www.projectdarkstar.com

Writing Darkstar Apps

Mark Rizzo

VP Platform Engineering

Perpetual Entertainment, Inc.

Chris Melissinos

Chief Gaming Officer

Sun Microsystems

Jeffrey Kesselman

Chief Darkstar Architect

Sun Microsystems

Perpetual Entertainment, Inc.

- Building games:
  - MMORPG Gods and Heroes – Late summer 2007
  - Star Trek Online – Late 2008
- Building platform technology
  - Billing, CS tools, community, thick game to thin client interface
  - Enables and enhances connected gaming
  - Enterprise Java framework, SOA architecture

Perpetual and Darkstar

- Huge demand in online game development community to lower cost of game development
- Many new markets opening that Perpetual wants to tap:
  - Light MMORPGs, casual multi-player, free to pay (digital object commerce), youth online
- Current heavy MMO development process too expensive and difficult to iterate
  - Make 15 $1M games vs. 1 $15M game

Perpetual and Darkstar

- Ease of integration between Perpetual platform and Darkstar
  - Java all around
- Rapid prototype on production scalable framework
- Support for thick C++ and Java thin clients
  - Gaming anywhere
- Open source model

JavaOne

Darkstar Applications

Business and technology

The hows and whys of writing applications for the Sun Game Server (SGS)

What will be covered

- The Business of Darkstar (10 min)
- Introduction to the SGS
- The SGS coding model *in brief*
- Real game implementation
- Where to go for more Information
- Q&A

What will be covered

- The Business of Darkstar (10 min)
  - *Chris Melissinos, Sun Chief Gaming Offficer*
- Introduction to the SGS
- The SGS coding model *in brief*
- Real game implementation
- Where to go for more Information
- Q&A

Why Project Darkstar?

- *Personal interest*
- *Sun Microsystems experience in online*
- *New approach to “old” problem*
- *Interest in growing the online game market*

New Gameplay Opportunities

- *MMOs today are not "massive" - but the potential audience is*
- *Allows players to engage in content from a variety of locations – "Live Anywhere" is a Sun concept, not Microsoft's.*
- *Touch the player on mobile, set-top, PC*
- *Levels the playing field for all developers*
- *Explosion of niche content possible through Darkstar*

New Business Opportunities

- *OpenSource **can** be free as in "free puppy"*
- *Services – Sun is best positioned to support*
- *We make systems and solutions – Darkstar brand servers*
- *Online Services – the cable TV model for online games*
- *Commercial licensing - innovate and expand, but you own it*

What will be covered

- The Business of Darkstar (10 min)
- Introduction to the SGS
  - *Jeff Kesselman, Chief Darkstar Architect*
  - Goals and Purpose of the SGS
  - Architecture of the SGS (very high level)
- The SGS coding model *in brief*
- Real Game Implementation
- Where to go for more Information
- Q&A

Purpose of the SGS

Problem it is intended to solve

- Make practical massively scalable,5-9s on-line game content in host-able model
  - Enable better games with more, smaller developers
- Current state of the massively multi-player on-line game industry
  - $30M base budget for a massively multi-player on-line game (MMO)
  - Scale only by dividing users, primitive world-space based load balancing (“zones” and “shards”)
  - Limited persistence and no fault-tolerance
  - Each game is a one-off

Source: Please add the source of your data here

Purpose of the SGS

Design Goals

- Make distributed persistent,fault-tolerant game servers easy and economical to write and administer
  - For the Developer
    - Make server-side game code reliable, scalable, persistent, and fault-tolerant in a transparent manner.
    - Present a simple single-threaded event-driven programming model to the developer. The developer should never have his or her code fail due to interactions between code handling different events.
  - For the Operator
    - Single point of administration
    - Load balance across entire data center

Source: Please add the source of your data here

Architecture of the SGS

Simplified and very brief

- In *design* much like a 3-tier enterprise system
  - Tier 1: Communications Layer
    - Publish/subscribe channels and direct client/server packets
    - Analgous to the “edge tier”
  - Tier 2: Execution Kernel
    - Executes “tasks” in response to “events”
    - Analagous to a J2EE app server
  - Tier 3: Object Store
    - Lightening fast, highly scalabe access to persistant objects
    - Abalgous to the DB tier.

