Emerson: Scripting for Federated Virtual Worlds

Bhupesh Chandra*, Ewen Cheslack-Postava*, Behram F.T. Mistree*, Philip Levis*, David Gay†
*Stanford University
†Intel Labs
Virtual Worlds

Second Life

WoW
Virtual Worlds

Second Life  WoW

Avatar
Virtual Worlds

Second Life

Avatar

Building (Entity)

WoW
Virtual Worlds

Second Life

Tree (Entity)

Avatar

Building (Entity)

WoW

Fighter (Entity)
Virtual Worlds

Tree (Entity)
Avatar
Building (Entity)

Ship (Entity)
Fighter (Entity)

Second Life  WoW
Scripted Entities

Adalace Jewell - Nonprofit Commons

default
{
  state_entry()
  {
    llSay(0, "Hello, Avatar!");
  }
  touch_start(integer total_number)
  {
    llSay(0, "Touched.");
  }
}
Scripted Entities

Second Life

Scripted Entity

Wednesday, October 13, 2010
Scripted Entities

Second Life

Scripted Entity

Script

Wednesday, October 13, 2010
Scripting in VW
Scripting in VW

- Lua (WoW), LSL (Second Life), UScript (Unreal)
Scripting in VW

- Lua (WoW), LSL (Second Life), UScript (Unreal)
- Emerson
  - Scripting language for future VW
  - Easy to Script
Future Virtual Worlds
Future Virtual Worlds

• Federation
  • Multiple parties cooperate to run the world
  • Web users can not only create but host their own content => extensible, flexible
Future Virtual Worlds

• **Federation**
  - Multiple parties cooperate to run the world
  - Web users can not only create but host their own content => extensible, flexible

• **Seamless and Scalable**
  - Distributed simulation of billions of entities
  - Entities must interact over the network
Ease of Scripting
Ease of Scripting

- Opportunistic programming
  - Copy-paste and modify, code reuse
Ease of Scripting

• Opportunistic programming
  • Copy-paste and modify, code reuse

• Iterative Development
  • Continuously running world
  • Modify entity without terminating execution
Design Challenges
Design Challenges

• Lack of trust between entities
  • Protect against untrusted operations
Design Challenges

- Lack of trust between entities
- Protect against untrusted operations
- Distributed Simulation of entities
- Large latencies, packet losses and node failures
Design Challenges

• Lack of trust between entities
• Protect against untrusted operations
• Distributed Simulation of entities
• Large latencies, packet losses and node failures
• Live, incremental scripting
Emerson: Main Features
Emerson: Main Features

- Entity, Presence, Object
- Federation and distributed simulation
Emerson: Main Features

- Entity, Presence, Object
  - Federation and distributed simulation

- Prototyping
  - Code reuse through prototypes
Emerson: Main Features

• **Entity, Presence, Object**
  • Federation and distributed simulation

• **Prototyping**
  • Code reuse through prototypes

• **Live Programming**
  • Execute code dynamically to modify behavior
    (more in the paper)
Emerson: Main Features

- Entity, Presence, Object
  - Federation and distributed simulation

- Prototyping
  - Code reuse through prototypes

- Live Programming
  - Execute code dynamically to modify behavior
    (more in the paper)

- Event Driven Pattern Matching
  - Address sphagetti if-else problem
Fundamentals
Fundamentals

• **Objects** encapsulate state and functionality
Fundamentals

- **Objects** encapsulate state and functionality
- **Entities** contain objects, event handlers
  - Obtain presences
  - Communicate with entities in the same world
  - Objects exist and are addressable within entity
Fundamentals

• **Objects** encapsulate state and functionality

• **Entities** contain objects, event handlers
  • Obtain presences
  • Communicate with entities in the same world
  • Objects exist and are addressable within entity

• **Presence** is connection of entity in the world
  • Geometry, communication
Art Gallery
Art Gallery

