From Programs to Systems: Building a Smarter World

I&C Research Day
EPFL
June 21, 2012

Joseph Sifakis
RiSD Lab, EPFL
Exponential technological progress and new applications increasingly shift focus from programs to systems
From Programs to Systems

System Design

Three Grand Challenges
  ▪ Marrying Physicality and Computation
  ▪ Component-based Design
  ▪ Adaptivity

A System-centric Vision for CS
From Programs to Systems – Systems Everywhere

Systems integrating **software and hardware** jointly and specifically designed to provide given functionalities, which are often **critical**.
From Programs to Systems – Significant Differences

**Program**

- \( f(i) = o \)

**System**

- \( f_i(i_i) = o_i \)

- Physical Environment

**Program characteristics**

- I/O values
- Terminating
- Deterministic
- Platform-independent behavior
- Theory of computation

**System characteristics**

- I/O streams of values
- Non-terminating
- Non-predictable
- Platform-dependent behavior
- No theory!
From Programs to Systems – Significant Differences

- I/O values
- Terminating
- Deterministic
- Platform-independent behavior
- Theory of computation

- I/O streams of values
- Non-terminating
- Non-predictable
- Platform-dependent behavior
- No theory!
Systems are hard to design due to unpredictable and subtle interactions with their environment, rather than to complex data and algorithms.
New trends break with traditional Computing Systems Engineering. It is hard to jointly meet technical requirements such as:

- **Reactivity**: responding within known and guaranteed delay
e.g. flight controller

- **Autonomy**: provide continuous service without human intervention
  e.g. no manual start, optimal power management

- **Dependability**: guaranteed minimal service in any case
  e.g. attacks, hardware failures, software execution errors

- **Scalability**: at runtime or evolutionary growth (linear performance increase with resources)
e.g. reconfiguration, scalable services

...and also take into account economic requirements for optimal cost/quality

**Technological challenge:**
Capacity to build systems of guaranteed functionality and quality, at acceptable costs.
From Programs to Systems

System Design

Three Grand Challenges

- Marrying Physicality and Computation
- Component-based Design
- Adaptivity

A System-centric Vision for CS
Design is a universal concept, a par excellence intellectual activity leading to artifacts meeting given requirements.

**Easy Apple Pie**

**RECIPE (Program)**
- Put apples in pie plate;
- Sprinkle with cinnamon and 1 tablespoon sugar;
- In a bowl mix 1 cup sugar, flour and butter;
- Blend in unbeaten egg, pinch of salt and the nuts;
- Mix well and pour over apples;
- Bake at 350 degrees for 45 minutes

**INGREDIENTS (Resources)**
- 1 pie plate buttered
- 5 or 6 apples, cut up
- ¾ c. butter, melted
- 1 c. flour
- ½ c. chopped nuts
- 1 tsp cinnamon
- 1 tbsp sugar
- 1 c. Sugar
System Design – Two Main Gaps

Correctness?  Correctness?

Requirements (declarative)

Application SW (executable)

System (HW+SW)

Proceduralization

Materialization
Trustworthiness requirements express assurance that the designed system can be trusted that it will perform as expected despite

- **HW failures**
- **Design Errors**
- **Environment Disturbances**
- **Malevolent Actions**

Optimization requirements are quantitative constraints on resources such as time, memory and energy characterizing

1) **performance** e.g. throughput, jitter and latency;
2) **cost** e.g. storage efficiency, processor utilizability
3) **tradeoffs** between performance and cost
Trustworthiness requirements characterize qualitative correctness – a state is either trustworthy or not.

Optimization requirements characterize execution sequences.

The two types of requirements are often conflicting. System design should determine tradeoffs driven by cost-effectiveness and technical criteria.

Trustworthiness vs. Optimization
System Design – Levels of Criticality

Safety critical: a failure may be a catastrophic threat to human lives.

Security critical: harmful unauthorized access.

Mission critical: system availability is essential for the proper running of an organization or of a larger system.

Best-effort: optimized use of resources for an acceptable level of trustworthiness.
### System Design – Status and Vision

**TODAY**

- We master – at a high cost – two types of systems which are difficult to integrate:
  - Safety and/or security critical systems of low complexity
    - Flight controller, smart card
  - Complex « best effort » systems
    - Telecommunication systems, web-based applications

**TOMORROW**

- We need:
  - Affordable critical systems in areas such as transport, health, energy management
  - Successful integration of heterogeneous mixed criticality systems
    - Internet of Things
    - Intelligent Transport Systems
    - Smart Grids
    - « Ambient Intelligence»

This vision is currently unattainable – Trustworthiness should be a guiding concern throughout the design process.
Traditional systems engineering disciplines are based on solid theory for building artefacts with predictable behaviour over their life-time.