Source: Please add the source of your data here

Architecture of the SGS

Simplified and very brief

- In *execution* very different
  - Tier 1: Communication
    - Reliable/unreliable ordered/unordered byte packet transport
    - Pluggable transports
    - **Optimized for lowest worst case latency**
  - Tier 2: Execution
    - Persistence of objects is ubiquitous and transparent (mostly)
    - Tasks are apparently mono-threaded
    - Objects are very simple, mostly normal J2SE
    - Stateless
    - **Optimized for lowest worst case latency**

Source: Please add the source of your data here

Architecture of the SGS

Simplified and very brief

- In *execution* very different
  - Tier 3: Object Store
    - All application state lives here
    - Custom in-memory data-store w/ secondary storage backup
    - Transactional and fault-tolerant but **not relational**
    - Deadlock detection for tier 2
    - Built to scale massively
    - **Optimized for lowest worst case latencies**

Source: Please add the source of your data here

What will be covered

- The Business of Darkstar (10 min)
- Introduction to the SGS
- The SGS coding model *in brief*
  - Events and Tasks
  - ManagedObjects and ManagedReferences
  - AppContext and Managers
  - Communication
- Real game implementation
- Where to go for more Information
- Q&A

Events and Tasks

Events

- Events are occurrences to which application code responds.
- There are two kinds of events
  - System events
    - Generated by the SGS infrastructure
  - Manager events
    - Generated by SGS managers
- Events result in a Task being queued for execution

Source: Please add the source of your data here

Events and Tasks

Tasks

- A task is a thread of execution
- Task execution *appears* to be mono-threaded
  - Task is transactional (ACID properties)
    - Appears to all happen at once to rest of system
  - Tasks take read and write locks on ManagedObjects
    - Locks freed at end of task.
  - Tasks abort and reschedule if a conflict arises.
- Tasks scale out horizontally over the back-end
  - Not a detail you need to think about
  - Just remember: fewer object access conflicts == greater scalability

Source: Please add the source of your data here

Events and Tasks

Task Ordering

- Task execution is mostly unordered and parallel
- *However* relative task ordering for a single user's input is assured.
  - Actions get executed in the order they arrive at the server.
  - An event generated by a user will not start processing until all processing of earlier events have successfully completed
- ***And*** parent-child task ordering is assured.
  - A task may use the TaskManager to queue child tasks.
  - A child task will not start processing until its parent task has successfully completed.

Source: Please add the source of your data here.

Events and Tasks

Event listener interfaces

- All events have a listener interface associated with them.
- ManagedObjects that wish to handle the event must implmement the appropriate interface.
- When a task for an event starts processing, it looks up and calls the handler for that event.

Source: Please add the source of your data here

ManagedObjects

Persistant SGS objects

- An SGS app is made of ManagedObjects
- MangedObjects..
  - Live in the object store
  - Are fetched by events
  - Are written back at successful termination of event
  - Are apparently mono-threaded in execution
  - Are referenced through ManagedReferences
  - Are normal Java objects that
    - Are Serializable
    - Implement the ManagedObject marker interface

Source: Please add the source of your data here

ManagedReference

References to ManagedObjects

- All SGS ManagedObjects must store references to other ManagedObjects in ManagedReferences.
- MangedReferences
  - Are Java reference types (like WeakReference etc)
    - Have the usual get() method
    - Also have getForUpdate()
  - Mark the persistance boundaries between ManagedObjects

Source: Please add the source of your data here

Example of ManagedObject fields

```
public class Foo implements Serializable, ManagedObject {

// bytes is part of the persisted state of foo
    byte[] bytes;

// junkString is a transient and is not persisted
    transient String junkString;

// barRef is a reference to a ManagedObject that has its
// own state
    ManagedReference barRef;

    ...
```

So where do you get a ManagedReference from?

