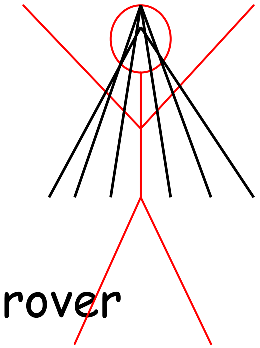
Formal Methods for Software Systems

Jeannette M. Wing

Assistant Director
Computer and Information Science and Engineering Directorate
NSF

HC-RTOS
Arlington, VA
9 July 2007

My Background and Interests



- V: Researcher in formal methods since 1980
  - Algebraic specifications
  - Interactive theorem provers: AFFIRM, InaJo, Reve,Larch Prover
  - Model checkers: SMV, SPIN, FDR
  - New spec languages, proof techniques, algorithms
- S: Applications in software and systems
  - Concurrent, multi-threaded systems: linearizable objects
  - Fault-tolerant distributed transactions: Avalon/C++, Avalon/CommonLisp
  - Object-oriented programming: behavioral subtyping
  - Networking: ipV6
  - Distributed file systems: AFS cache coherence
  - Storage systems: RAID error recovery
  - Security: authentication, attack graphs
  - Privacy: secrecy, confidentiality properties
- P: Cyber-physical platforms
  - Embedded systems in automobiles, airplanes, controllers, …
  - Tamper-resistant embedded systems

Why Formal Methods

- It's a proven success
  - Hardware companies, e.g., Intel, use it routinely
  - Software companies, e.g., Microsoft, use it increasingly, e.g., SLAM for device drivers
- Why?
  - Save lives
  - Save money
- Why now more than ever?
  - Systems continue to grow in complexity (size & functionality, time & space dimensions).
  - Interactions, especially unforeseen, between systems add to that complexity
  - Our daily lives (transportation, financial, medical, ….) rely on these interacting systems to function.
- Testing and simulation are not enough. Verification offers stronger guarantees and resuable models and theories.

Important Trends in Verification

- Lightweight formal methods [Jackson and Wing 96]
  - Laser beam vs. light bulb approach
    - Focus on one critical property (at a time)
    - Focus on one critical component (at a time)

- Verification as a debugging tool
  - Value of counterexamples: Hardware and protocol design
  - Most systems are not correct
  - Acceptance of both false positives and false negatives

- Integration of theorem proving, model checking, decision procedures (SAT), and program analyses

