HACMS Panel, LAW 2012, Orlando FL

HACMS Ground Team Integration

John Rushby

Computer Science Laboratory
SRI International
Menlo Park CA USA

Overview

- SRI performs in the systems and controls task areas

- And we are also the Ground Team Integrator
  - Assembling and integrating code from other performers and getting it onto the Landshark and Cadillac
  - Integrating the formal assurances supplied with the code

- I'm going to focus on the last of these

- And our technology for doing this
  - The Evidential Tool Bus (ETB)

Aside: Low-Level Formal Integrations

- Modern formal verification and synthesis are seldom performed with a single monolithic tool (cf. VCC)
  - Emerging market of interacting specialized components
  - Very rapid technology development
  - E.g., predicate abstractors, abstract interpreters, relational abstractors, invariant generators, (infinite) bounded model checkers, SMT solvers, language translators, typecheckers, VC generators
- Typical verifications weave several supported and ad hoc tools plus lots of glue code into a one-off workflow
- Almost impossible to replicate someone else's results
- But Airbus (say) needs to be able to trust the claims of these improvisations
- And to be able to revisit a modified design in 50 years time

Aside: High-Level Formal Integrations

- Separately verified or synthesized elements need to compose
- Some components discharge assumptions of others
  - Possibly mutually (that's assume/guarantee reasoning)
- Others have their own assumptions, models of the environment, etc. and we need overall coherence
- We use idea of an assurance case (cf. safety case) as the top-level organizing principle
  - Claims, Argument, Evidence
- What's the difference between an assurance case and a formal verification or synthesis?
  - Verification allays logic doubts
  - What remains are epistemic doubts

  Cf. V&V: verification and validation

Evidential Tool Bus: Purpose

The Evidential Tool Bus

- A way to assemble the claims made by different formally assured developments using different tools
  - And to compose them into an assurance case
- And a way to assemble the code they generate
- In a way that keeps everything consistent

