Writing Massively Multiplayer Online Games With Project Darkstar

Chris Melissinos
Chief Gaming Officer

Jeffrey Kesselman
Chief Darkstar Architect

Sun Labs East, Sun Microsystems
www.projectdarkstar.com

Java

Sun microsystems

Project 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 Project Darkstar (10 min)
- Introduction to Project Darkstar
- The SGS Coding Model IN BRIEF
- Real Game Implementation: DarkMMO
- Where to Go for More Information
- Q&A

What Will Be Covered

- **The Business of Project Darkstar (10 min)**
  - Chris Melissinos, Sun Chief Gaming Officer
- Introduction to Project Darkstar
- The SGS Coding Model IN BRIEF
- Real Game Implementation: DarkMMO
- Where to Go for More Information
- Q&A

Why Project Darkstar?

- Personal interest
- Sun Microsystems' experience in on-line systems
- New approach to "old" problem
- Interest in growing the on-line 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
- Touch the player on mobile, set-top, PC
- Levels the playing field for all developers
- Explosion of niche content possible through Project Darkstar

New Business Opportunities

- Open Source can be free as in “free puppy”
- Services—Sun is best positioned to support
- We make systems and solutions—Project Darkstar brand servers
- On-line Services—the cable TV model for on-line games
- Commercial licensing—innovate and expand, but you own it

What Will Be Covered

- The Business of Project Darkstar (10 min)
- **Introduction to Project Darkstar**
  - Jeff Kesselman, Chief Instigator
  - Goals and Purpose of Project Darkstar
  - Architecture of Project Darkstar
- The SGS Coding Model IN BRIEF
- Real Game Implementation: DarkMMO
- 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 (MMO)game industry
  - $20M - $30M base budget for an 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

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

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
    - Analogous to the “edge tier”
  - Tier 2: Execution Kernel
    - Executes “tasks” in response to “events”
    - Analogous to a Java™ 2 Platform, Enterprise Edition (J2EE™) platform app server
  - Tier 3: Object Store
    - Lightening fast, highly scalable access to persistent objects
    - Analogous to the DB tier

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 Java 2 Platform, Standard Edition (J2SE™ platform)
    - Stateless
    - **Optimized for lowest worst case latency**

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 with 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**

What Will Be Covered

- The Business of Project Darkstar (10 min)
- Introduction to Project Darkstar
- **The Project Darkstar Coding Model**
  - Tasks and Managers
  - ManagedObjects and ManagedReferences
  - AppContext and Managers
  - Communication
- Real Game Implementation: DarkMMO
- 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 Project Darkstar infrastructure
    - initialize() and loggedIn()
  - Manager events
    - Generated by Project Darkstar managers
    - Managers are pluggable additions to the infrastructure
    - All events other then those above come from managers
- Events result in a Task being queued for execution

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 game objects
    - 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

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

Events and Tasks

Event listener interfaces

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

Managers

- Are pluggable components of the infrastructure
- Present an API to the application
- Generate events
- Three standard managers:
  - Task Manager
  - Data Manager
  - Communication Manager

ManagedObjects

Persistent Project Darkstar objects

- An app is made of ManagedObjects
- ManagedObjects
  - Live in the object store
  - Are fetched and locked by Tasks
  - 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

ManagedReference

References to ManagedObjects

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

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 Project Darkstar communication
  - Client /Server
    - Directly between 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
  - Communication Channels
    - From server to many clients
    - Supported by ChannelManager
      - Controlled through ChannelManager on server
      - Accessed through ClientChannel/ClientChannelListener on client
    - More efficient then single cast

What Will Be Covered

- The Business of Project Darkstar (10 min)
- Introduction to Project Darkstar
- The SGS Coding Model IN BRIEF
- **Real Game Implementation**
- Where to Go for More Information
- Q&A

Real Game Implementation

- DarkMMO
  - A basic MMORPG skeleton
    - Movement
      - Server side cheat detection
      - Zone traversal
    - Chat
      - Scoped to area
    - Inventory management
      - Get/Drop
    - Server side logic
      - Open/Close/Use

What We Will Cover

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

What We Will Cover

- Basic intro to design
- Demo of result

First Step: Consider Technical Challenges

- Consider your latency issues early
  - Set design limits
    - DarkMUD designed for approximately 1000msec worst case
  - Consider results of latency spikes
    - Real time game
    - Limited guess/correct possible
  - Easiest just to allow remote lag
    - Respond locally immediately
      - Check results with server for correctness
    - Allow others to lag on screen
      - Loose combat model covers most ills

