TeaTime: Designing the Architectural Framework for Croquet

David P. Reed

Hewlett-Packard Labs &

MIT Media Lab

October 19, 2005

[with David A. Smith, Andreas Raab, and Alan Kay of Viewpoints Research Institute]

Overview

Decentralized, real-time, p2p, shared reality, collaboration *framework*

Technology assumptions: low-latency, high-connectivity, abundant computation

Focus: coordination, extensibility, systems issues (i.e. security, robustness, evolvability)

Agenda

Motivation for new architecture

What is OO computing, exactly?

System Structure

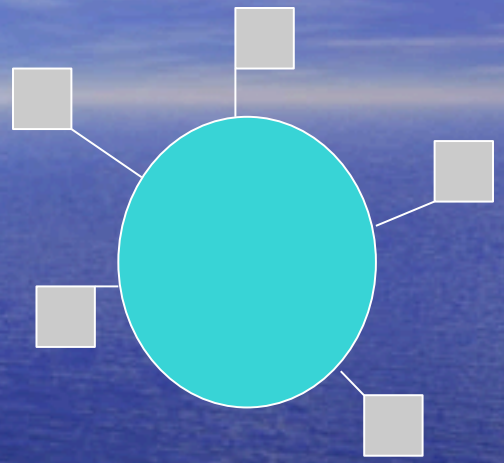
The TeaTime model of computation

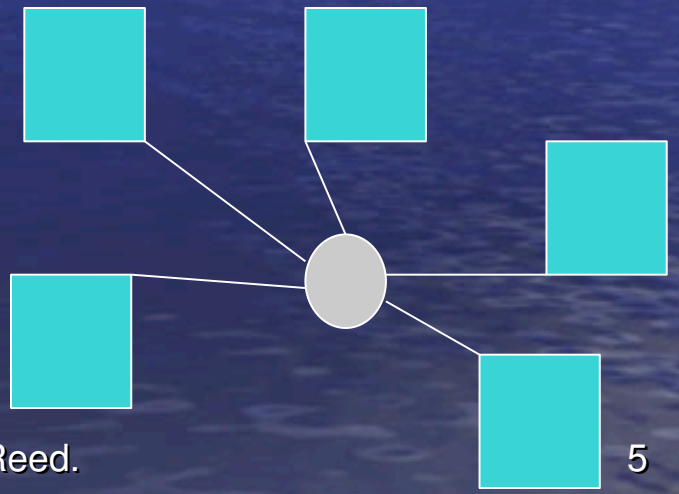
Messaging Infrastructure

Replication Infrastructure

Freely Scalable Platform

- Turn network inside out
- Capability grows with use
- Brilliant members replace
  - WIMP clients/LAME servers
- Resiliency and other systems issues become a user/application choice