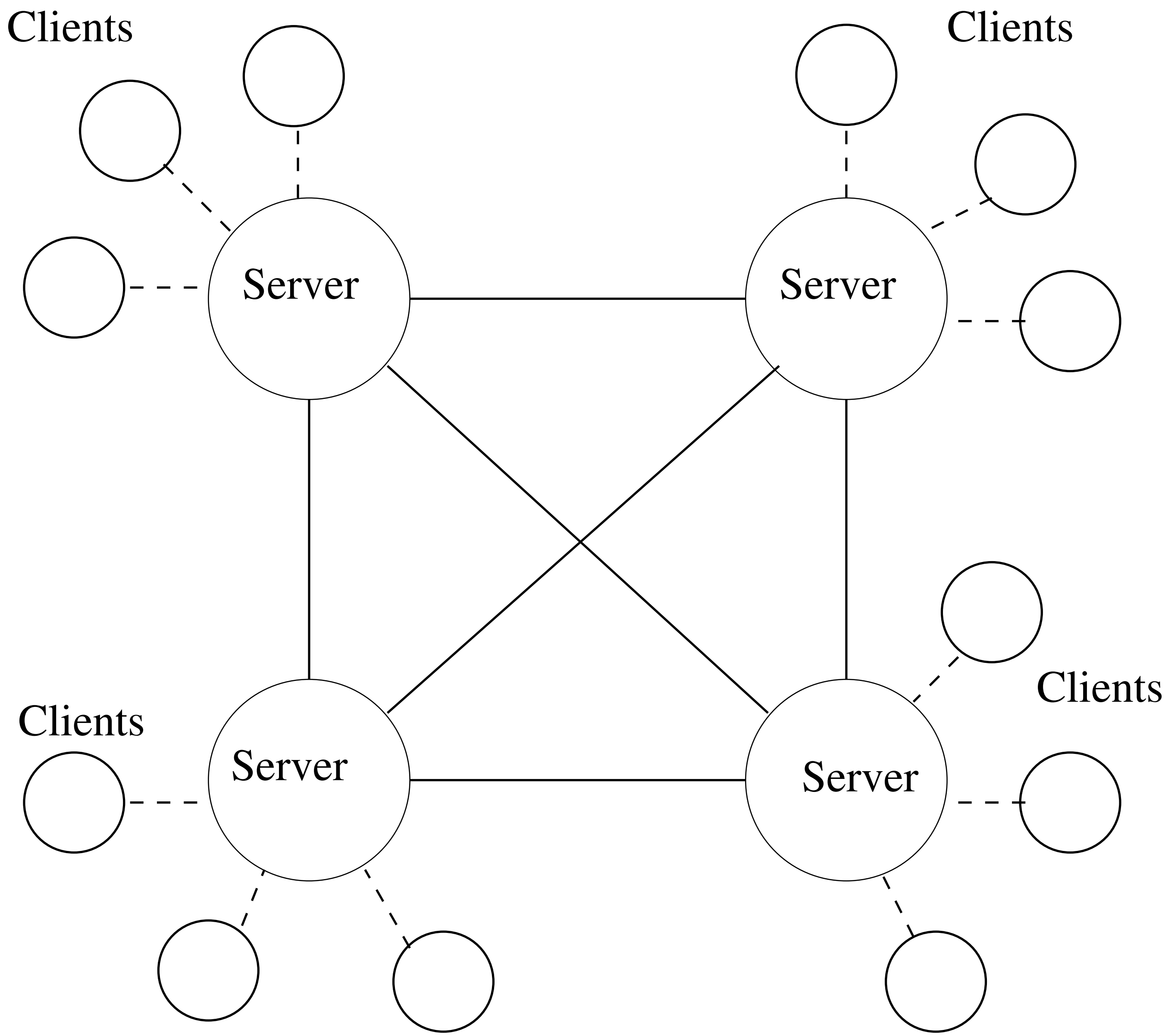
The Evidential Tool Bus

- A distributed, location-transparent way of invoking tools
  - A way for one tool to invoke services of another
  - And for scripting workflows
- And for accessing files, specs, etc.
- Cost of attaching tools to the ETB must be low
  - Lightweight wrappers
  - No mandated logic, format, methodology

ETB Architecture: Servers, Tools and Files

- ETB needs to be distributed
  - Some tools run only in specific places, on specific systems
  - Users are in many different places
- So the ETB is a fully connected graph of servers
  - Distributed on a subnet or via SSH tunnels
- Servers can come and go
- Servers can run various tools (and tool components)
  - Some servers may run no tools
  - Some may run many
  - Tools can run on one or more servers
  - Tools can be scripts
- Servers also store files

ETB: Picture


Clients
Clients
Server
Server
Clients
Clients
Server
Server

Architecture: Clients

- Humans interact with the ETB via clients
- Which connect to a server using an API (about 20 methods)
- Clients have no ETB state,
- Currently, we provide just a simple shell
- Hope that safety case managers such as ASCE will choose to use ETB as their back end (i.e., they become ETB clients)

Architecture: Mechanisms

- Each server runs a simple daemon (written in Python) that exchanges messages with the others
  - When something happens
  - Or periodic heartbeat
- Underlying protocols use XML-RPC
  - With data represented in JSON
- Files are stored in a GIT repository on each server
  - Hence, are global, but consistency is lazy (by need)
  - Referenced by name (relative to server directory) and SHA1 hash
  - Hence, unique

ETB Predicates, Claims, and Workflows

- The unit for computation and for claims is a predicate
  - Like a (remote) procedure call that also attests a claim
- An ETB predicate is of the form
  - `name(arg1, arg2, ..., argn)`

  Where the `arg`s are variables, or data
  - If this is issued by a client, then it's a query
  - If this is the output of a tool, then it's a claim
- Claims are recorded in the ETB claims table
  - Which is later analyzed to yield the assurance case
- The `name` can be interpreted or uninterpreted
  - interpreted predicates cause invocation of tools
  - uninterpreted predicates invoke workflows

Example Interpreted Predicates

- `YicesCheck(Fmla, SAT?)`
  - Where `Fmla` is an SMT formula (or file)
  - And `SAT?` is a variable

  This is a query (queries can also be ground)

- Can be evaluated by a server that has the Yices SMT solver
  - Will instantiate the variables
  - And yield a claim (attested ground predicate)
  - e.g. `YicesCheck(Fmla, "satisfiable")` where `satisfiable` is a literal that indicates `Fmla` is satisfiable

- Can then do `YicesShowModel(Fmla, MODEL?)` to obtain model

- Claims Table keeps detailed log of claims

Tools, Wrappers, Scripts

- Tools attach to the ETB via wrappers
  - Typically a dozen lines of Python
  - Export appropriate predicates for that tool
  - Possibly of various granularities
    - ⋆ e.g., specific proof vs. all proofs in a file
- A wrapper may include fairly complex scripting
  - Can issue queries, make claims (including "error claims")
  - Can establish sessions, run interactive tools and invoke external activity (e.g., "ask Sam to prove this")
- Later, may want to deconstruct tools into shared components
- Claims established by interpreted predicates provide attestation (e.g., "proved by PVS", "John says it's so")
- But are internally opaque (trust bottoms out here)
  - i.e., they do not provide an ETB-level proof
  - That's what uninterpreted predicates are for

Support Tools

- Some interpreted tools just check the format of a file
- Others do translations between formats/logics
- Not everything is a specification or a theorem
  - Also have counterexamples, sets of predicates (for predicate abstraction), interpolants, etc.
  - Anticipate evolution of a 2-dimensional ontology
    - ⋆ Kinds of things x logic/representation
- Some tools run a makefile, create code
  - Code goes in a file, just like other data
- At present, limited fault tolerance, load balancing, security, job management

