecture Definition
  (e.g., determination of Protocols and Communication standards)
- System Partitioning and Subsystem Specification
- Critical Software Development
- System Integration

Electronics for the Car: A Distributed System



Today, more than 80 Microprocessors and millions of lines of code

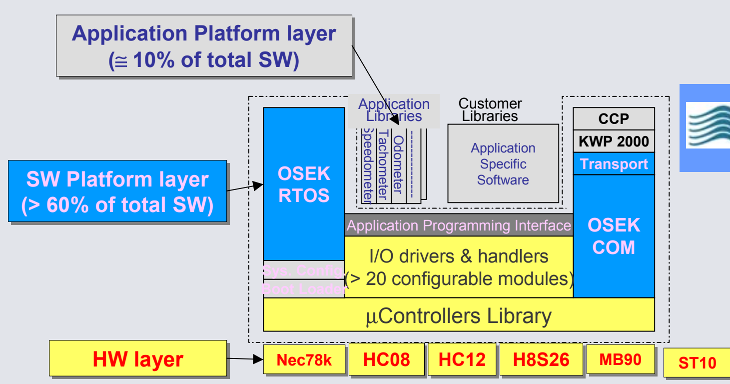
Automotive Supply Chain: Tier 1 Subsystem Providers



1 Transmission ECU
2 Actuation group
3 Engine ECU
4 DBW
5 Active shift display
6/7 Up/Down buttons
8 City mode button
9 Up/Down lever
10 Accelerator pedal position sensor
11 Brake switch

- Subsystem Partitioning
- Subsystem Integration
- Software Design: Control Algorithms, Data Processing
- Physical Implementation and Production

Automotive Supply Chain: Tier 2 Platform & IP Providers



- "Software" platform: RTOS and communication layer
- "Hardware" platform: Hardware and IO drivers

Complexity, Quality, Time-to-Market: TODAY

	PWT UNIT	BODY GATEWAY	INSTRUMENT CLUSTER	TELEMATIC UNIT
				
MEMORY	256 KB	128 KB	184 KB	8 MB
LINES OF CODE	50.000	30.000	45.000	300.000
PRODUCTIVITY	6 LINES/DAY	10 LINES/DAY	6 LINES/DAY	10 LINES/DAY*
RESIDUAL DEFECT RATE @ END OF DEV	3000 PPM	2500 PPM	2000PPM	1000 PPM
CHANGING RATE	3 YEARS	2 YEARS	1 YEAR	< 1 YEAR
DEV. EFFORT	40 MAN-YEAR	12 MAN-YEAR	30 MAN-YEAR	200 MAN-YEAR
VALIDATION TIME	5 MONTHS	1 MONTH	2 MONTHS	2 MONTHS
TIME TO MARKET	24 MONTHS	18 MONTHS	12 MONTHS	< 12 MONTHS

* C^{++} CODE

FABIO ROMEO, Magneti-Marelli
Design Automation Conference, Las Vegas, June 20th, 2001