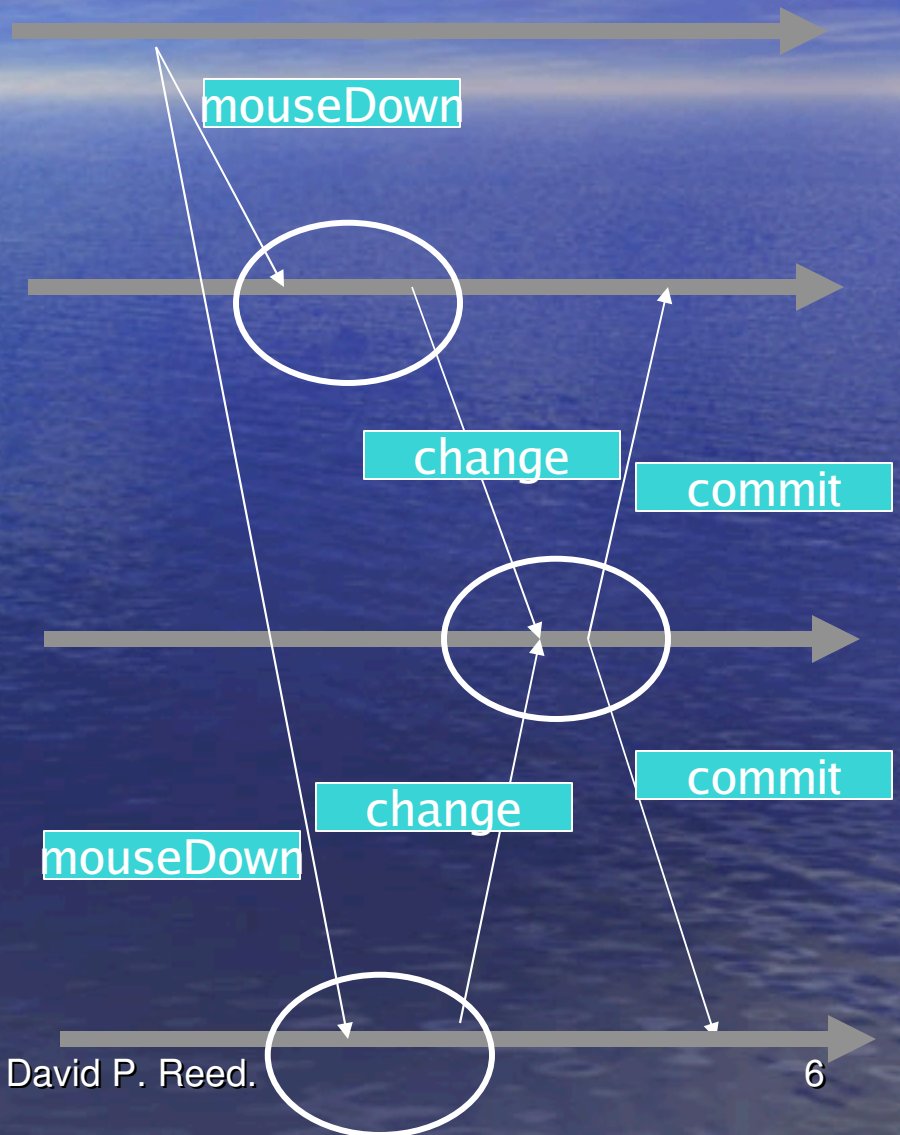


Objects Behave!

Time-centered computing

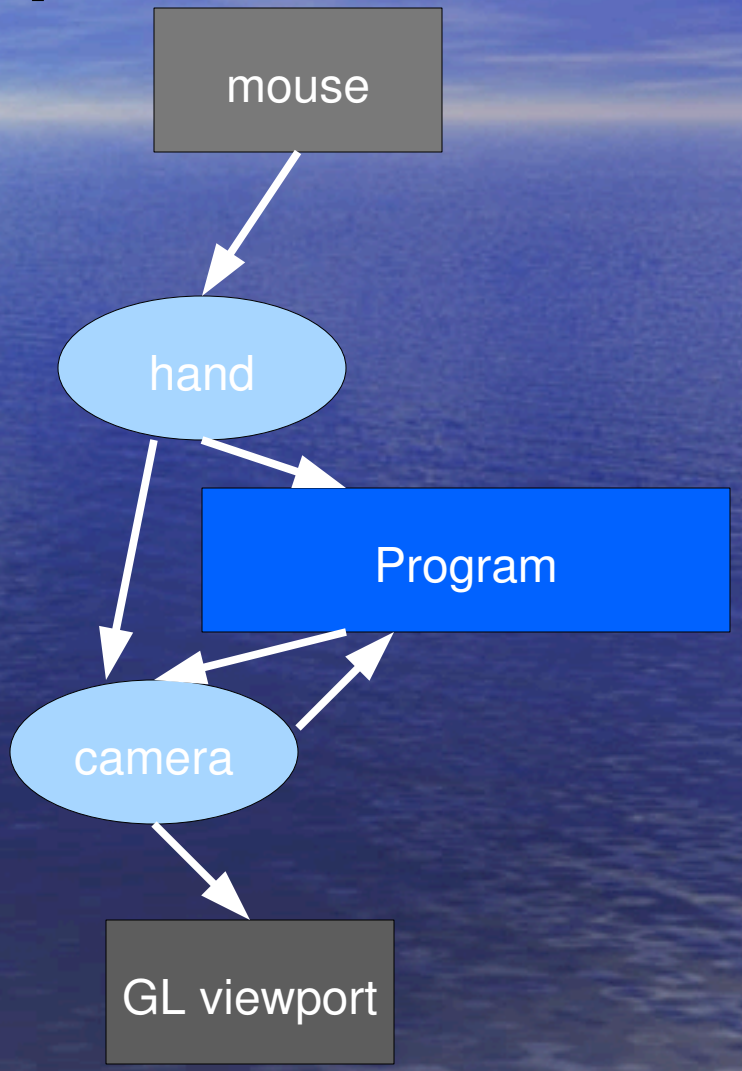
Replication and persistence of behaviors, not data