First Step: Consider Technical Challenges

- Consider your scaling issues early
  - Set design limits
    - DarkMUD should scale to tens of thousands of simultaneous players
  - N-squared is the enemy
    - Divide and conquer
      - World is zoned
      - Zones are tile-based
      - Limit awareness to size of one tile
    - Global chat needed
      - Limit to Buddy-lists

Second Step: Define Your ManagedObjects

- Entities
  - Players
  - Characters
  - MOBs
  - Placeables
  - Zones
  - Tiles
  - Walk Meshes

Second Step: Define Your ManagedObjects

- Entities
  - Players
    - Represent a user account
    - Has permissions
    - Owns multiple characters
  - Characters
  - MOBs
  - Placeables
  - Zones
  - Tiles
  - Walk Meshes

Second Step: Define Your ManagedObjects

- Entities
  - Players
  - Characters
    - Player's proxies in the game world
    - Have game statistics
    - Move around
    - Fight
    - Interact
  - MOBs
  - Placeables
  - Zones
  - Tiles
  - Walk Meshes

Second Step: Define Your ManagedObjects

- Entities
  - Players
  - Characters
  - MOBs (Mobile Objects)
    - Non-player characters
    - Move around
    - Have AI
    - Have game statistics
    - Fight
  - Placeables
  - Zones
  - Tiles
  - Walk Meshes

Second Step: Define Your ManagedObjects

- Entities
  - Players
  - Characters
  - MOBs
  - Placeables
    - Can block movement
    - Do not move around
      - May be moved (?)
    - Interact with players and MOBs
  - Zones
  - Tiles
  - Walk Meshes

Second Step: Define Your ManagedObjects

- Entities
  - Players
  - Characters
  - MOBs
  - Placeables
  - Zones
    - Play spaces
    - Made up of tiles
    - Can support transfer from/to
    - Support Players, MOBs and Placeable
  - Tiles
  - Walk Meshes

Second Step: Define Your ManagedObjects

- Entities
  - Players
  - Characters
  - MOBs
  - Placeables
  - Zones
  - Tiles
    - Discrete chunks of geometry
    - Have associated walk mesh
  - Walk Meshes

Second Step: Define Your ManagedObjects

- Entities
  - Players
  - Characters
  - MOBs
  - Placeables
  - Zones
  - Tiles
  - Walk Meshes
    - Simplified tile geometry
    - Labeled with walkable/unwalkable
    - Used to calculate movement blocking
    - Could support terrains
      - E.g., slow walk in “mud”

Second Step: Define Your ManagedObjects

- Mapping of Entities to Events/Interfaces
  - Players
    - Handle user packets: ManagedObject, ClientSessionListener
  - Characters
    - Just game world constructs: ManagedObject
  - MOBs
    - Need to run AI: ManagedObject, Task
  - Zones and Placeables
    - May have timed effects: ManagedObject, Task
  - Tiles, Walk Meshes
    - Just game world constructs: ManagedObject
  - MCP
    - Handles logon/logoff: ManagedObject, AppListener

What We Will Cover

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

Third Step: Solve N-squared for motion

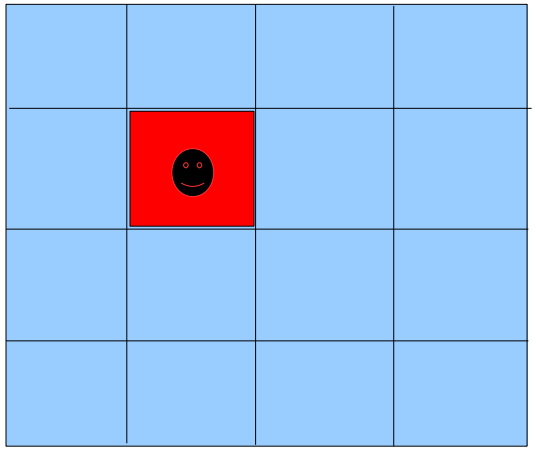
- Do the hardest part first
  - "Walking" is generally your worst case
    - If you can navigate comfortably, the rest is usually easy
    - Movement creates a lot of packets
- Recall:
  - We can allow remote players to lag
  - We can start local movement immediately
  - We can allow remote players to "lag" on local screen
  - Need to cut down number of other players with which we are communicating.