Uninterpreted Predicates

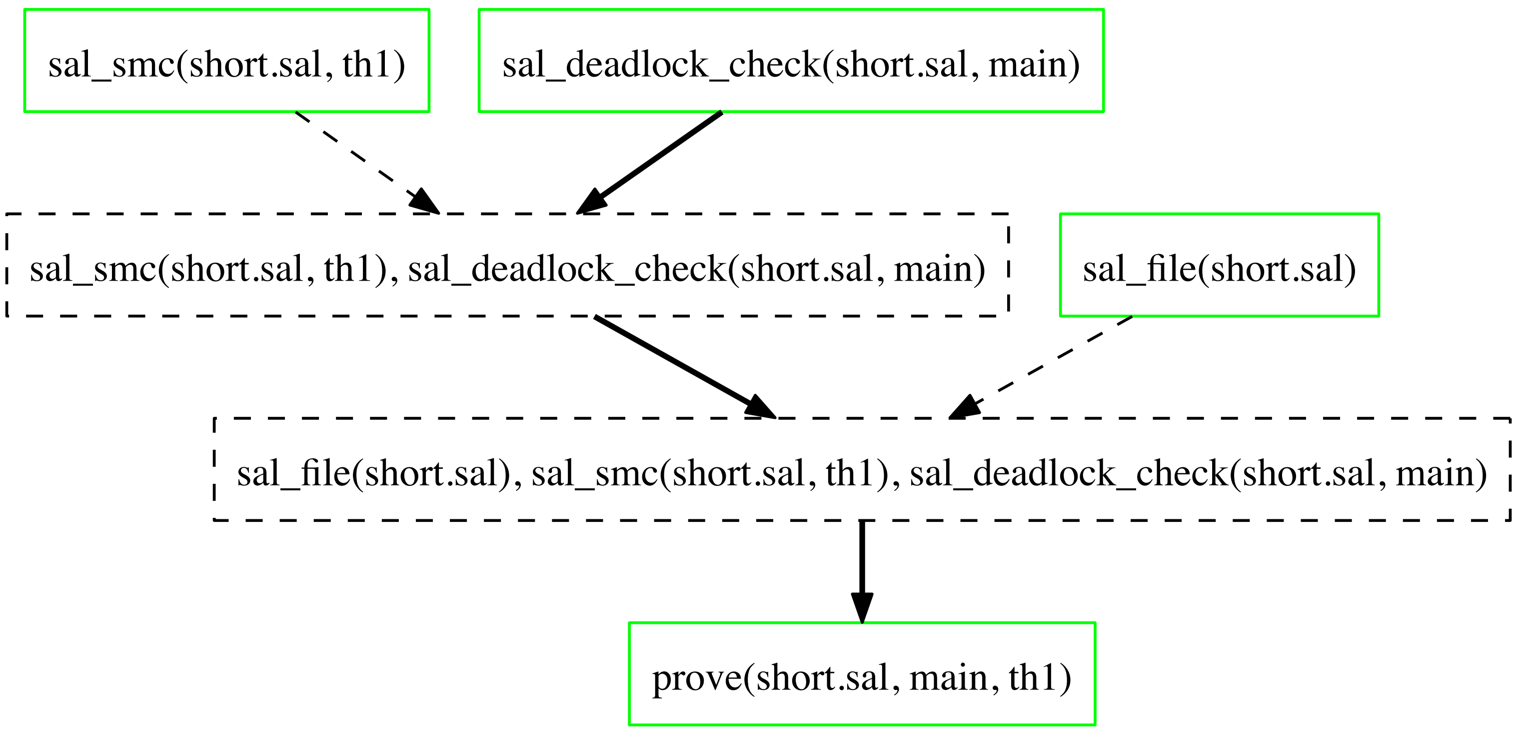
- ETB has a simple logic engine (inspired by Datalog)
- Uninterpreted predicates are defined by Horn-clause rules that are evaluated directly by the ETB: e.g.,

```
prove(F,M,P) :- sal_file(F),
                sal_smc(F,P),
                sal_deadlock_check(F,M).
```

- These define workflows
- Evaluation builds an ETB proof connecting claims
- Workflows can provide different proof modes
  - e.g., discovery vs. certification
  - First might call many SMT solvers, use first to complete
    - ⋆ There's an API query for tool completion
  - Second might call many, require all to give same answer
  - Or might call a trusted solver

ETB: Proof Tree

This is from the query `prove(short.sal, main, th1)` using the rule on the previous page



Conclusion

- ETB tries to address two urgent new problems
  - Linking tools and components together into flexible workflows
  - Tracking and assembling the interdependent claims of multiple tools working on part of the same problem
- Bridges the gap between formal verification/synthesis and assurance
- Creates the opportunity to formalize the upper levels of argument
  - Cf. Adelard FOG project
- And exposes the epistemic elements to scrutiny
- Workshop VeriSure: Verification and Assurance at CAV 2013, St Petersburg