Uni–Tea: Towards a unified model of distributed “object”

Mark S. Miller
Virus–Safe Computing Project
HP Labs
Overview

• What we are & aren’t unifying. Why and how.
  • To build virtual worlds worth living in

• Elements of the unified model

• Generate special models by setting knobs
  • Support inescapable trade-offs
    • Example surprise: fault-tolerance vs. mischief-tolerance

• Open Issues & Conclusions
Models we wish to unify

• “Simple things must be simple, ...”
  - Pass-by-proxy
  - Pass-by-copy

• “… complex things should be possible” (—Alan Kay?)
  - Loose-coupled form of David Reed’s “Tea-Party”
    - Mobile, fault-tolerant, symmetrically replicated state & behavior
    - Unit of replication & virtual synchrony
  - Chip Morningstar’s classic “Unum”
    - Smart “proxies” with best-efforts replication of some state.
    - Basis for Electric Communities’ Habitat
  - E. Dean Tribble’s “Available Objects”
    - Adaptive division of state, mixed behavior
    - Smoothly degrades during partition
    - Beyond the scope of this talk
How we try to unify

- Special models are points in a space
- Particular choices, appropriate in different circumstances
- EC focused on security & performance
- Tea-Time focuses on fault-tolerant replication
- By compare/contrast, find generative elements
Simultaneous Requirements

- Mutually suspicious objects & machines
  - Encapsulation, integrity, authenticity, equality, naming
  - Extensibility via mobile code demands easy POLA
  - No globally relied-upon 3rd parties—no Verisign or ICANN

- Concurrency
  - Consistency, latency, liveness, performance, scaling
  - Simple model: e.g., partial causal order of local atomic events

- Fault tolerance
  - Correctness during & after crash-recovery & partition

- Ease of programming
  - Easy to spot & reason about vulnerabilities!
  - Hide complexity within externally simple building blocks
  - Construction by composition. Preserve “object-ness”
What’s not included

- Non-local atomicity/simultaneity
  - The cross-coupled Tea-Party
  - Argus Guardians
  - Masking faults transparently
    - Classic serializable distributed databases
    - Synchronized checkpoints & rollbacks

- Adaptive & Subjective Presences
  - e.g., The rest of Chip’s original Unum model

- Fancy Protocols & Hardware
  - Multi-Party Secure Distributed Computation
  - Resistance to Traffic Analysis (e.g., blinding, packet mixes)
  - Resistance to DDOS
  - Copy protection, Remote attestation
  - Zero-knowledge, Byzantine, Lamport time, Paxos
  - Off-line, non-interactive protocols (e.g., certificates)
Elements of Uni–Tea
Elements of Uni–Tea

Host
(Machine, Process, Vat)
Elements of Uni–Tea

Unum
(Tea Party,
Distributed “Object”)
Elements of Uni–Tea

Authoritative Presence (Authority)

Shadow Presence (Shadow)
Elements of Uni–Tea

- Replicated Open State
- Encapsulated State
Elements of Uni–Tea
Elements of Uni–Tea
Elements of Uni–Tea
Elements of Uni–Tea

Consensus protocol (e.g., TeaTime magic!)
Elements of Uni–Tea

Local ref
Elements of Uni–Tea

Local ref

Per–presence state
Host–Unum Duality

- All inter-Host **arrows** are intra-Unum channels...
  - ...to internal “coordination” facets of each presence
  - From “inside”, different presences act differently
  - Internal protocols must deal with network hazards
    - Burden of the Unum’s designer

- All inter-Unum **arrows** are intra-Host local refs...
  - ...to external “client” facet of the Unum’s local presence
  - From “outside”, Unum’s presences all implement same contract
  - Refs to client facet (eventually) unserializes to local presence
  - External protocol hides many network hazards
    - Simplicity for the Unum’s user
Preserve Reference
Asymmetry

• Authority–hosts mutually trusted to provide object
  • Replicate and coordinate for fault tolerance
    • Less vulnerable to unreliable (fail–stop) hosts
    • More vulnerable to dishonest or flaky hosts
  • Design coordination/consensus strategy
    • Single stationary authority
    • Primary + understudies. Leader election
    • Quorum voting, ...

