Down the Rabbit Hole

Spelunking the internals of Croquet
Part the First: Core Concepts and Classes

John Dougan

Hedgehog is Interim

* Unfinished Toolkit/API
  * Not an application
* Has some flashy demos
* Makes it look like more than it is

Key Components

* TeaTime, the distributed actions system
  * TIsland
  * future
  * Message Router
  * TFarRef
  * TController

Key Components

* TObject
  * TFrame/TSpace scene graphs
  * Avatars
* OpenGL interface
  * ST syntax extensions
* Tweak

Ignored

- There is a lot I'm not covering here
  - Scene graph and rendering: Camera, Lights, Robots, Texturing, 3D Math.
  - Harness and other support classes, etc.
  - Communications details
  - Political and organizational issues

Ignored

- Some things are changing, others will just confuse you initially.
- I want you to get the core concepts and be able to find the relevant bits in the image.
  - I can't teach 3D graphics creation, OpenGL, etc. Too much.

Style Note

* In general "T" is the standard class prefix used by the original Croquet devs. The "K" prefix is used for the Wisconsin/KAT classes.

TeaTime

* The system for distributing computations
* Current version is simplified and interim
* Most important component of Croquet.
  * The 3D system, while flashy and getting all the attention, isn't really that significant.

TeaTime

- There is a problem with Croquet terminology, several of the words have overloaded meanings.
  - TeaTime, Replica, Island...
  - Stop me if you're confused.

TeaTime - Island

- An Island is a bundle of state and computation that is being run identically on different computers
  - A Process has an Island
  - There is a (per-image) default island if no other is assigned.

TeaTime - Island

* Assertion: If you have 2 computations with the same inputs, they should have the same outputs.
* Even on different computers

TeaTime - Island

- How do you manage this?
  - Implement your platform to remove platform irregularities (floating point)
  - Never call on external inputs from inside an island computation (time clocks, random number generators)

TeaTime - Island

* How do you manage this?
  * All messages sent into the island computation are done through a special interface that ensures all island replicas get the same message at the same point in the computation.

