Trustworthy Self-Assembly:
A Use-Case for Distributed Runtime Verification

John Rushby

Computer Science Laboratory
SRI International
Menlo Park, California, USA
Introduction: Systems of Systems

- We’re familiar with systems built from components
- But increasingly, we see systems built from other systems
  - Systems of Systems
- The component systems have their own purpose
  - Maybe at odds with what we want from them
- And they generally have vastly more functionality than we require
  - Provides opportunities for unexpected behavior
  - Bugs, security exploits etc. (e.g., CarShark)
- Difficult when trustworthiness required
  - May need to wrap or otherwise restrict behavior of component systems
  - So, traditional integration requires bespoke engineering
Accidental Systems of Systems

- Whether intended or not, systems necessarily interact with their neighbors through the effect each has on the environment of the others
  - Stigmergic interactions
  - Particularly those involving the “plant”

- Unmanaged interactions can be deleterious

- Get emergent misbehavior

- So better if systems are open (to interactions) and adaptive

- Not all interactions can be pre-planned

- So systems need to self-integrate at runtime
Self-Assembling/Self-Integrating Systems

• Imagine systems that recognize each other and **spontaneously integrate**
  ◦ Possibly under the direction of an “integration app”
  ◦ Examples on next several slides

• As noted, systems often interact through shared “plant” **whether we want it or not (stigmergy)**
  ◦ Separate medical devices attached to same patient
  ◦ Car and roadside automation (autonomous driving and traffic lights)
    And it would be best if they “**deliberately**” integrated

• These systems need to “**self integrate**” or “**self assemble**”

• And we want the resulting system to be trustworthy

• That’s a tall order

• Note that desirable system properties can **break local ones through downward causation**
Scenarios

- I’ll describe some scenarios, mostly from medicine

- And most from Dr. Julian Goldman (Mass General)
  - “Operating Room of the Future” and
  - “Intensive Care Unit of the Future”

- There is Medical Device Plug and Play (MDPnP) that enables basic interaction between medical devices

- And the larger concept of “Fog Computing” to provide reliable, scaleable infrastructure for integration

- But I’m concerned with what the systems do together rather than the mechanics of their interaction
Anesthesia and Laser

- Patient under general anesthesia is generally provided enriched oxygen supply
- Some throat surgeries use a laser
- In presence of enriched oxygen, laser causes burning, even fire
- Want laser and anesthesia machine to recognize each other
- **Laser requests reduced oxygen** from anesthesia machine
- But...
  - Need to be sure laser is talking to anesthesia machine connected to this patient
  - Other (or faulty) devices should not be able to do this
  - Laser should light only if oxygen really is reduced
  - In emergency, need to enrich oxygen should override laser
Other Examples

- I’ll skip the rest in the interests of time

- But they are in the slides (marked SKIP)
• Very ill patients may be on a heart-lung machine while undergoing surgery

• Sometimes an X-ray is required during the procedure

• Surgeons turn off the heart-lung machine so the patient’s chest is still while the X-ray is taken

• Must then remember to turn it back on

• Would like heart-lung and X-ray mc’s to recognize each other

• X-ray requests heart-lung machine to stop for a while
  ◦ Other (or faulty) devices should not be able to do this
  ◦ Need a guarantee that the heart-lung restarts

• Better: heart lung machine informs X-ray of nulls
Patient Controlled Analgesia and Pulse Oximeter

- Machine for Patient Controlled Analgesia (PCA) administers pain-killing drug on demand
  - Patient presses a button
  - Built-in (parameterized) model sets limit to prevent overdose
    - Limits are conservative, so may prevent adequate relief
- A Pulse Oximeter (PO) can be used as an overdose warning
- Would like PCA and PO to recognize each other
- PCA then uses PO data rather than built-in model
- But that supposes PCA design anticipated this
- Standard PCA might be enhanced by an app that manipulates its model thresholds based on PO data
- But...
PCA and Pulse Oximeter (ctd.) SKIP

- Need to be sure PCA and PO are connected to same patient
- Need to cope with faults in either system and in communications
  - E.g., if the app works by blocking button presses when an approaching overdose is indicated, then loss of communication could remove the safety function
  - If, on the other hand, it must approve each button press, then loss of communication may affect pain relief but not safety
  - In both cases, it is necessary to be sure that faults in the blocking or approval mechanism cannot generate spurious button presses
- This is hazard analysis and mitigation at integration time
Blood Pressure and Bed Height

- Accurate **blood pressure sensors** can be inserted into intravenous (IV) fluid supply
- Reading needs correction for the **difference in height** between the sensor and the patient
- Sensor height can be standardized by the IV pole
- Some hospital beds have **height sensor**
  - Fairly **crude device** to assist nurses
- Can imagine an ICU where these data are available on the local network
- Then integrated by monitoring and alerting services
- But...
Blood Pressure and Bed Height (ctd.) SKIP

- Need to be sure bed height and blood pressure readings are from same patient

- Needs to be an ontology that distinguishes height-corrected and uncorrected readings

- Noise- and fault-characteristics of bed height sensor mean that alerts should be driven from changes in uncorrected reading

- Or, since, bed height seldom changes, could synthesize a noise- and fault-masking wrapper for this value

- Again, hazard analysis and mitigation at integration time
What’s the Problem?