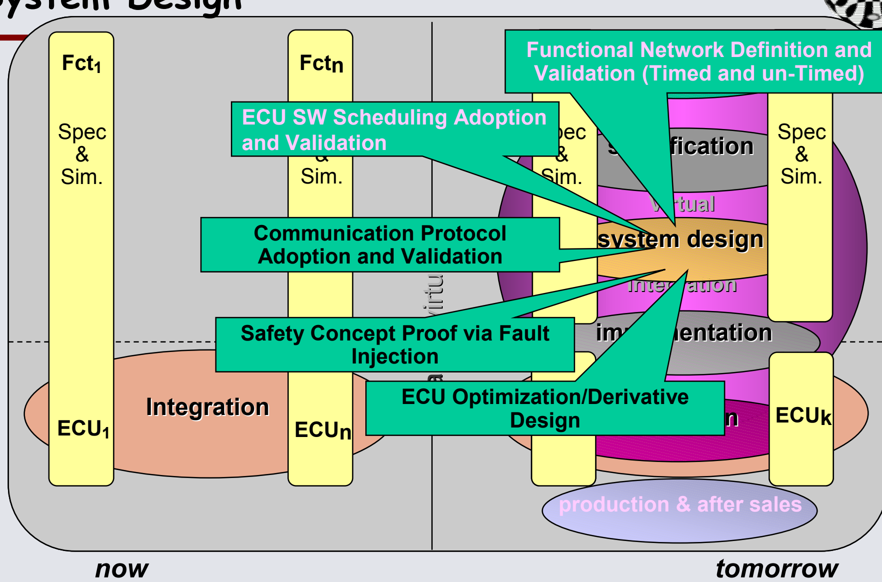
Embedded Software Design: Our Take

- Embedded Software Design must not be seen as a problem in isolation, it is an, albeit essential, aspect of *EMBEDDED SYSTEM DESIGN*
- Our vision is to change the way in which ESW is developed today by linking it:
  - Upwards in the abstraction layers to system functionality
  - Downwards in the programmable platforms that support it thus providing the means to verify whether the constraints posed on Embedded Systems are met.

Outline

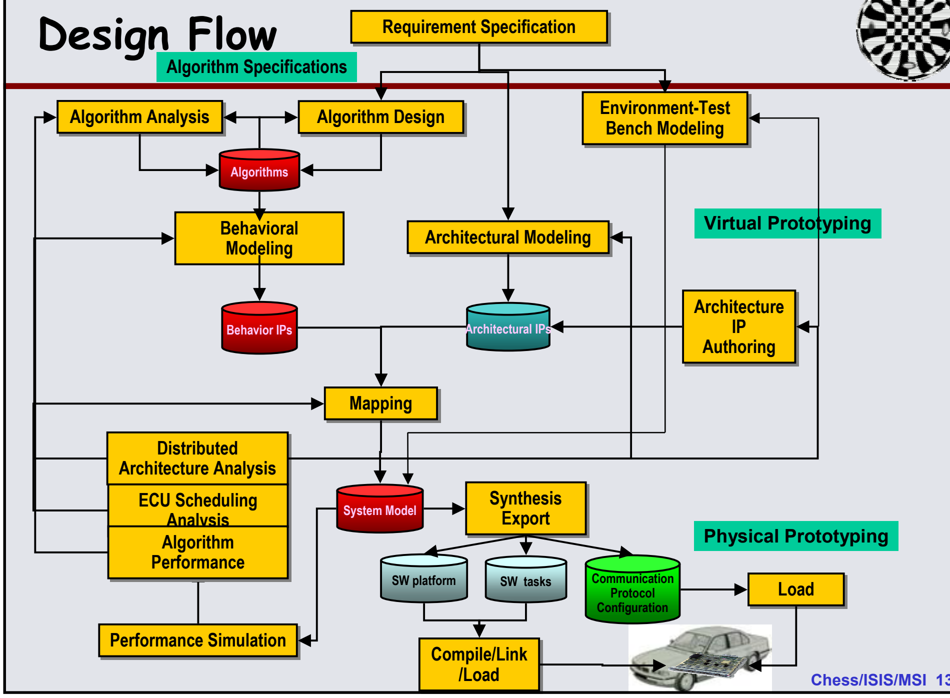
- Automotive Applications
- Distributed System Design Methodology and Flow
- Platform-based Design
- UAV Control Example
- Metropolis

Virtual Integration is key for Distributed System Design



Source BMW

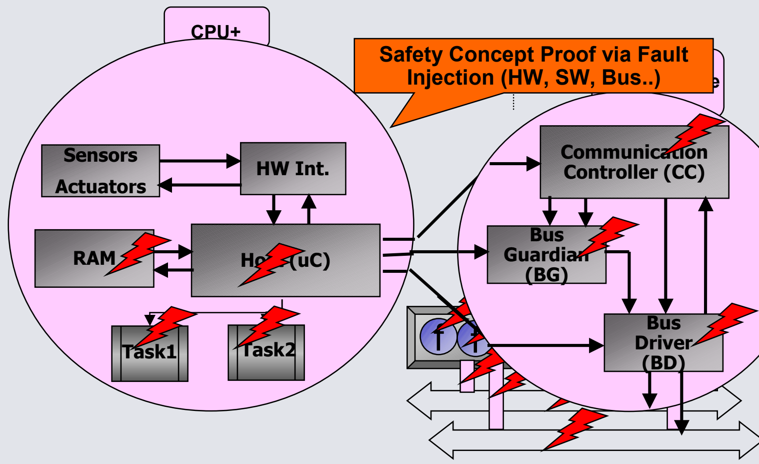
Design Flow



Focus on Safety-Critical Real Time

- Most challenging problem
- Needs tight integration between algorithms and implementation
- Constraints include timing and fault tolerance
- Fault tolerance can be addressed at all levels of abstraction