TeaTime

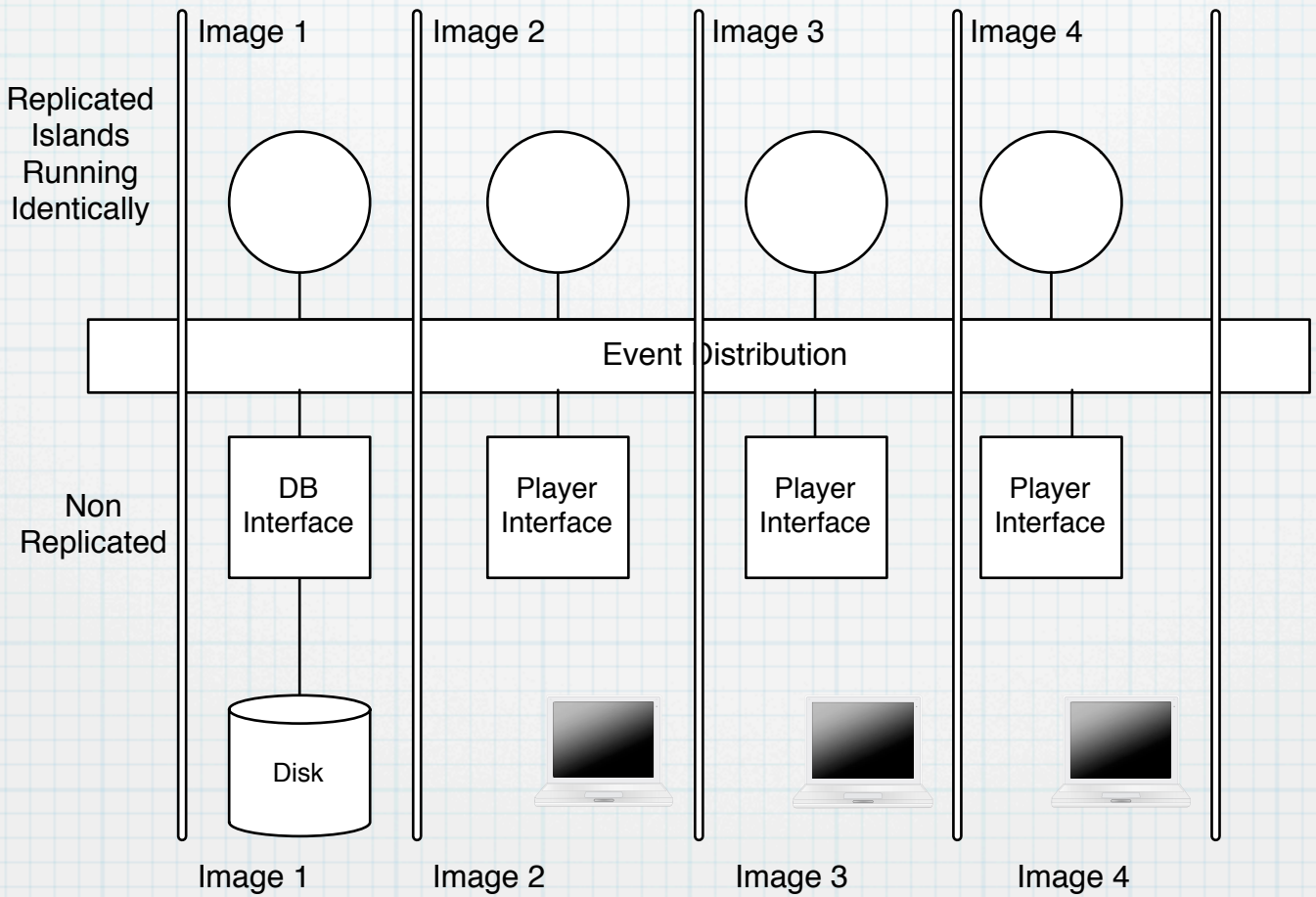


Image 1
Image 2
Image 3
Image 4
Replicated Islands Running Identically
Event Distribution
Non Replicated
DB Interface
Player Interface
Player Interface
Player Interface
Disk
Image 1
Image 2
Image 3
Image 4

TeaTime - TIsland

- TIsland
  - representative class for an island in Hedgehog
  - Has a registry that can be used inside and outside the island to access objects

TeaTime - FarRefs

- You aren't allowed to touch an object inside a different island directly.
- TFarRef
  - A Proxy into an object in a different island than yours.

TeaTime - Future

* Future - sending a message into an island
  * aTFarRef future doSomething: arg.
* Compiler translates to
  * aTFarRef futureDo: #doSomething: arguments: { arg }.

TeaTime - Distribution

- We have sent a message into the island and now need to coordinate the ordering of message sends into an island across all hosts running peer copies.
  - The islands share a loosely synchronized pseudo-time called TeaTime (in ms).
  - A distributed message runs at the same TeaTime on all replicated islands

TeaTime - Router

* In the interim Hedgehog TeaTime, there is a central message router that distributes messages to the replicated islands
* It assigns an execution TeaTime to each message
* It also sends heartbeat update ticks (Master Clock)

TeaTime - Controller

* Once the message is distributed by the router each island queues it
* The queue is handled by a TController. It monitors the current TeaTime and releases the queued messages that have that timestamp into the local island replicas

TeaTime-In Island Future

- You can send to the future inside an island but it requires indicating the delta into the future:
  - (islandObj future: 20) doSomething: (currVal + 1)
  - As this is already shared, it bypasses the router and adds directly to the controller
  - Useful for animation and timed loops