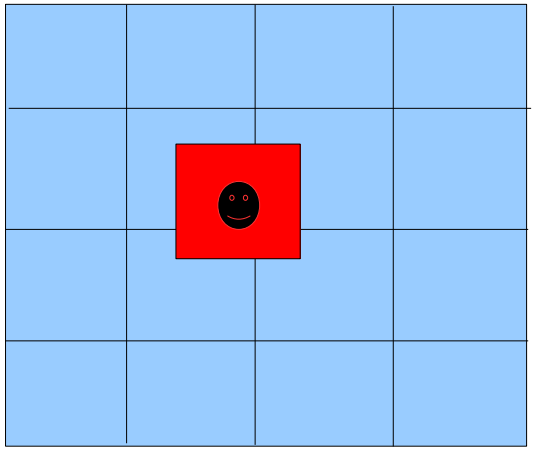
Naïve Solution

- Dynamic space partitioning
  - Quadtree
  - Large structure that must be searched on every position change
  - Must be frequently modified
  - All this spells *contention*
    - Can kill multi-threaded performance

Better Solution

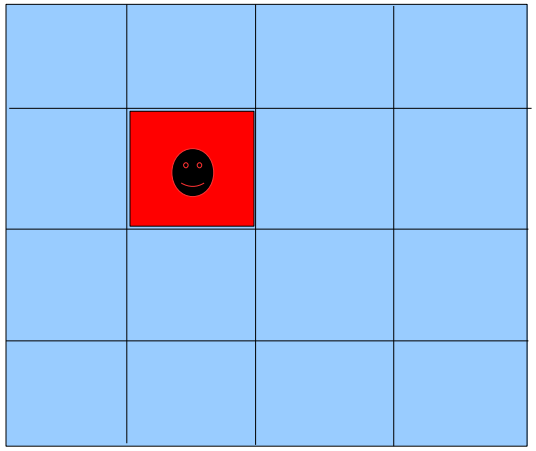
- Grid the Zone
  - Cell size is equal to maximal view distance
    - What cant be seen does not matter
  - Can see at most 4 cells at a time

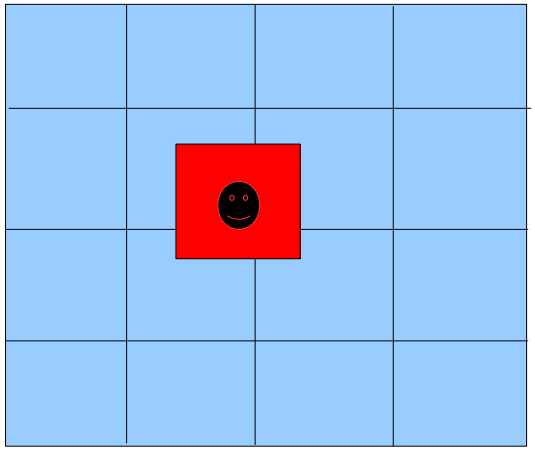




Divide and conquer

- Attach a channel to each cell
  - Player talks to current cell's channel
  - Player listens to the 4 covered by their view
    - Local update to cell contents
    - Low contention





Other Issues

- Positional differences in combat
  - Exact position does not matter
  - Range does
  - Distance between players could be different on different screens
  - Two Cases: Hand to Hand and Ranged

Hand to Hand Combat

- Have a fixed “close range”
- If server decides attacker is within close range
  - Serve locks defender in place
  - Server moves attacker to within melee range
- Issues
  - Traps?
    - Auto detect and walk around or abort attack?

Ranged Combat

- Viewers can't tell exact range
- As long as server thinks target is in range, allow attack
- Server checks blocking
  - If blocked, server sets arrow flight animation
  - If hit, client sets arrow flight animation
- Issues
  - Arrow might occasionally penetrate wall or other block in client's view

DarkMUD

All artwork is from the commercial game Neverwinter Nights

Summary

- Project Darkstar vastly simplifies writing scalable, fault tolerant on-line games
  - BHO written in approximately 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
  - Create games you could never afford before
  - Create games across platforms!

For More Information

Lab 7210—Hands-on with Project Darkstar: The JavaOneSM Conference MUD

- For client coding resources in Java code:
  - Slick
    - 2D API used for BHO! and Project Darkstar coursework
      - http://slick.cokeandcode.com
  - JMonkey Engine
    - 3D game engine used for DarkMUD (also being used commercially and in Wonderland)
      - http://www.jmonkeyengine.com
  - Java platform game developer community
    - www.javagaming.org

For More Information

Business Contacts

Jennifer Kotzen – Project Manager
Karl Haberl – Sun Labs Director
Mike Gialis – Managed Services Development

Technical Contacts

Jeff Kesselman – Chief Instigator
Jim Waldo – Lead Architect
www.projectdarkstar.com

Lab 7210—Hands-on with Project Darkstar: The JavaOneSM Conference MUD

Q&A

Jeff Kesselman | Chris Melissinos

THANK YOU

Chris Melissinos
Chief Gaming Officer

Jeffrey Kesselman
Chief Darkstar Architect

Sun Labs East, Sun Microsystems
www.projectdarkstar.com