HACMS kickoff meeting: TA4
Technical Area 4: Integration

John Rushby

Computer Science Laboratory
SRI International
Menlo Park, CA
Overview

• There are several parts to our effort under TA4

• Bob Bolles will cover vehicle integration in the breakout

• I’ll mention other parts at the end

• But here I’m going to focus on
  The Evidential Tool Bus (ETB)

• Because that is what we use to develop and assemble and deploy proofs and code in a distributed manner
  ○ We use it in the DARPA CASIO project
  ○ We and Honeywell use it on a NASA project
  ○ But the HACMS applications are much more ambitious
Integration Opportunities

- Assemble **code** from various developers, integrate it, get it on the vehicle, test it

- But this code is formally verified or synthesized
  - So need a chain of **provenance** from top-level claims

- And the **formal assurance** needs to be “assembled” also
  - Initially, stovepipes and a pile of disparate claims
  - Later, shared assumptions, mutual assumes/guarantees
  - Later still, fully compositional

- And the formal tools need to **interoperate**
  - Initially, stovepipes, but mutually accessible
  - Later, integrated workflows
  - Later still, modular tools built from components

- And may want distinct **development** and **certify** modes
Evidential Tool Bus: Purpose

The Evidential Tool Bus

- A way to assemble the **claims** made by 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 is **low**
  - Lightweight wrappers
  - No mandated logic, format, methodology
ETB Architecture: Servers, Tools and Files

- The ETB is a fully connected graph of servers.
- Servers are distributed:
  - On a subnet or via SSH tunnels.
- Servers can come and go.
- Servers can run various tools:
  - 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.
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

• You can also write your own (e.g., for Eclipse)

• We do have a Java-based project-specific graphical client for CASIO
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

• The unit for computation and for claims is a predicate
  ○ Like a (remote) function call that also attests a claim

• An ETB predicate is of the form
  ○ name(arg1, arg2, ..., argn)
  Where the args are variables, or data

• 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
  
  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 $\times$ logic/representation
- Some tools run a **makefile**, create code
  - Code goes in a file, just like other data
- Limited fault tolerance, load balancing, security, job management at present
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.,

\[
\text{prove}(F,M,P) :- \text{sal\_file}(F), \\
\hspace{1cm} \text{sal\_smc}(F,P), \\
\hspace{1cm} \text{sal\_deadlock\_check}(F,M).
\]

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

```
sal_smc(short.sal, th1)
sal_deadlock_check(short.sal, main)
sal_smc(short.sal, th1), sal_deadlock_check(short.sal, main)
sal_file(short.sal)
sal_file(short.sal), sal_smc(short.sal, th1), sal_deadlock_check(short.sal, main)
prove(short.sal, main, th1)
```
Plan

• Further develop and deploy the ETB
  ○ Gregoire Hamon

• With your input
  ○ This is our third attempt, also the simplest
  ○ Seek early adopters
  ○ Technical introductions by Webex, welcome visitors
Other Parts of TA4

- Trusted tools: **Kernel Of Truth** (KOT), Shankar
  - Tower of increasing powerful verifiers and synthesizers
  - Each formally verified using the ones below

- Compositional Verification: **Lazy Composition**, Shankar
  - Assume/Guarantee is sound but not credible for genuine components
  - Designed in ignorance, why would my guarantees match your assumes?
  - So synthesize weakest assumptions

- Top-Level: **Assurance Case**, John Rushby
  - Tradeoff *epistemic* and *logic* doubt