->Coordination via control of message scheduling, not control of access to data



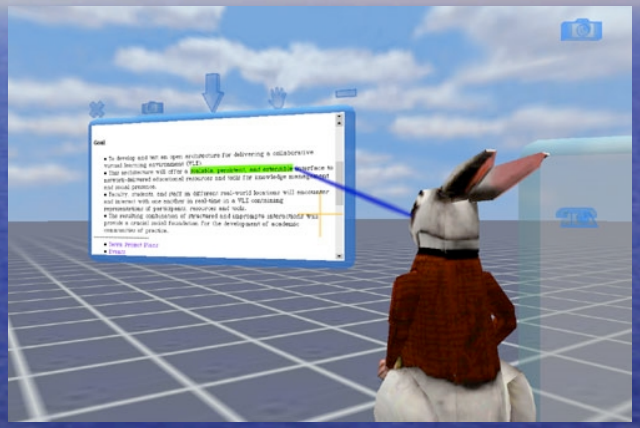
Embedded Virtuality

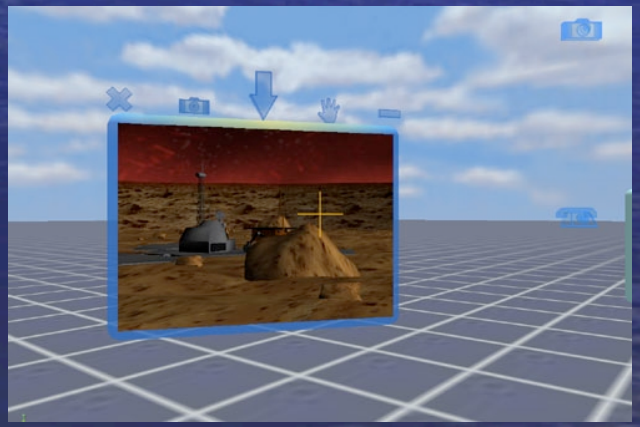
- The physical world is made up of *objects behaving*
- Common message distribution framework supports broad range of device ability (filtering/proxying)
- Temporal coordination across multiple devices in multiple places
- Controlled latency for natural media interaction



A powerful user framework

- Shared spaces hold behaviors
- Portals link spaces
- User and their agents are objects
- Visual “filters” represent “tools” that operate on objects
- Message-flow constrains security





Legacy software support

- Desktop or window rendered into a shared space accessible to multiple users
- Specialized computing devices or programs can be cloaked by Croquet object proxy

