Part 3: Models of Computation

- FSMs
- Discrete Event Systems
- CFSMs
- Data Flow Models
- Petri Nets
- The Tagged Signal Model
- Synchronous Languages and De-synchronization
- Heterogeneous Composition: Hybrid Systems and Languages
- Interface Synthesis and Verification
- Trace Algebra, Trace Structure Algebra and Agent Algebra

Design

- From an idea…
- … build something that performs a certain function
- Never done directly:
  - some aspects are not considered at the beginning of the development
  - the designer wants to explore different possible implementations in order to maximize (or minimize) a cost function
- Models can be used to reason about the properties of an object

Formalization

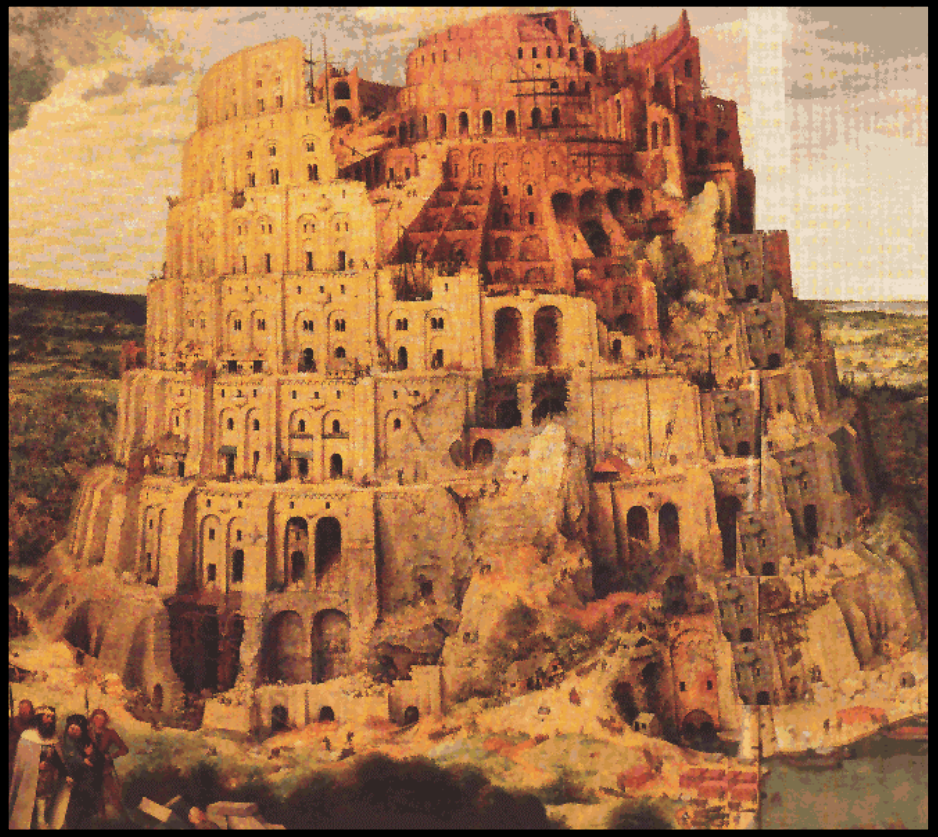
- Model of a design with precise unambiguous semantics:
- Implicit or explicit relations: inputs, outputs and (possibly) state variables
- Properties
- “Cost” functions
- Constraints

Formalization of Design + Environment = closed system of equations and inequalities over some algebra.

Models of Computation: And There are More...

- Continuous time (ODEs)
- Spatial/temporal (PDEs)
- Discrete time
- Rendezvous
- Synchronous/Reactive
- Dataflow
- ...

Each of these provides a formal framework for reasoning about certain aspects of embedded systems.



Tower of Babel, Bruegel, 1563

Model Of Computation

Definition: **A mathematical description that has a syntax and rules for computation of the behavior described by the syntax (semantics). Used to specify the semantics of computation and concurrency.**

Examples: Finite State Machine, Turing Machine, differential equation

An MoC allows:

- To capture unambiguously the required functionality
- To verify correctness of the functional specification wrt properties
- To synthesize part of the specification
- To use different tools (all must “understand” the model)
- MOC needs to
  - be powerful enough for application domain
  - have appropriate synthesis and validation algorithms

Usefulness of a Model of Computation

- Expressiveness
- Generality
- Simplicity
- Compilability/ Synthesizability
- Verifiability

The Conclusion

One way to get all of these is to mix diverse, simple models of computation, while keeping compilation, synthesis, and verification separate for each MoC. To do that, we need to understand these MoCs relative to one another, and understand their interaction when combined in a single system design.

Common Models of Computation

- Finite State Machines
  - finite state
  - no concurrency nor time
- Data-Flow
  - Partial Order
  - Concurrent and Determinate
  - Stream of computation
- Discrete-Event
  - Global Order (embedded in time)
- Continuous Time
- The behavior of a design in general is described by a composition

Control versus Data Flow

- Fuzzy distinction, yet useful for:
  - specification (language, model, ...)
  - synthesis (scheduling, optimization, ...)
  - validation (simulation, formal verification, ...)
- Rough classification:
  - control:
    - don't know when data arrive (quick reaction)
    - time of arrival often matters more than value
  - data:
    - data arrive in regular streams (samples)
    - value matters most

Control versus Data Flow

- Specification, synthesis and validation methods emphasize:
  - for control:
    - event/reaction relation
    - response time

      (Real Time scheduling for deadline satisfaction)
    - *priority* among events and processes
  - for data:
    - functional dependency between input and output
    - memory/time efficiency

      (Dataflow scheduling for efficient pipelining)
    - all events and processes are *equal*

Telecom/MM applications

- Heterogeneous specifications including
  - data processing
  - control functions
- Data processing, e.g. encryption, error correction…
  - computations done at regular (often short) intervals
  - efficiently specified and synthesized using DataFlow models
- Control functions (data-dependent and real-time)
  - say when and how data computation is done
  - efficiently specified and synthesized using FSM models
- Need a common model to perform global system analysis and optimization

Reactive Real-time Systems

- Reactive Real-Time Systems
  - "React" to external environment
  - Maintain permanent interaction
  - Ideally never terminate
  - timing constraints (real-time)
- As opposed to
  - transformational systems
  - interactive systems

Models Of Computation for reactive systems

- We need to consider essential aspects of reactive systems:
  - time/synchronization
  - concurrency
  - heterogeneity
- Classify models based on:
  - how specify behavior
  - how specify communication
  - implementability
  - composability
  - availability of tools for validation and synthesis

Models Of Computation for reactive systems

- Main MOCs:
  - Communicating Finite State Machines
  - Dataflow Process Networks
  - Petri Nets
  - Discrete Event
  - (Abstract) Codesign Finite State Machines
- Main languages:
  - StateCharts
  - Esterel
  - Dataflow networks