- Could build all these as bespoke systems

- More interesting is the idea that the component systems discover each other, and self integrate into a bigger system

- Initially will need an extra component, the integration app to specify what the purpose should be

- But later, could be more like the way human teams assemble to solve difficult problems
  - Negotiation on goals, exchange information on capabilities, rules, and constraints

- I think this is how the Internet of Things will evolve
What’s the Problem? (ctd. 1)

- Since they were not designed for it
- It’s unlikely the systems fit together perfectly
- So will need shims, wrappers, adapters etc.
- So part of the problem is the “self” in self integration
- How are these adaptations constructed during self integration?
What’s the Problem? (ctd. 2)

- In many cases the resulting assembly needs to be trustworthy
  - Preferably do what was wanted
  - Definitely do no harm

- Even if self-integrated applications seem harmless at first, will often get used for critical purposes as users gain (misplaced) confidence
  - E.g., my Chromecast setup for viewing photos
  - Can imagine surgeons using something similar (they used Excel!)

- So how do we ensure trustworthiness?
Aside: System Assurance

- State of the art in system assurance is the idea of a safety case (more generally, an assurance case)
  - An argument that specified claims are satisfied, based on evidence (e.g., tests, analyses) about the system

- System comes with machine-processable online rendition of its assurance case
  - Not standard yet, but Japanese DEOS project does it
  - Essentially a proof, built on premises justified by evidence (see my AAA15 paper, cf. ones on Ontological Argument)

- Ideally: when systems self integrate, assurance case for the overall system is constructed automatically from the cases of the component systems

- Hard because safety often does not compose
  - E.g., because there are new hazards
  - Recall laser and anesthesia
What’s the Problem? (ctd. 3)

- While building the assurance case at self-integration time
- Likely must **eliminate** or **mitigate** some **hazards**
- May be able to do this by **wrappers**, or by **monitoring**

### Aside: the power of monitors
- A monitor can be very simple
- Can make a claim that it is **probably fault-free**
  - **This** is the claim that verification delivers
- Prob. of failure of **system** is then
  - **prob. of failure** of operational component times prob. **monitor** is fault-free
- Nb. **cannot** multiply probs. of failure
- See TSE 2012 paper by Littlewood and me

- **How do these wrappers and monitors get built?**
Models At Runtime (M@RT)

- If systems are to adapt to each other
- And wrappers and monitors are to be built at integration-time
- Then the systems need to know something about each other
- One way is to exchange models
  - Machine-processable (i.e., formal) description of some aspects of behavior, claims, assumptions
- This is Models at RunTime: M@RT
- When you add aspects of the assurance case, get Safety Models at RunTime: SM@RT (Trapp and Schneider)
- Most recent in a line of system integration concepts
  - Open Systems, Open Adaptive Systems, System Oriented Architecture
Four Levels of SM@RT

- Due to Trapp and Schneider

- **Safety Certificates @ runtime** (feasible today)
  - Each system maintains its own local safety objective
  - But composed system may not be safe

- **Safety Cases @ runtime** (feasible tomorrow)
  - Component system safety cases guide adaptation
  - Integrated dynamically for safe & assured assembly
  - E.g., one system may need to demonstrate it delivers properties assumed by another

- **V&V @ runtime** (my goal, feasible soon)
  - May be that one system cannot deliver assumptions required by another
  - So adjustments needed
  - E.g., wrappers or monitors to exclude some class of faults

- **Hazard Analysis & Risk Assessm’t at RT** (infeasible today)

John Rushby, SRI
Example: SILF SKIP

SILF: Semantic Interoperability Logical Framework

- Developed by NATO to enable dependable machine-to-machine information exchanges among Command and Control Systems

- Extensive ontology to describe content of messages exchanged
  - So in SM@RT terms, ontological descriptions (e.g., in OWL) are the models

- Mediation mechanism to translate messages as needed
  - Synthesized at integration time

- Mediation can be performed by centralized hub, or by wrappers at either the sender or receiver
ONISTT and Onward

- **ONISTT** is an SRI project, **prototyped** ideas of SILF
  - Ad-hoc Prolog program synthesizes the mediator
    - Now uses F-Logic and Flora2
  - Synthesis procedure can also decide when incompatibilities too great to meet purpose of integration
  - Used successfully to integrate live and virtual simulation systems for military training
- **ONISTT** achieves restricted form of **safety cases @ runtime**
- More general applications likely require **richer models than ontologies**
  - E.g., state machines and formal specifications
- How to perform synthesis on these?
Some Heresies

