Mace: Language Support for Building Distributed Systems.

Killian et al. PLDI 2007

538B projects

- Proposals received! I'll get you comments later this week.
- *Please make sure to allocate time for your projects each weak*
- Next deliverable:
  - Project update and an email to schedule meetings with me by November 13th
  - "Project update**:** a maximum two page update on your project: what you have done so far, initial result, what remains to be done, unexpected challenges you have encountered, etc. This update will define the agenda for your project update meeting with Ivan."

Mace paper

- How is this 2007 *paper* different from the RPC/Argus/Emerald papers? (meta level)
- Writing: More diagrams, more take-aways.
- More result oriented. More high-level. Argues its case continuously throughout the paper. There is a value-add to every sentence in the paper.
- Eval: experiences instead of experiments. There are *users:* novices (students), experts (themselves). Care about lines of code/productivity (new measure) — good match to their "product".
- Compare head-to-head with previous work (non-Mace systems).
- Many systems in eval: 8 systems in Figure 7. Systems are "deployed"; not "toy" systems. Larger systems (4K+ LOC)
- A lot of work! 5 Authors!
- Paper as artifact (for systems papers). You can download the code!
- All of these are choices made. Pay attention to the choices.

Mace compiler

- Mace is a compiler.. but how is the Mace compiler different from the Emerald compiler?
    - Source to source compiler (Mace -> C++). Why not x86 output?
        - Partially the goal is system comprehension. Re-use existing tools. Easier to interpret output and integrate with other systems.
        - It's hard to build a new compiler! C++ compiler already exists (and it's not bad). Smart choice! Saves time. Faster phd. Faster publication. More papers :-)
        - Written in Perl (omg)
- Emerald language, output assembly. Designed for one objective: object mobility.
- Goal: bridge gap between high level spec and low-level implementation. Use abstractions. Correctness of the entire system (not really spec). Not restricted to any particular flavour of systems (or is it?).
    - Designed for event-based systems with explicit state machines/transitions. Need to follow this abstraction.
    - Concurrency/parallelism is poorly described.
    - "Self contained" — write everything in Mace. Harder to integrate with other systems.
    - *What happens at the boundary between Mace and non-Mace?*

Mace: DSL on top of C++

- *What's the problem* that it is trying to solve?
  - Help with debugging distributed systems; help with "experience" of writing such a system
  - Help link high-level and low-level spec
  - Help get high performance
  - Help with conciseness of implementation
- *Lots of general purpose PLs out there.. but none for distributed systems. Let's fix that!*
- *Ambitious!*
- *"Open DSL" : extends a PL rather than restricting. But, constraints the structure of the resulting code. Impose constraints on programs (to get above benefits).*
- *Paper as criticism of general PL: existing PLs are too wild and freedom giving. Developers need discipline/regularity/abstractions for this domain (distributed systems).*

Mace design ideas

- Mace contributes key concepts for this domain
- Layers ~ objects (encompass state): Relaxed encapsulation. Invoke a method to generate an event (async invocations). Message passing between layers.
- Layer/object interfaces: for constraining what can be invoked/generated by a layer.
- Events (trigger state transitions), correspond to methods and allow messaging between components. Upcalls, downfalls, timer events.
- Aspects : for cross-cutting concerns (concerns that span multiple objects). For correctness, for failure handling, logging,…
- Performance: Errors can percolate up the stack of layers to the “right place” where it can be handled best. Concurrency through async event handling at layers (events get queued up and processed concurrently).

Mace

- Impose explicit state definitions: control states and data states.
- Control states: high-level protocol states. Capture the goal of my system.
- Data states: lower-level implementation details that support the control states.
- What is distributed state here? Not reflected in the Mace spec.
- Events cause modification to state
- Explicit states: make logging easier, easier to reason about critical sections.
- (Best part for me) Enables various program analysis: model checker, deterministic replay, detection aspects (runtime verification, on event granularity), causal path construction (aspects enabled this), logging.
    - Model checking (bug finding view of MC): find bugs in a model. Exhaustively explore “states” in the model. Check visited state against a human given “spec”, and report failure when the state doesn’t match the spec.
    - Visit states by computing them: follow the transitions that are available in your system (transfer function).
    - Typically state is poorly defined in a general purpose PL (state relevant to the spec that you are checking). Control states are the things being explored in MaceMC (partly the reason for distinguished control states from data states).

Mace eval

- Eval: experiences instead of experiments. There are *users:* novices (students), experts (themselves). Care about lines of code/productivity (new measure) — good match to their "product".
    - "Asked" novice users = ugrad students. Ethical? Informed consent && ability to opt out are minimal ethical requirements. Highly realistic!? 4th year -> new employee at Google, not a stretch. Users studies are tricky.
    - Timing result: 12 hours to write! (Effort estimate). But.. by who?
- Compare head-to-head with previous work (non-Mace systems).
    - LOC ~ productivity? Doesn't measure design effort to re-express systems in their structure. It's concise! Fits on a page! Do they count lines before or after compilation? (Probably before). Would have been nice to know post-compilation: if you have a (compiler) bug.. you'll need to contend with the output.
    - Minimal performance evaluation (Fig 8,9; mostly Figure 9). Comparison against a Java-based system. Java v. C++ comparison doesn't help to validate the value of their ideas!
    - Fig 9 seems constant.. unclear why. Don't explain their results.
- Many systems in eval: 8 systems in Figure 7. Systems are "deployed"; not "toy" systems. Larger systems (4K+ LOC)
- Introduce many parts: model checker, aspects… so many things. None of them are evaluated! This creates an evaluation gap (between design and eval). This is poor practice!
- Ali: how is this even accepted? (Lots of work is apparent!)

MC note

- Model checking — we will discuss a model checking paper a week from today, *MODIST*. An entire paper on model checking distributed systems!

Next: Verdi

- Verdi is from PLDI 2015 (nice contrast to Mace)
- More PL, but this time with a verification focus
- Probably the most PLish paper we will read (even features Coq! Theorem prover)
- Think about the modelling of distributed systems: what makes a good/valid model for verification?