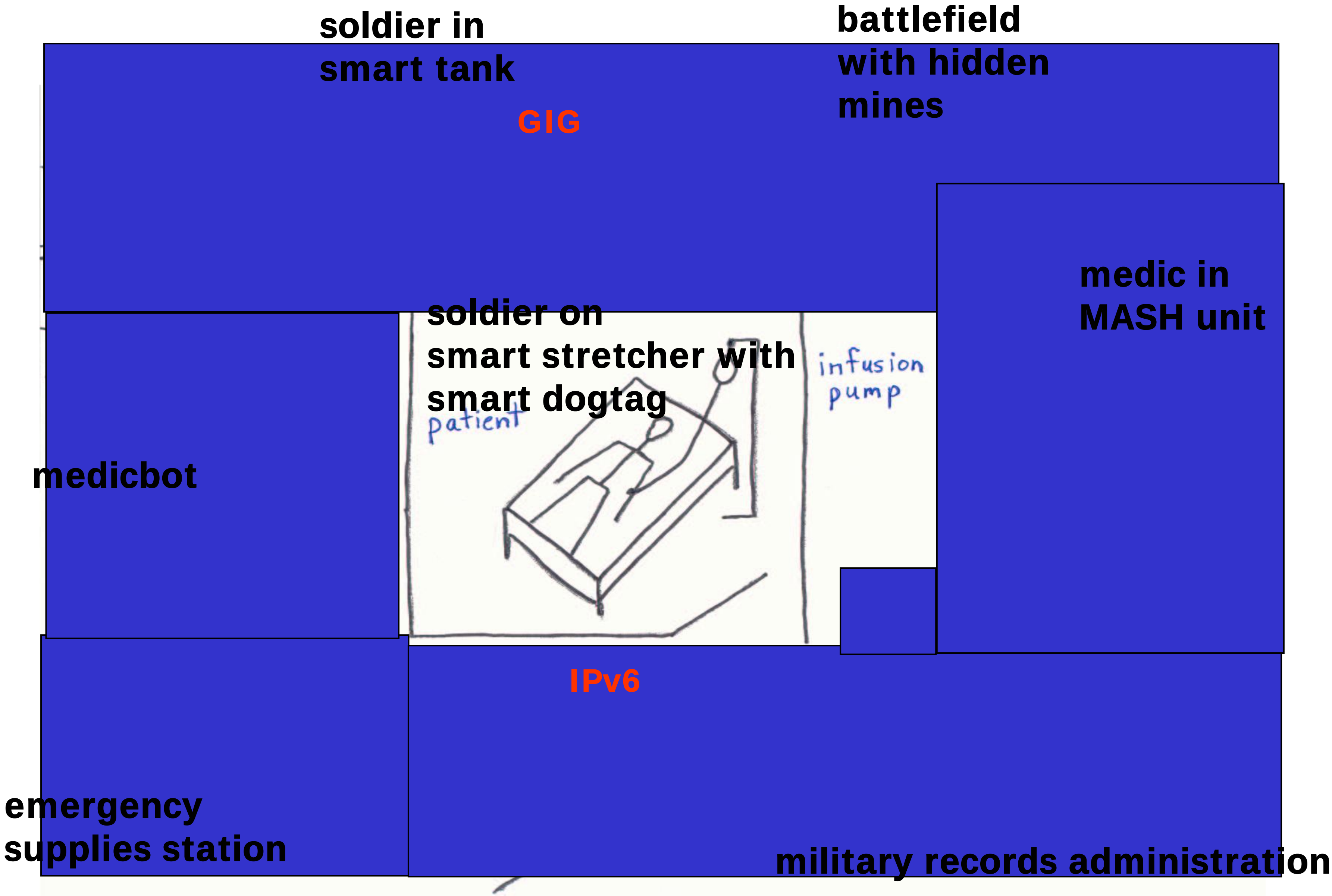
Important Trends in Systems

How can we build complex systems simply?

- Nature of tomorrow's systems
  - Dynamic, ever-changing, 24/7 reliability
  - Self-* (aware, diagnosing, healing, repairing, managing)
- Two important classes converging
  - Embedded
    - Networked architecture, e.g., sensor nets (see below)
    - Safety-critical apps, e.g., medical, automotive, aero&astro
    - Challenge: Reasoning about uncertainty, e.g., Human, Mother Nature, the Adversary
  - Pervasive and mobile
    - Focus on sensors and actuators, not just the devices and communication links
    - Prevalence of cell phones, iPods, RFIDs, …
    - Implications for HCI, embedded systems, sensor nets (see above)

Looking Ahead: Research Challenges

This is Not Science Fiction



Some Common Themes

- Cyber-physical systems
  - Safety-critical: infusion pump, smart cars, smart highways
- Networked systems
  - Heterogeneous: communications (wireless, broadband, ethernet, land lines, …) and components (iPhones, PDAs, laptops, workstations, storage systems, smart devices (e.g., robots)…)
- Distributed systems
  - Fault-tolerant, availability 24x7, reliable, secure
- Databases
  - Privacy: patient control vs. medical needs vs. admin convenience vs. legal constraints …
- Human-computing interfaces
  - Social acceptance: nursebot
- Scale: size, complexity

State of the Art

- Point solutions for some of these problems individually
  - Verify a few safety properties of a single blood infusion pump
  - Verify "no collision" for adaptive cruise control
  - Certifiable mobile code
- Open problems for the rest

- Point solutions for some of the integrated aspects
  - Transactional distributed databases
  - (Insecure) wireless, ad-hoc networking
- Open problems for the rest

- Point solutions work for small systems or as research prototypes
- Open issues: Scalability and interoperability requires concerted engineering and tech transition

Software Systems: Observation

- It's the software that effects this functional complexity.

Software Everywhere: Implications

- Hard to circumscribe ("Software Without Borders"
  - Little pieces, e.g., applets, scripts, program module
  - Big pieces, e.g., O/S, database, browser, mailer

- Hard to pin down
  - Ephemeral and elusive (mobile), both computation and data
  - Permanent, e.g., patient records

- Hard to identify owner (responsibility)
  - Untrusted: unknown source, unknown creator
  - Certified: trusted third-party

Software Systems Everywhere: Research Challenges

- We need new advances in software foundations.
  - What does "correctness" mean?
    - Factor in context of use, unpredictable environment, emergent properties, dynamism
  - What are the desired properties of and metrics for both software (e.g., weak compositionality) and systems (e.g., power)?
- We need new advances in formal models and logics for complex systems, e.g., hybrid systems.
  - For verification, simulation, prediction
- We need new advances in verification tools for systems builders and domain engineers
  - Push-button
  - Usable
  - Integrated with rest of system development process
- We need new engineering processes for creating software-intensive systems.
  - Traditional ones won't work.