- **Worst-case complexities don’t matter** much for applied formal methods
  - Everything is exponential or worse (nonelementary, undecidable)
- **What matters is typical** performance
- E.g., Propositional SAT is NP-Complete, presumably exponential
  - But routine for modern SAT solvers to solve problems with millions of variables and clauses in seconds
- Prefer not to use LTL etc., to specify sequencing
  - Desired properties are either trivial (invariants, bounded eventuality)
  - Or complex—in which case engineers find it hard to write correct LTL, PSL formulas
- Use (skeletons of) **synchronous observers** instead
Synthesis as Exists/Forall Problem

- At integration time, systems need to synthesize wrappers, monitors, shims etc.
- Synthesis can be seen as a generate and verify search problem
  - Construct a candidate program
  - Try to formally verify that it meets specification
  - If not, generate new candidate and iterate
- Unrestricted search will not work
- Have human provide template/sketch, synthesis fills in details
- Simple example of a template for an invariant $Ax + By = C$
- Formally, this can be expressed as

$$\exists A, B, C : \forall x, y : Ax + By = C$$

where $x$ and $y$ are program variables, and the parameters $A$, $B$, $C$ must be instantiated by the synthesis procedure

- Note two-level quantification: Exists/Forall (EF)
Synthesis as Exists/Forall Problem (ctd. 1)

- Variants on EF formulation can express
  - Invariant generation
  - Assumption synthesis
    - Find the weakest environment in which a given component meets its requirements
  - Supervisory controller synthesis
    - Design an algorithm to selectively disable component actions so that it satisfies some goal in the face of uncontrollable actions by the environment
  - Full synthesis
    - Design an algorithm to achieve some goal

- So how do we solve EF problems?

- Start by solving one-level problems: Exists or Forall
Synthesis as Exists/Forall Problem (ctd. 2)

• **Satisfiability Modulo Theories (SMT)**

• A breakthrough in automated theorem proving, 15 years ago

• Decides Boolean formulas over combination of theories

• ... Boolean formulas: e.g., \((x \leq y \vee y = 5) \land (x < 0 \vee y \leq x) \land x \neq y\)
  
  ... continued for many terms

• ... over combination of theories

\[ e.g., 2 \times \text{car}(x) - 3 \times \text{cdr}(x) = f(\text{cdr}(x)) \supset \]
\[ f(\text{cons}(4 \times \text{car}(x) - 2 \times f(\text{cdr}(x)), y)) = f(\text{cons}(6 \times \text{cdr}(x), y)) \]

Uses equality, uninterpreted functions, linear arithmetic, lists

• Can extend to one level of quantification
  
  (i.e., either Exists or Forall)

• There are many SMT solvers, honed by competition

• Routine to handle hundreds of thousands of terms in seconds
Synthesis as Exists/Forall Problem (ctd. 3) SKIP

- **EF-SMT** solver uses an ordinary SMT solver as a component
  1. **Guess** (cleverly) instantiations for the **Exists** variables and query the SMT solver with the resulting **Forall** formula
  2. If this succeeds, we are done
  3. If it fails, use the result (i.e., **counterexample**) of the **Forall** query to help in finding the next instantiation of the **Exists** variables

- Key in making this efficient is to use (i.e., learn from) the result of failed verification (Forall) steps to prune the search space for subsequent synthesis (Exists) steps

- Many SMT solvers being extended to EF solving (e.g., **Yices**)
Composition

- EF solvers can maybe synthesize monitors for local properties
- But we need global properties
- So need to compose local monitors (and maybe other algorithmic elements) to yield distributed runtime monitors
- Aha! The topic of this workshop
- Although most of this talk is from a paper “Trustworthy Self-Integrating Systems” in the 12th International Conference on Distributed Computing and Internet Technology (ICDCIT), Bhubaneswar, India, January 2016; published as Springer LNCS Vol 9581, pp. 19–29
Vision

- Systems come together
- Exchange models, assurance cases
- Under guidance of an integration app
  - Which expresses the purpose of the integration
    * E.g., as a template or sketch
- Connectors, wrappers, monitors, and shims are synthesized
  - By EF-SMT solver
- Global properties are ensured by composing these to yield distributed runtime monitors
- And system assurance case is composed from those of component systems and global monitors
- Delivers a trustworthy integration

John Rushby, SRI
Prospects

- **Trustworthy self integration** is within reach
  - For simple cases...

- **Need theorem proving at integration time**
  - To *synthesize* the connectors, monitors etc.
  - And to *build* the composed assurance case

- **So a theorem prover will be at the heart of self integration**

- In future, will likely also use *learning* to infer properties beyond supplied models

- Further ahead, will integrate **highly autonomous systems**
  - Numerous failures in HMI (e.g., Air France and Air Asia crashes) show *this is difficult*

- **So must exchange more strategic information** than SM@RT

- **Maybe beliefs, desires, intent (BDI), even a system of ethics**

- This is the **future of IoT**