Croquet Abstraction and Implementation

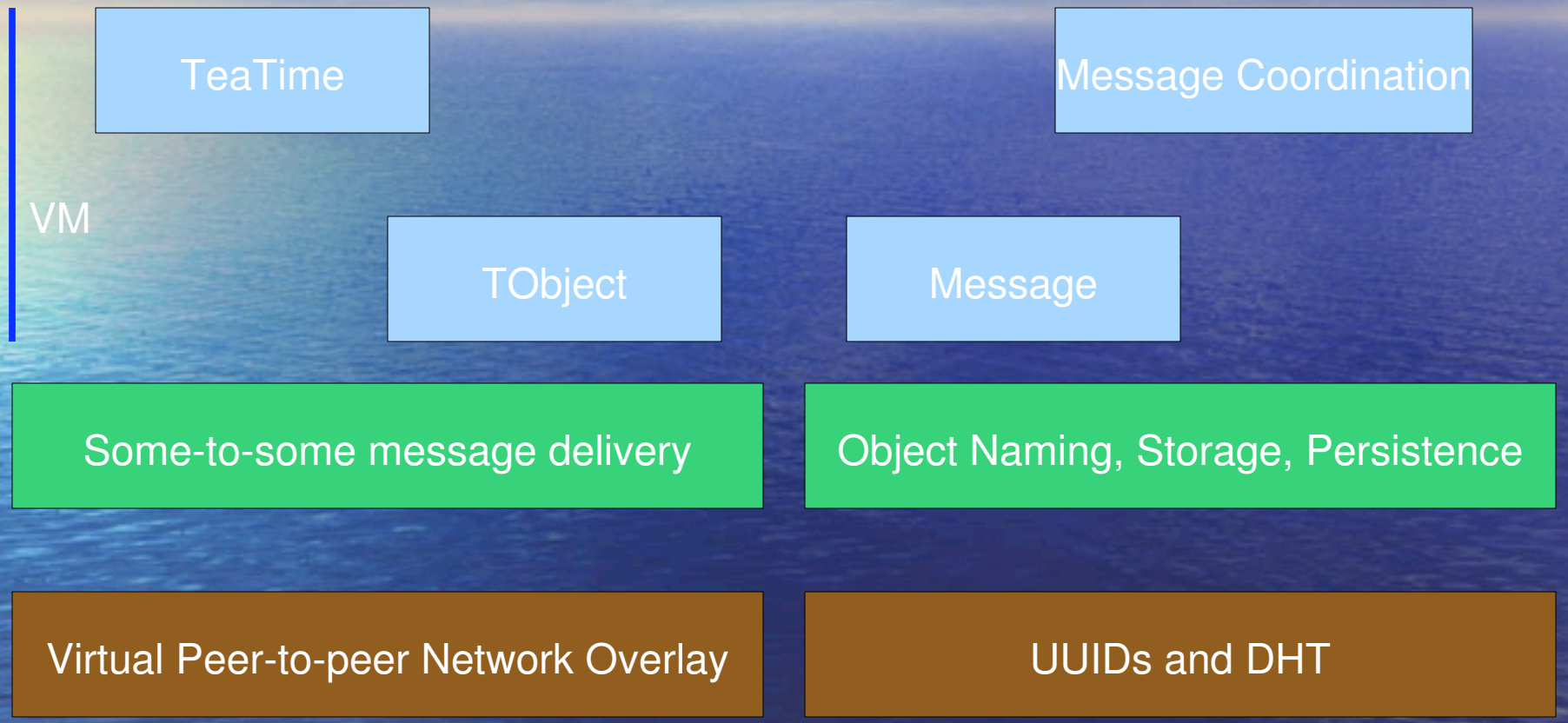
Croquet is not Squeak or Smalltalk

Implemented in Squeak

"Virtual machine" – not a language (yet?)

Eventually a family of languages?

Systems Structure



Computational model - TeaTime

TObjects – objects that behave

Croquet Messages – connect objects

Primitive TObject classes:

- TeaTime – expresses coordination
- TeaParty – manage replication
- TeaParticipant – virtual host machine

TObjects

- Objects express behavior over time by a stream of messages
- Object behavior controlled by a stream of messages
- Objects map received messages to sent messages via methods
- Objects can be thought of as an evolving history of messages sent
- Objects are causal – the output stream is a function of inputs

Croquet messages

- Messages select a method, carry parameter objects, and *specify "when" they take effect*
- Optionally, messages provide the name of an object to which a response is sent
- Messages belong to a TeaTime, and contain explicit timing constraints

Objects are uniquely named

An object is fundamentally a unique name, not a particular piece of storage, and has no inherent location.

Replication is done by allowing the same name to exist in multiple places - *clones*

Computational execution - sea of messages

Messages are issued by objects, but are not delivered immediately

Active messages in transit queues are prioritized by deadline – earliest deadline first

Example

Normal message send

obj oper: -expr-

obj perform: #oper: argument: -expr-

creates a message in the current TeaTime, result passed to next piece of code in block

"Future send"

obj future: 30.0 performAndCommit: #oper: argument: -expr-

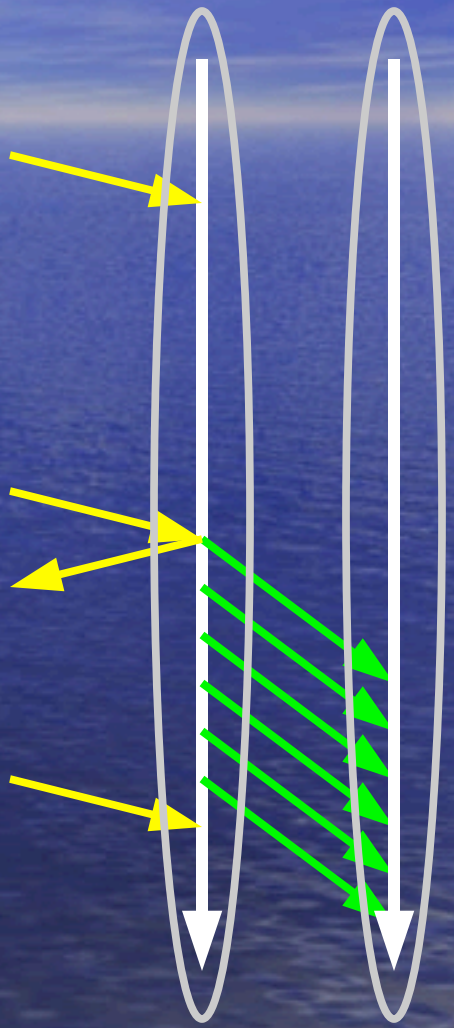
creates a new TeaTime 30 msec. in the future of the current TeaTime, and creates a message to obj in that TeaTime. result is the created TeaTime, passed to next piece of code in block.