Computing systems engineering lacks similar constructivity results:

- only partial answers to particular design problems
- predictability is hard to guarantee at design time
- *a posteriori* validation remains essential for ensuring correctness
Design of large IT systems is a risky undertaking, mobilizing hundreds of engineers over several years.

Obstacles
- Complexity – mainly for building systems by reusing existing components
- Requirements are often incomplete, and ambiguous (specified in natural language)
- Design approaches are empirical and based on the expertise and experience of teams

Consequences
- Poor understanding of systems dynamics and emergent properties
- Patches are used to fix vulnerability and safety problems – this is not compatible with enhanced autonomy and reactivity.

You can’t fix, what you don’t understand!
System Design – Checking Trustworthiness

Model

Should be:
- **faithful** e.g. whatever property is satisfied for the model holds for the real system
- generated **automatically** from system descriptions

Verification Method

Requirements

Should be:
- **consistent** e.g. there exists some model satisfying them
- **complete** e.g. they tightly characterize the system’s behavior

YES, NO, DON’T KNOW

- As a rule, for infinite state models all non trivial properties are undecidable e.g. x<100
- Intrinsically high complexity for finite state models (state explosion problem)
Verification techniques

- are mainly and extensively applied for detecting design errors in large hardware and medium size software systems for trustworthiness requirements that can be formalized.

- cannot be applied to systems as we have no faithful modeling techniques - interaction between application software, the underlying HW platform and external environment is ill-understood.

- suffer inherent complexity limitations
- From Programs to Systems
- System Design
- Three Grand Challenges
  - Marrying Physicality and Computation
    - Component-based Design
    - Adaptivity
- A System-centric Vision for CS
Marrying Physicality and Computation

**HW Platform:**
- CPU speed
- memory
- power
- failure rates
- temperature

**Environment:**
- deadlines
- jitter
- throughput

**SYSTEM**

**Software:**
- application SW
- middleware
- OS
Marrying Physicality and Computation

**System Environment:**
- deadlines
- jitter
- throughput

**Hardware Platform:**
- CPU speed
- memory
- power
- failure rates
- temperature

**Software:**
- application SW
- middleware
- OS

SW Design cannot ignore HW design
Marrying Physicality and Computation

**HW Platform:**
- CPU speed
- memory
- power
- failure rates
- temperature

**Environment:**
- deadlines
- jitter
- throughput

**Software:**
- application SW
- middleware
- OS

**SYSTEM**

SW Design cannot ignore control design
We need to revisit and revise computing to integrate methods from EE and Control.
- From Programs to Systems
- System Design
- Three Grand Challenges
  - Marrying Physicality and Computation
  - Component-based Design
  - Adaptivity
- A System-centric Vision for CS
Component-based Design

- Building complex systems by composing a small number of types of components is essential for any engineering discipline.
- This confers numerous advantages such as mastering complexity, enhanced productivity and correctness through reuse.
- Component composition orchestrates interactions between components. It lies at the heart of the system integration challenge.

No Common Component Model for Systems Engineering!
System designers deal with a large variety of components, with different characteristics, from a large variety of viewpoints, each highlighting different dimensions of a system.

Consequences:
- Using semantically unrelated formalisms e.g. for programming, HW description and simulation, breaks continuity of the design flow and jeopardizes its coherency.
- Costly system development decoupled from validation and evaluation.
Compositionality rules guarantee that composite components inherit essential properties of constituent components.

We need compositionality results for progress properties such as deadlock-freedom and liveness.
**Component-based Design – Correctness-by-Construction**

Composability rules guarantee that adding new components does not jeopardize essential properties of integrated components.

Feature interference in OS, middleware, telecommunication systems and web Services are all due to lack of composability!
From Programs to Systems

System Design

Three Grand Challenges
   ▪ Marrying Physicality and Computation
   ▪ Component-based Design
   ▪ Adaptivity

A Vision for CS
Systems must provide services meeting given requirements in interaction with intrinsically uncertain (non-deterministic) environments.

Adaptivity consists in using control-based techniques to ensure correctness despite uncertainty.
Strong AI Vision – Intelligence Matching Human Intelligence

- Considers that human intelligence can be so precisely described that it can be matched by a machine.
- Supercomputers are used to perform heavy computational tasks e.g. problem solving or reasoning.

IBM Deep Blue (1997)

IBM “WATSON” (2011)
The Jeopardy!-playing question answering system
Uncertainty can be measured as the difference between average and extreme system behavior. It has two main sources:

- External environment e.g. varying throughput, attacks
- Hardware platforms e.g. manufacturing errors or aging, varying execution times due to layering, caches, speculative execution.