Safety Critical Issues: Fault Analysis



DRAFTS: Distributed Real-time Applications Fault Tolerant Scheduling

- Automatic (off-line) synthesis of fault tolerant schedules for periodic algorithms on a distributed architecture
- Automatic (off-line) verification that all intended faults are covered

Long-term goals:

- Design Methodology for Safety Critical Distributed Systems
- Manage the design complexity of modern Drive-By-Wire applications

C. Pinello, UCB, T. Demmeler and J. Ehret, BMW

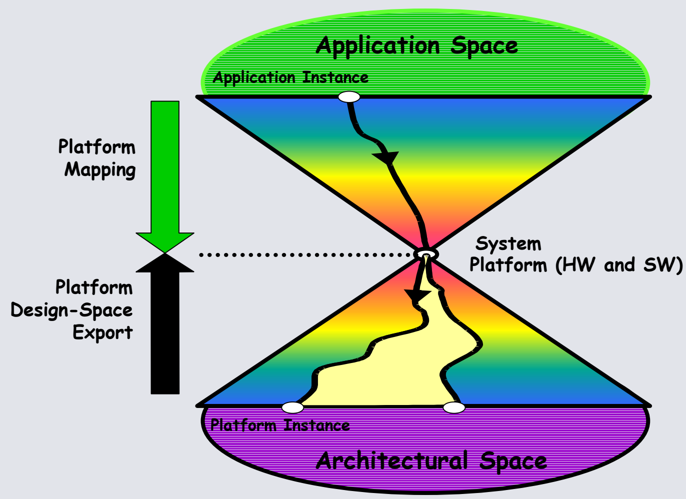
DRAFTS Strategy

- Identify critical functionality and possible faults
- Replicate critical functionality to withstand faults
- Exploit architecture redundancy to speed-up execution (in absence of faults)
- Functional Verification that all intended faults are covered

Outline

- Automotive Applications
- Distributed System Design Methodology and Flow
- Platform-based Design
- UAV Control Example
- Metropolis

ASV Triangles



Platforms: Evolution

In general, a platform is an abstraction layer that covers a ***number of possible refinements into a lower level***. The platform representation is a library of components including interconnects from which the lower level refinement can choose.

Principles of Platform methodology: Meet-in-the-Middle

- Top-Down:
  - Define a set of abstraction layers
  - From specifications at a given level, select a solution (controls, components) in terms of *components (Platforms)* of the following layer and propagate constraints
- Bottom-Up:
  - Platform components (e.g., micro-controller, RTOS, communication primitives) at a given level are abstracted to a higher level by their functionality and a set of parameters that help guiding the solution selection process. The selection process is equivalent to a covering problem if a common semantic domain is used.

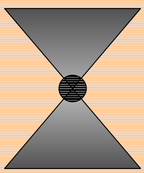
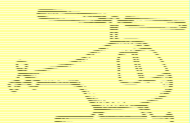
Outline

- Automotive Applications
- Distributed System Design Methodology and Flow
- Platform-based Design
- UAV Control Example
- Metropolis

Platform-Based Design of Unmanned Aerial Vehicles (source: J. Liebman)



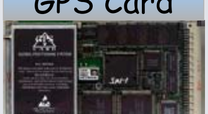
I	II	III	
Platform-Based Design	UAV System	Synchronous Embedded Control	Synchronous Platform Based UAV Design