TeaTime enables temporal reflection

- Objects expose a portion of their temporal history to read and modify, selected by TeaTime of the incoming message
- Until that history is “committed” it can be adjusted
- TeaTime is the control means by the object implements its own scheduling.

Key insight: objects are not states

- TeaTime is held in the messages
- Object is time-independent, programmed mapping from a set of input messages to a set of output messages
- "Code" of an object just defines the mapping
  - must be causal, incrementally executable



It's messages all the way down

[Unlike C++] primitives aren't read/write memory locations.

Objects are deterministic mappings from sets of messages [ordered by TeaTime] to sets of messages.

The input set of messages to an object evolves over time, and the output set evolves as a consequence of the input set.

Replicated computation

Croquet replicates computations

What does this mean when computations are composed of messages?

Each *clone* of an object has an identical history;

Messages between objects are replicated as needed so that each clone eventually receives an identical sequence of input messages

Tea Party – unit of replication

The granularity of replication is typically not the underlying object, and decision making naturally groups objects

Each object is a member of exactly one *tea party*; a tea party is a logical group of objects

Tea parties are the unit of replication – all objects in a tea party are cloned on the same set of machines

Example code

```
tp := TeaParty new. “new tea party”
space := (tp global: #TSpace) new. “new object in tea party”
tp future: 100.0 accept: { aTeaParticipant }. “replicate tea party”
tt := space future: 200.0 performAndCommit: #start: argument:
    self. “send msg to object”
tt committed ifTrue: [ … ].
```

… in TSpace …

```
start: sender
    sender future: 10.0 performAndCommit: #done. “reply”
```

Non-replicated tea parties

Objects that are specific to a particular machine are members of *non-replicated* tea parties

- the "home space" where the user enters
- I/O devices
- legacy applications on specific machines

Model with a tea party that cannot have clones

The everywhere replicated tea party

Certain "immutable" objects can be logically replicated everywhere

e.g. large read-only files that *never change once created* (textures, movies), true random number generators

What does *immutable* mean?

- Every message commutes with every other message – complete independence.

Enables special-case mechanism to manage clones

Replication of messages

Message replication is controlled by the replication of tea parties

Special case: from a tea party to itself

Other cases: one-to-many, many-to-many, many-to-one

Determinism and replication

Any tea party objects that are fully deterministic (across its boundaries) can be replicated arbitrarily.

Factor non-deterministic behaviors into separate non-replicated tea parties.

Applications easily organized by TeaParty concept

3d sound-based collaboration

Distributed computation – e.g. distributed collision detection (factor loosely connected computations into separate groups)

Persistence

Key issue is behavioral persistence

"servers" hold persistent state

(key idea is that you can "rent a participant")

Tea time coordinates all state

TeaTime objects

Fundamental unit of coordination

State diagram monotonic in time

- undecided -> committed
- undecided -> aborted

TeaTimes are totally ordered

All messages and histories are sequences of tea times

TeaTime and 3-phase commit

A causal, deterministic set of messages executes in an instant of tea time

While the tea time is undecided, it can affect the future of the objects it touches

If committed, it can only *observe* the effects of prior tea times

If aborted, it can have no effect and cannot be observed in computations carried out during other tea times.

Some-to-some messaging

Network layer implication of replicated message-based computation.

Each clone of a target object must receive one copy of the message from a source

Any source can provide for each target

(unicast, multicast, anycast are instances of some-to-some)

Macro structure

Croquet Places – “web page” or 3D space consist of multiple tea parties

Gateways – a small tea party that provides an advertised “entry point” to space

Rendezvous involves finding a gateway – the protocol begins by authenticating credentials

Security is “capability-based” – rendezvous is the bootstrap for security.

Network protocols

Gateway discovery (LAN broadcast and DHT)

Inter-tea party some-to-some peer protocol – under active research by DPR.

Tea-time commit (derived from my thesis, and reminscent of Paxos from Lamport)

Micro-structure

Use Squeak VM optimizations (including JIT someday).

"Veil"m echanism wraps all references between object in one tea party and another, so messages are sent through a "cached" path in the veil.

Optimizations factored into a "TeaPartyLink" structure