Communication

- Two kinds of Darkstar communciation
  - Client/Server
    - Directly btw one client and the server
      - Used to send commands to server and get responses
    - Supported by the kernel
      - Accessed through ClientSession.send(...) on Server
      - Accessed through ServerSession.send(...) on Client
  - Public/Subscribe Channels
    - Between clients but controlled by server
    - Supported by ChannelManager
      - Controlled through ChannelManager on server
      - Accessed through ClientChannel/ClientChannelListener on client
    - More efficient: Does not involve persistence and task systems

What will be covered

- The Business of Darkstar (10 min)
- Introduction to the SGS
- The SGS coding model *in brief*
- Real game implementation
- Where to go for more Information
- Q&A

Real game implementation

- BHO: Bunny Hunter Online
  - A real,little SGS game written just for this talk
    - 2D multiplayer action game
    - Kill innocent small furry animals!
    - “Accidentally” kill other hunters!
  - Rules of Game
    - Killing a bunny gets you a point
    - Killing a hunter costs you a point
    - Last hunter alive is the only one to get ANY points

What we'll cover

- Basic intro to design
- A look at the hardest problem
- Demo of result

What we'll cover

- Basic intro to design
- A look at the hardest problem
- Demo of result

First step: Consider technical challenges

- Consider your latency issues early
  - Set design limits
    - BHO designed for approximately 1000msec worst case
  - Consider results of latency spikes
    - Real time game
    - 2D provides too much information for “fudging”
    - Don't want sum of worst cases
    - Game input is too high frequency to guess/correct
  - Use Age of Empires solution
    - Runs on 1000msec tick
    - Lagging user stalls while game continues for others
    - Consequences – might be unplayable for a user on really bad connection

First step: Consider technical challenges

- Consider your scaling issues early
  - Set design limits
    - BHO is a small group game.
    - Natural design lends itself to 4 players per board
      - Nice low N for N-squared
      - Means a LOT of boards
        - Boards don't have to inter-communicate!
        - Boards can process in parallel!
        - Boards can interleave processing!
      - Lobby is potential bottle neck
        - Only talk to lobby at start and end of game!
        - Might have to scale out lobbies if we ever add lobby chat.

Second step: Define your ManagedObjects

- Entities
  - Hunters
  - Bunnies
  - Carrots
  - Hedges
  - Game Board
  - MCP

Second step: Define your ManagedObjects

- Entities
  - Hunters
    - Represent players
    - Can move on board
    - Can shoot bunnies
    - Can shoot each other
    - Can plant carrots
    - Can die
  - Bunnies
  - Carrots
  - Hedges
  - Game Board
  - MCP

Second step: Define your ManagedObjects

- Entities
  - Hunters
  - Bunnies

    - Robots
    - Spawned by system periodically
    - Attracted to carrots
    - Afraid of gunshots
    - Can be killed
  - Carrots
  - Hedges
  - Game Board
  - MCP

Second step: Define your ManagedObjects

- Entities
  - Hunters
  - Bunnies
  - Carrots

    - Can be planted by players
    - Limited (cost points to plant?)
    - Consumed by bunnies
  - Hedges
  - Game Board
  - MCP

Second step: Define your ManagedObjects

- Entities
  - Hunters
  - Bunnies
  - Carrots
  - Hedges
    - Block movement
    - Block shots
  - Game Board
  - MCP

Second step: Define your ManagedObjects

- Entities
  - Hunters
  - Bunnies
  - Carrots
  - Hedges
  - Game Board
    - Play space for all of the entities above
    - Rectangular grid
    - All entities exist in one square at a time
    - Game play is 1/10sec per turn real-time
    - 4 hunters per game board

Drawn on the fly be combining background with other in-game objects.