UAV System: Sensor Overview

R-50 Hovering

GPS Card



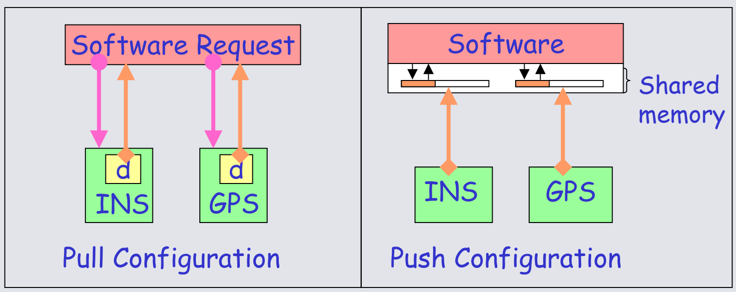
GPS Antenna

- Goal: basic autonomous flight
  - Need: UAV with allowable payload
  - Need: combination of GPS and Inertial Navigation System (INS)
- GPS (senses using triangulation)
  - Outputs *accurate* position data
  - Available at *low rate* & has jamming
- INS (senses using accelerometer and rotation sensor)
  - Outputs estimated position with *unbounded drift* over time
  - Available at *high rate*
- Fusion of GPS & INS provides needed high rate and accuracy

INS

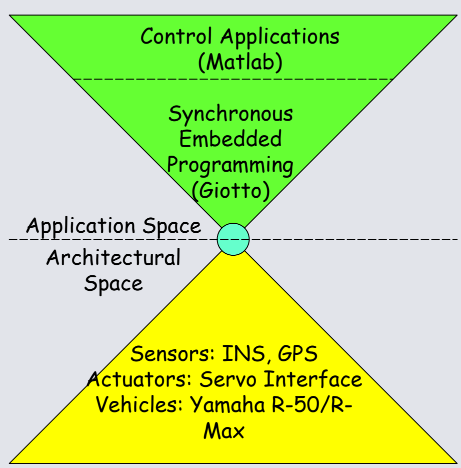
UAV System: Sensor Configurations

- Sensors may *differ* in:
  - Data formats, initialization schemes (usually requiring some bit level coding), rates, accuracies, data communication schemes, and even data types
- Differing Communication schemes requires the most custom written code per sensor



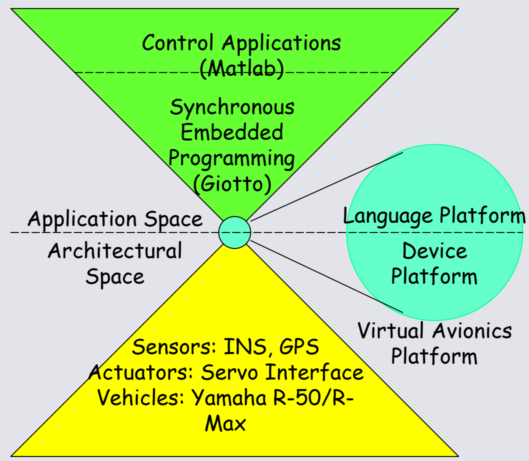
Platform Based Design for UAVs

- Goal
  - Abstract details of sensors, actuators, and vehicle hardware from control applications
- How?
  - Synchronous Embedded Programming Language (i.e. Giotto) Platform



Platform Based Design for UAVs

- Device Platform
  - Isolates details of sensor/actuators from embedded control programs
  - Communicates with each sensor/actuator according to its own data format, context, and timing requirements
  - Presents an API to embedded control programs for accessing sensors/actuators
- Language Platform
  - Provides an environment in which synchronous control programs can be scheduled and run
  - Assumes the use of generic data formats for sensors/actuators made possible by the Device Platform



Outline

- Automotive Applications
- Distributed System Design Methodology and Flow
- Platform-based Design
- UAV Control Example
- Metropolis

Metropolis Framework

Application-specific methodologies

Multi-media, wireless communication, mechanical controls, processors

Meta-model Library

- Models of computation

Infrastructure

- Metropolis meta-model
  - language
  - modeling mechanisms
- Meta-model compiler

Meta-model Library

- Architecture platforms

Tools

Simulator | QSS | PIG | STARS | SPIN | ...