Intrinsic uncertainty is aggravated by lack of precise modeling and analysis techniques (estimated uncertainty)

![Diagram showing execution times and uncertainty bounds](image_url)
Adaptivity – Critical vs. Best Effort Engineering

Two diverging design paradigms

BAD STATES

Critical systems engineering based on worst-case analysis and static resource reservation e.g. hard real-time, massive redundancy.

Leads to over-provisioned systems

ERROR STATES

Best effort engineering based on average case analysis and dynamic resource management e.g. to optimize speed, memory, bandwidth, power.
No guaranteed availability
Adaptivity – Mixed Criticality Systems

Separation between critical and best-effort designs is no longer affordable. In a car

- more than 50 ECU’s at different criticality levels designed separately
- federated architectures by using networks
- poor global reliability and high development costs

Towards integrated mixed criticality systems allowing applications at different criticality levels to interact and co-exist on the same computational platform e.g. IMA for avionic systems, AUTOSAR for automotive systems.

Adaptive control techniques

- allow meeting both trustworthiness and optimization requirements in mixed criticality systems
  - first and foremost critical applications use with high priority a provably sufficient amount of global shared resources
  - secondarily, non-critical applications are served in a best-effort mode
- are extensively and increasingly applied in many areas e.g. robotics, multimedia systems, networks, data mining
Adaptivity – Adaptive Control

Learning
Movie would have been better ...

Management of objectives
Go to: 1) Stadium 2) Movie 3) Restaurant

Planning
From Programs to Systems

System Design

Three Grand Challenges

- Marrying Physicality and Computation
- Component-based Design
- Adaptivity

A System-centric Vision for CS
A System-Centric Vision for CS

System Design

- Raises a multitude of deep theoretical problems such as the conceptualization of needs in a given area and their effective transformation into correct artifacts.

- has attracted little attention from scientific communities and is relegated to second class status
  - design is by its nature multi-disciplinary and requires consistent integration of heterogeneous system models supporting different levels of abstraction including logics, algorithms and programs as well as physical system models.

- is central to CS. Awareness on its centrality is a chance to reinvigorate CS research and build new scientific foundations matching the needs for increasing system integration and new applications.
A System-Centric Vision for CS – The Frontiers

Mathematics
Logic

Physics
Computer Science
Biology
We need theory, methods and tools for climbing up-and-down the abstraction hierarchy
A System-Centric Vision for CS – The Frontiers

**Physics**
- Deals with phenomena of the «real» physical world (transformations of matter and energy)
- Focuses mainly on the discovery of physical laws.
- Physical systems – Analytic models
- Continuous mathematics
- Differential equations - robustness
- Predictability for classical Physics
- Mature discipline

**Computer Sc.**
- Deals with the representation and transformation information
- Focuses mainly on the construction of systems
- Computing systems – Machines
- Discrete mathematics – Logic
- Automata, Algorithms, Complexity Theory
- Verification, Testing,
- Young fast evolving discipline
Artificial vs. Natural Intelligence
Living organisms intimately combine interacting physical and computational phenomena that have a deep impact on their development and evolution

- Shared characteristics with computing systems
  - use of memory
  - distinction between hardware and software
  - use of languages

- Remarkable differences:
  - robustness of computation
  - built-in mechanisms for adaptivity
  - emergence of abstractions – concepts

Interactions and cross-fertilization

- Non von Neumann computing ↔ Neuromorphic, Cognitive Computing
- CAD methods&tools ⇒ Synthetic Biology
A System-Centric Vision for CS – Looking for Foundations

Theoretical research in CS often focuses on “nice theory” that is not always practically relevant

“Make everything as simple as possible, but not simpler”

“….. in the academic world the theories that are more likely to attract a devoted following are those that best allow a clever but not very original young man to demonstrate his cleverness.”

….. while practitioners propose ad hoc frameworks hardly amenable to formalization e.g. non-orthogonal concepts, ambiguous semantics

“Perfection is reached not when there is no longer anything to add, but when there is no longer anything to take away”
A System-Centric Vision for CS – Looking for Foundations

Is it possible to find a mathematically elegant and still practicable theoretical framework for system design?

Physics and Biology study a given “reality”

The key issue is discovering laws governing phenomena

"The most incomprehensible thing about the world is that it is at all comprehensible."

Computer Science mainly deals with building systems (artifacts)

The key issue is building correct systems cost-effectively
The physical world is part of our conditions of existence.

We mold the conditions of existence of the cyber-world.

Is everything for the best in the best of all possible cyber-worlds?
Thank You