Partnerships

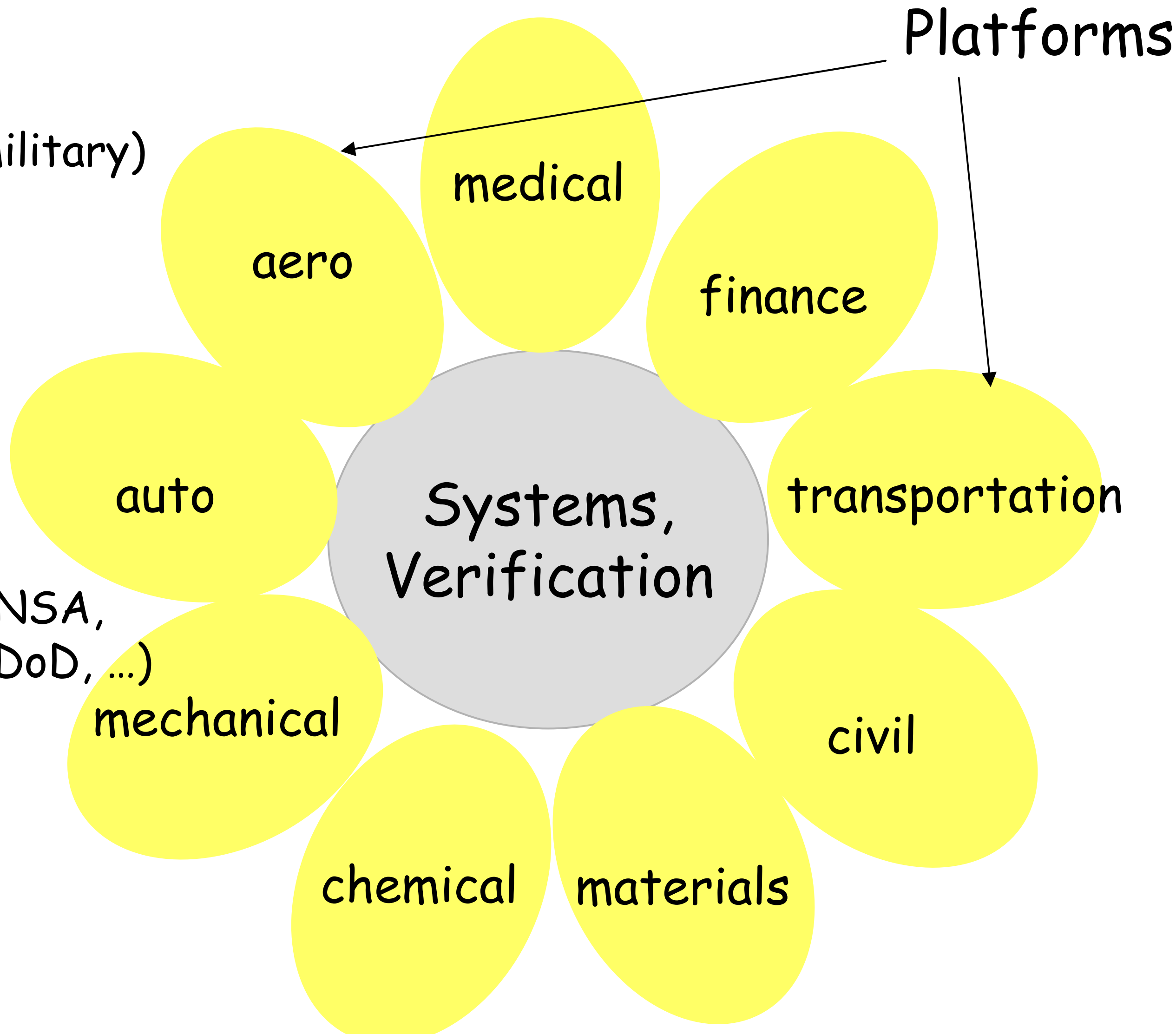
- Theoreticians, experimentalists, domain experts
- S-V, V-P, S-P, S-V-P
- Industry, Academia, Government
  - domain experts, domain problems
  - general solutions that work for specific problems

A Model for Expediting Progress

Industry
Gov't (e.g., military)

Industry
Gov't
Academia

Academia
Gov't (NSF, NSA,
NIH, DoD, …)



Thank you!