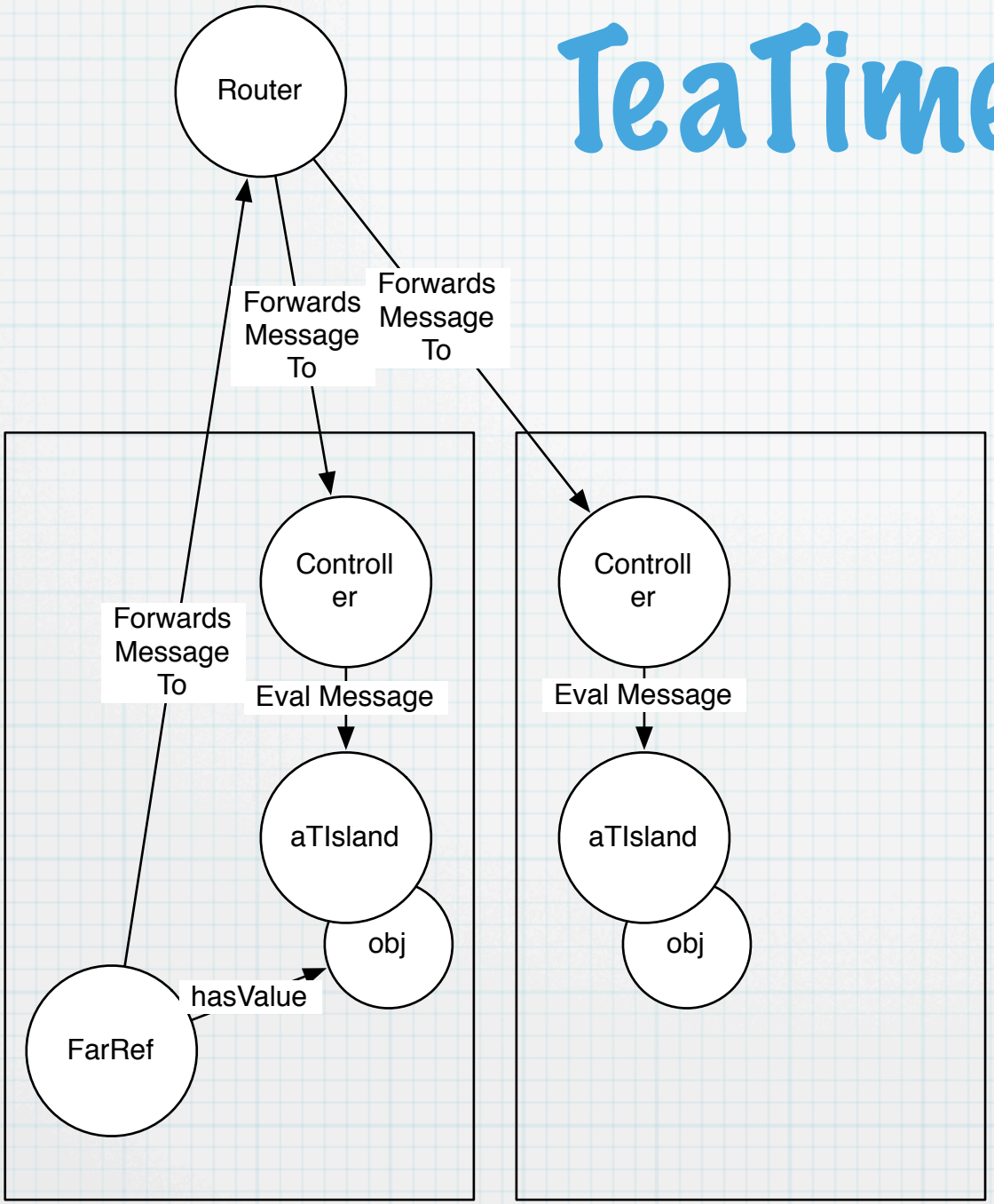
TeaTime - Signals

- There is an asynchronous (sorta) signal mechanism inherited from Tweak
  - This is used to push information out of the island
  - Careful selection of the signaling receiver can result in fairly efficient communication

TeaTime - Signals

- Registration, invocation, result.
  - externalRcvr runScript: #holdNotice: when: { islandObj . #holdingThing }
  - islandObj signal: #holdingThing with: IslandThing
  - externalRcvr holdNotice: thingFarRef

TeaTime



[*farRef* future doSomething]

Double bordered box encloses computations done in parallel across all island copies. *Replica* refers to the TAvatarReplica. *User* to the TAvatarUser. *Game* to a game replica for the game control to use. *Dbi* to the game DatabaseInterface

User hits an arrow key

user keyStroke: *event*
replicaFarRef future requestMovePlayerBy: (*dx@dy*)

the [*replica* requestMovePlayerBy: (*dx@dy*)] is distributed to all of the island copies

replica requestMovePlayerBy: (*dx@dy*)
game requestMovePlayer: *replica* by: (*dx@dy*)
game signal: #requestMovePlayer with: *game* with: *replica* with: (*dx@dy*)

User Image

Nothing.
No signal handler for {game. #requestMovePlayer}

DB Image

Catches signal {*game*. #requestMovePlayer}
dbi game: *gameFarRef* requestMovePlayer: *replicaFarRef* by: (*x@y*).
<Database actions here>
replicaFarRef future immMovePlayerTo: (*x@y*)

the [*replica* immMovePlayerTo: (*x@y*)] is distributed to all of the island replicas

replica immMovePlayerTo: (*x@y*)
(*replica* future: 25) translate: (*x@y@z*) "Moves player mesh"
replica signal: #playerMoved with: *replica* with: (*x@y*)