Metropolis Project: main participants

- UC Berkeley **(USA)**: methodologies, modeling, formal methods
- Cadence Berkeley Labs **(USA)**: methodologies, modeling, formal methods
- Politecnico di Torino **(Italy)**: modeling, formal methods
- Universitat Politecnica de Catalunya **(Spain)**: modeling, formal methods
- Philips Research **(Netherlands)**: methodologies **(multi-media)**
- Nokia **(USA, Finland)**: methodologies **(wireless communication)**
- BWRC **(USA)**: methodologies **(wireless communication)**
- BMW **(USA)**: methodologies **(fault-tolerant automotive controls)**
- Intel **(USA)**: methodologies **(microprocessors)**
- STMicroelectronics **(France, Italy)**: methodologies **(wireless platforms)**
- Cypress **(USA)**: methodologies **(network processors, pSOC, all projects)**

Metropolis meta-model

Concurrent specification with a formal execution semantics:

- **Computation** : $f : X \rightarrow Z$
  - **proc**

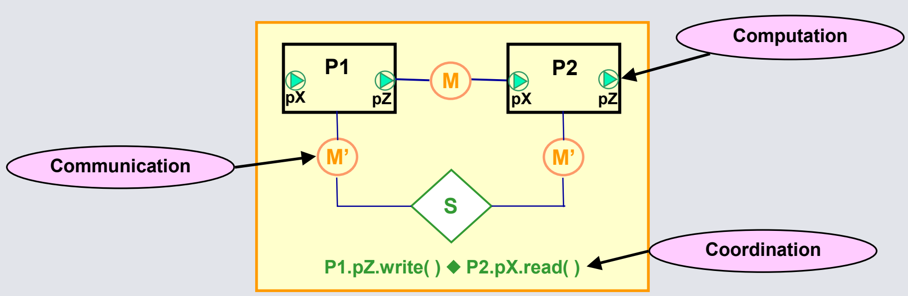
Key difference with respect to UML, SystemC, …!!!

- **Commu**
  - **medium** : defines *states* and *methods*

- **Coordination** : constraints over concurrent actions
  - **quantity** : annotation of each event (time, energy, memory, …)
  - **logic** : relates events and quantities, defines axioms on quantities
  - **quantity-manager** : algorithm to realize annotation subject to relational constraints

Metropolis Meta-Model

- Must describe objects at different levels of abstraction
  - Do not commit to the semantics of any particular model of computation
- Define a set of "building blocks"
  - specifications with many useful MoCs can be described using the building blocks
  - Processes, communication media and schedulers separate computation, communication and coordination



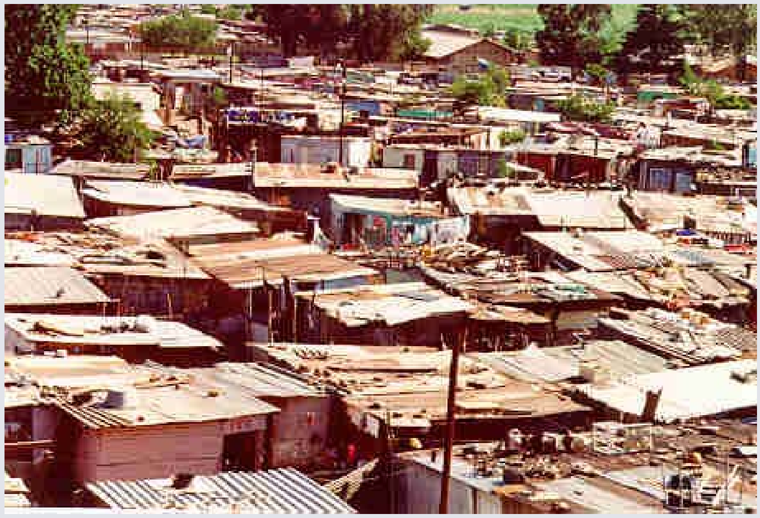
Supporting Theory

- Provide a semantic foundations for integrating different models of computation
  - Independent of the design language
  - Not just specific to the Metropolis meta-model
- Maximize flexibility for using different levels of abstraction
  - For different parts of the design
  - At different stages of the design process
  - For different kinds of analysis
- Support many forms of abstraction
  - Model of computation (model of time, synchronization, etc.)
  - Scoping
  - Structure (hierarchy)

Concluding Remarks

- Applications are critical to drive research and to test quality of results
- Safety-critical Real Time emphasis
- Rigorous methodology for distributed systems
- General framework to express designs at all levels of hierarchy and to support integration of foreign tools and designs

Embedded Software: Today



Embedded Software: Future?