Second step: Define your ManagedObjects

- Entities
  - Hunters
  - Bunnies
  - Carrots
  - Hedges
  - Game Board
  - MCP (with apologies to TRON)
    - Collects users
    - Creates game board
    - Keeps high score board
    - A simple “lobby' mechanism

Logic Object (No in game representation)

Second step: Define your ManagedObjects

- Mapping of Entities to SGS Events/Interfaces
  - Hunters
    - Handle user packets: ManagedObject, ClientSessionListener
  - Bunnies and Carrots
    - Just game world constructs: ManagedObject
  - Hedges
    - Terrain: nothing (just part of state of Game Board)
  - Game Board
    - Gets a .5 sec tick: ManagedObject, Task
  - MCP
    - Handles logon/logoff: ManagedObject, AppListener

Logic Object (No in game representation)

What we'll cover

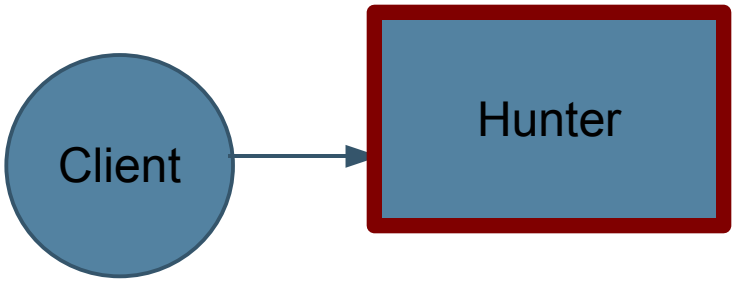
- Basic intro to design
- A look at the hardest problem
- Demo of result

Third step: Solve Client/Server connectivity

- Do the hardest part first
  - Hardest thing is smooth and “good feeling” game-play across varying latency conditions
  - “Walking” is generally your worst case.
    - If you can navigate comfortably, the rest is usually easy
- Recall:
  - Our game runs on a game-tick of .5 sec
  - Challenge is to make that feel smooth to player
  - Secondary challenge is to make it perform well on server
  - Key to server performance is avoiding object contention

First Movement Algorithm:

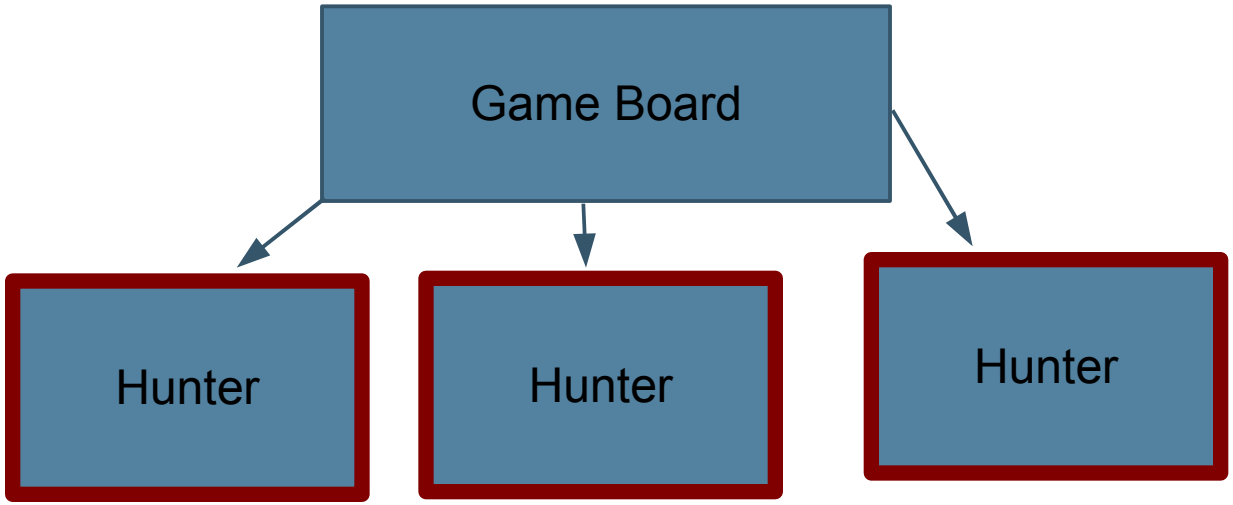
- Client sends move packet to hunter
  - Hunter queues packet in a list
-



Lock taken on hunter to update queue

First Movement Algorithm:

- On game board tick (1 “move”)
  - GameBoard reads each Hunter's queue
  - GameBoard updates Hunter's state (position)
-



Lock taken on hunters to update queue and state

First Movement Algorithm:

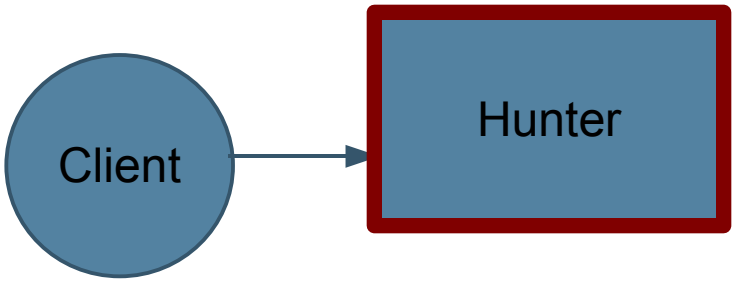
- Problems
  - Massive contention on hunters
    - Burdens CPU
    - Stalls out game with more then 1 hunter
  - No frame synchronization
    - Latency spikes can cause synch to fail between client and server
    -

Second Movement Algorithm:

- Refactor into non competing chunks of data
  - Keep player move on Hunter object
  - Move state to seperate Managed Object
- Non locking movement read
  - Only hold one move at a time
  - Move has incrementing “move count”
  - Store last move count number in state

First Movement Algorithm:

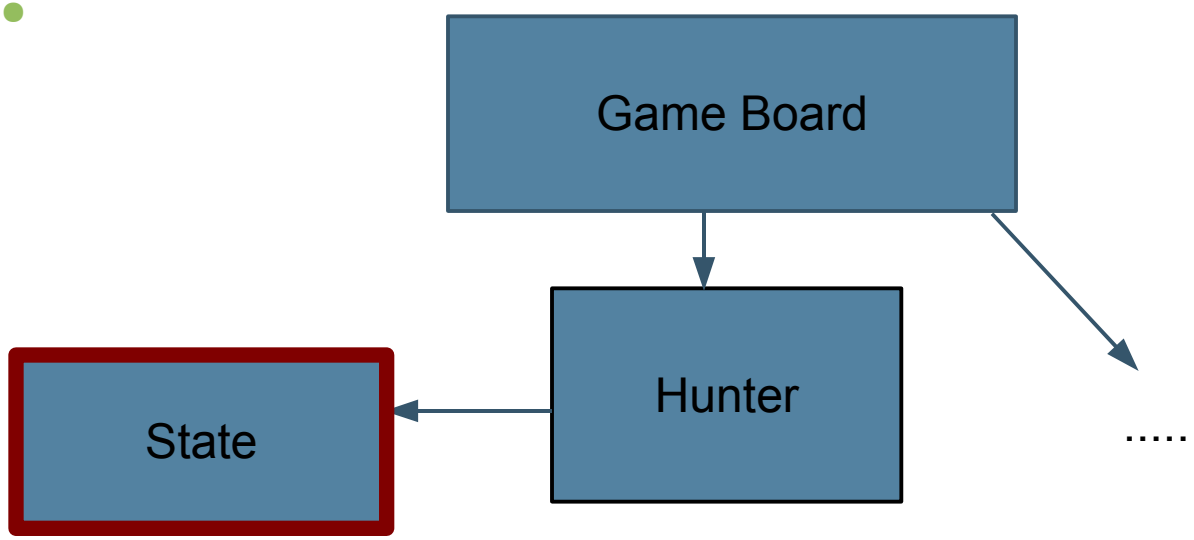
- Client sends move packet to hunter
  - Hunter over-writes any previous move with this one
-



Lock taken on hunter to write move

Second Movement Algorithm:

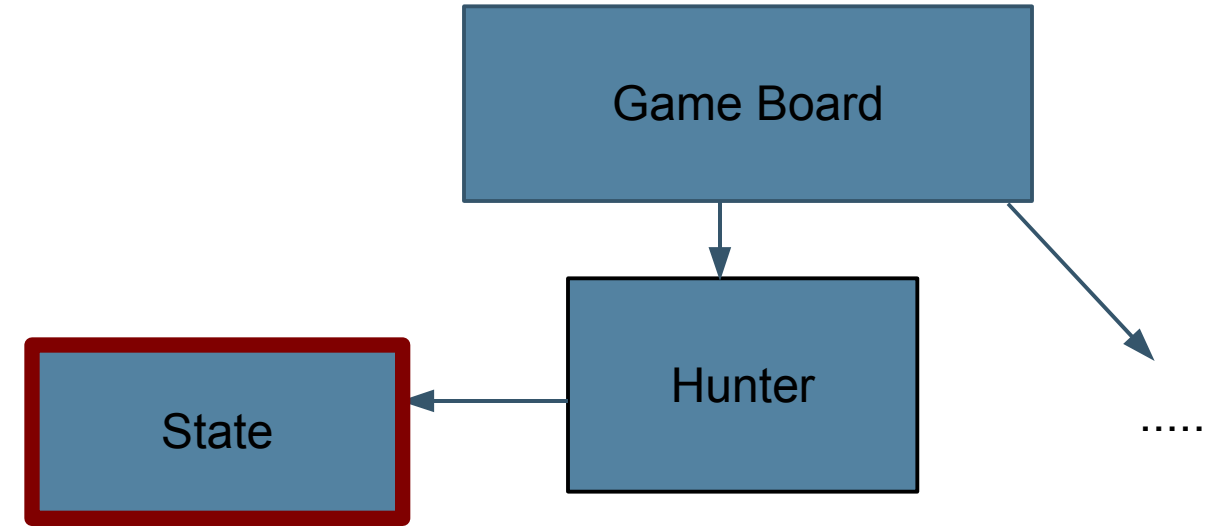
- On game board tick (1 “move”)
  - GameBoard gets state from Hunters.
  - GameBoard locks Hunter's state object
-



Only state is locked

Second Movement Algorithm:

- Game Board reads Move from hunter
- GameBoard reads last move number from stae
- If last move< this move's move number
  - Take move and update last move number



Only state is locked

Second Movement Algorithm

- Advantages
  - No contention
  - Game-play goes on in-step no matter what
    - Player with latency spike is just skipped
  - Moves don't back up on latency spike
    - Actually good we throw away those that the player hasn't had feedback for.
- Problems
  - Have to wait for each frame to tick to get feedback
  - Feels “laggy”

Third Movement Algorithm

- Almost the same as second
- Start local move immediately
  - Use animation time to cover frame tick
- Block further movement until frame tick
  - Avoid getting ahead of server in a latency spike
  - Preserves game synch
- Advantages
  - Same as second but without the lag
- Disadvantages
  - Player synch approximate (but good enough)

Other actions

- Treated as “moves”
- Only one move per tick
- Ordered move resolution
  - First: shoot,plant,eat
  - Second: walk
  - Shoot before walk assures fairness of shoot hit/miss determination.
- In such a simple game, with the connectivity solved the rest is just coding.

DEMO

Bunny Hunters Online!

Character and Object Art and Animation created by Va Lee and Kevin O'Neill courtesy of Mind Control Software, INC.

Special thanks to Andrew Leker and Eduardo Baraf

Summary

- Darkstar vastly simplifies writing scalable, fault tolerant on-line games
  - BHO written in approx. 1 man week for client *and* server.
- Opportunity for small and big developers
  - Get in without $30M
    - SGS SDK is Open Source
    - Playground program provides free beta hosting
    - Hosting providers can revenue share
  - Do games you could never afford before
  - Do games across platforms!

For More Information

See...

- Lab 7210: Hands-on with Project Darkstar: The JavaOne Conference MUD
- www.projectdarkstar.com
- For client coding resources in Java:
  - Slick
    - 2D API used for BHO!
      - http://slick.cokeandcode.com
  - Jmonkey Engine
    - 3D game engine being used commeercially
      - http://www.jmonkeyengine.com
  - Java game developer community
    - www.javagaming.org

Q&A

Jeff Kesselman
Chris Melissinos