User Image

Catches signal {*replica*, #playerMoved}
user player: *replicaFarRef* movedTo: (*x@y*)
<etc.>

DB Image

Nothing.
No signal handler for {*replica*. #playerMoved}

TeaTime

Double bordered box encloses computations done in parallel across all island copies.
Replica refers to the TAvatarReplica. *User* to the TAvatarUser. *Game* to a game replica for the game control to use. *Dbi* to the game DatabaseInterface

User hits an arrow key

user keyStroke: *event*
replicaFarRef future requestMovePlayerBy: (*dx@dy*)

TeaTime

the [*replica* requestMovePlayerBy: (*dx@dy*)] is distributed to all of the island copies

replica requestMovePlayerBy: (*dx@dy*)
game requestMovePlayer: *replica* by: (*dx@dy*)
game signal: #requestMovePlayer with: *game* with: *replica* with: (*dx@dy*)

User Image

Nothing.
No signal handler for {game. #requestMovePlayer}

DB Image

Catches signal {*game*. #requestMovePlayer}
dbi game: *gameFarRef* requestMovePlayer: *replicaFarRef* by: (*x@y*).
    <Database actions here>
replicaFarRef future immMovePlayerTo: (*x@y*)

the [*replica* immMovePlayerTo: (*x@y*)] is distributed to all of the island replicas

User Image

Nothing.
No signal handler for {game. #requestMovePlayer}

DB Image

Catches signal {*game*. #requestMovePlayer}
dbi game: *gameFarRef* requestMovePlayer: *replicaFarRef* by: (*x*@*y*).
    <Database actions here>
replicaFarRef future immMovePlayerTo: (*x*@*y*)

the [*replica* immMovePlayerTo: (*x*@*y*)] is distributed to all of the island replicas

replica immMovePlayerTo: (*x*@*y*)
    (*replica* future: 25) translate: (*x*@*y*@*z*) "Moves player mesh"
replica signal: #playerMoved with: *replica* with: (*x*@*y*)

User Image

Catches signal {*replica*, #playerMoved}
user player: *replicaFarRef* movedTo: (*x*@*y*)
<etc.>

DB Image

Nothing.
No signal handler for {*replica*.
#playerMoved}

TeaTime

TObject

- TObject is the root class for many of the Croquet classes
  - Added is some support for signals/events and properties which are heavily used in Croquet

TObject

- TFrame - root class of the scene graph classes
- TSpace - root object for a scene graph
- TAvatarUser, TAvatarReplica - the two halves of an avatar

TFrame/TSpace

* A scene graph in Hedgehog is a tree of TFrames rooted in an instance of TSpace.
* There can be multiple scene graphs per island
* Each TFrame has a position and rotation relative to it's parent TFrame (4x4 Matrix)

Avatar

- 2 primary classes
- TAvatarUser
  - External to shared island, handles avatar controls, talks to avatar replica.
- TAvatarReplica
  - Shared TFrame object, lives in the shared TIsland

OpenGL

- Hedgehog is spec'ed to requite OpenGL 1.3 or later.
- OpenGL is used almost raw.
  - OpenGL commands are written to an OpenGL context during a recursive walk of the TFrame scene graph

OpenGL

- New Syntax introduced:
  - rcvr methodname(arg1, arg2, arg3).
  - Reads much more naturally with C API's
  - Oddly enough, they don't use it!
  - Not completely integrated with dev tools

Tweak

- Why Tweak?
  - Tweak is a new UI system for Squeak
  - Tries to have the flexibility of Morphic and the structure of MVC
- Origin of the Island concept
  - But they are different from Croquet TIslands.

Tweak

- Tweak has a number of changes to ST semantics:
  - event model
  - property assignments have the semantics of using an accessor
- In the Croquet demos where Tweak is used, Tweak lives in it's own local island and you have to use signals to talk to it.

Next Steps?

* Examine OpenGL and 3D Graphics.
* Starting at CroquetParticipant, examine the startup sequence.
* Examine the details of the message communications.
* http://croquetconsortium.org// - Tutorials, Docs, Mailing Lists.