• Shadow–hosts separately trusted to use object
  • All shadows believe any authority
    • Replicated state, update msgs, upstream authority list
    • Shadows believe only authorities

• Each local object relies on its own host anyway
  • and so can rely on the local shadows of Unums it uses.
Pass-by-proxy

Diagram showing state transitions and pass-by-proxy mechanism.
Pass-by-proxy

- One stationary authority (the “real” object)
  - Mutable encapsulated accurate state
  - Original host is leader for life—has sole right-to-provide
  - No distinction between “internal” and “external” facets
  - Shadows never updated

- Live shadows (proxies) delegate all to leader
  - Stateless, so no stale state
  - All behavior is remote, round-trip, and accurate
  - Service breaks (fail-stop) once leader is unreachable
  - SturdyRefs to recover connection
    - Client must manually re-establish consistency

- “Selfish” equality—canonical creation identity
  - Serialized form is just capability to authority
  - Proxies canonicalized on arrival
Pass-by-copy

Good state

Good state
Pass-by-copy

- All presences authoritative, no shadows
  - Immutable transparent replicated state
  - Behavior is local and accurate
  - No internal coordination, since none needed
    - No internal facets or protocol
  - Trivially robust during partition. Nothing to re-establish.

- “Selfless” equality—value-based
  - Serialized form encodes contents (state + behavior)
  - Two identical copies are truly indistinguishable
  - No need to canonicalize
Loose-coupled Tea-Party
Loose-coupled Tea-Party

• All presences authoritative, no shadows
  • Mutable, encapsulated(?) , possibly-stale state
  • Accuracy is whatever a quorum agree on
    • Service breaks (suspends?) on minority side of partition
  • Highly mobile, fault-tolerant
  • All users are providers. Risk of dishonest user majority

• Virtual Synchronous Time
  • Speculative behavior is local and tentative
  • Commit is remote, round-trip, and accurate

• “Selfish” equality—canonical creation identity
  • Serialized form could contain state and presence list
  • Presences canonicalized on arrival(?)
Classic Unum
Classic Unum

• One stationary authority splits state
  • All state accurate, by definition (primary copy)
    • Stationary mutable encapsulated state
    • Replicated staleness–tolerant transparent state
  • Original host is leader for life—has sole right-to-provide
  • Best-efforts notification of Shadows to update state

• Shadows split behavior
  • Staleness–tolerant queries are local, fast, and stale
    • Could continue while leader is unreachable
  • Remaining behavior is remote, slow, and accurate
    • Breaks or suspends while leader is unreachable

• “Selfish” equality—canonical creation identity
  • Serialized form is just capability to authority
  • Presences canonicalized on arrival (& instantiated if needed)
Primary Copy + Leader Election

Leader

Good state

Replication Election

Understudy

Stale state

Leader?

Stale state

Stale state

Leader?
Primary Copy + Leader Election

• All authorities split all state
  • Leader’s state accurate, by definition (primary copy)
    • Replicated staleness-tolerant mutable encapsulated state
    • Replicated staleness-tolerant transparent state
  • Best-efforts notification of others to update state
  • If understudy is elected, his stale-state is deemed accurate

• Non-leaders split behavior
  • Staleness-tolerant queries are local and stale
    • Could continue while no leader is reachable
  • Remaining behavior is remote, round-trip, and accurate
    • Breaks or suspends while no leader is reachable

• “Selfish” equality—canonical creation identity
  • Serialized form is list of authoritative-coordination-facets
  • Presences canonicalized on arrival (& instantiated if
Related Open Issues

- Apparent need to collect distributed cycles
  - Network weak pointers? (Seems hard)
  - True distributed GC (Bejar algorithm?)
  - Ping-pong between registration and update
    - (Currently used by the E Lamport-Slot)

- Need Unum Construction Toolkit
  - Automate the important internal coordination patterns
  - New linguistic abstractions might help (Be very afraid)

- No flag days—Version mismatch tolerance
  - Within a machine, or within an Unum? (It’s a difficult choice)

- Other patterns needed for real scaling
  - Spontaneous multicast trees for downstream messages
  - What happens to equality? Path-based?
Some Choices

• Could unify at a lower layer
  • As we are doing: Pass-by-{proxy, copy, construction}
  • All the rest are built from Pass-by-construction
    • (plus needed identity canonicalization logic)

• Could special-case for virtual worlds
  • Just make some choices
  • Sacrifice generality for simplicity—remember Chip’s lessons!

• Could design one unified generative model

Like: “Why make everything an object? Real programmers use scalars, records, functions, control-flow, …”

“After all, we support special models directly anyway.”

“After all, most people won’t see or use the generality.”
Why One Unified Model?

- Inside design choices mostly hidden from outside
  - What kind of object/Unum is it really? Why should you care?
  - Late-bound compositional programming

- Transference of learning, reuse of tools
  - Many lessons are true for any kind of object/Unum
  - Much code can handle any kind of object/Unum

- Eases invention of new/mixed special models
- Solve hard distribution, performance, security problems once