Artist Entity Host
LONDON

VirtualWorld

Entity

Artist Entity

Artist Entity Host

Entity Host
Art Gallery

LONDON

VirtualWorld

Entity

Artist Entity Host

Entity Host

Artist Entity
Art Gallery

Artist Avatar (Presence)

Entity

Entity Host

Artist Entity Host

Art Gallery Entity

Artist Entity

Entity

VirtualWorld

LONDON
Art Gallery

LONDON

VirtualWorld

Art Gallery (Presence)

Artist Avatar (Presence)

Entity

Entity

Entity Host

Artist Entity Host

Object

root

getArtDetails()

inventory

Art Gallery Entity

Artist Entity

Art Gallery (Presence)
Entities
Entities

• Communicate by sending asynchronous messages over the network

• Short event handlers; don’t block other entities (Helps Seamless Scaling)
Entities

- Communicate by sending asynchronous messages over the network
- Short event handlers; don’t block other entities (Helps Seamless Scaling)
- Boundary of trust (Helps Federation)
- Exclusive right to change their state/behavior
Code Reuse
Code Reuse

• Important for easy scripting
Code Reuse

• Important for easy scripting

• Class based Inheritance (Java)

• Subclassing
Code Reuse

- Important for easy scripting

- Class based Inheritance (Java)
  - Subclassing

- Prototype based (JavaScript)
  - No Classes
  - Objects inherit from objects
Code Reuse

- Important for easy scripting
- Class based Inheritance (Java)
  - Subclassing
- Prototype based (JavaScript)
  - No Classes
  - Objects inherit from objects
Code Reuse

- Important for easy scripting

- Class based Inheritance (Java)
  - Subclassing

- Prototype based (JavaScript)
  - No Classes
  - Objects inherit from objects

List

| size=2 |
| length() |

CustomerQueue

| size=4 |
| next() |
| prototype |
Code Reuse

- Important for easy scripting
- Class based Inheritance (Java)
  - Subclassing
- Prototype based (JavaScript)
  - No Classes
  - Objects inherit from objects

CustomerQueue

List
  - size=2
  - length()

size=4
  - next()
  - prototype
Code Reuse

- Important for easy scripting
- Class based Inheritance (Java)
  - Subclassing
- Prototype based (JavaScript)
  - No Classes
  - Objects inherit from objects

List
- size=2
- length()

CustomerQueue
- size=4
- next()
- prototype

prototype
Code Reuse

• Important for easy scripting

• Class based Inheritance (Java)
  • Subclassing

• Prototype based (JavaScript)
  • No Classes
  • Objects inherit from objects

List
  • size=2
  • length()

Prototype based extension

CustomerQueue
  • size=4
  • next()
  • prototype
Code Reuse

- Important for easy scripting
- Class based Inheritance (Java)
  - Subclassing
- Prototype based (JavaScript)
  - No Classes
  - Objects inherit from objects

CustomerQueue

CustomerQueue.length() ?
Code Reuse

• Important for easy scripting

• Class based Inheritance (Java)
  • Subclassing

• Prototype based (JavaScript)
  • No Classes
  • Objects inherit from objects

List
  size=2
  length()

CustomerQueue
  size=4
  next()
  prototype

extension

prototype

CustomerQueue.length()?
Code Reuse

- Important for easy scripting
- Class based Inheritance (Java)
- Subclassing
- Prototype based (JavaScript)
- No Classes
- Objects inherit from objects

List

size=2

length()

CustomerQueue

size=4

next()

prototype

extension

prototype

CustomerQueue.length()
Code Reuse

- Important for easy scripting
- Class based Inheritance (Java)
  - Subclassing
- Prototype based (JavaScript)
  - No Classes
  - Objects inherit from objects

List
- size=2
- length()

CustomerQueue
- size=4
- next()
- prototype

extension

prototype

CustomerQueue.length()?
**Code Reuse**

- Important for easy scripting
- Class based Inheritance (Java)
  - Subclassing
- Prototype based (JavaScript)
  - No Classes
  - Objects inherit from objects

```
List
  ^
  size=2
  length()

CustomerQueue
  prototype
  next()
  size=4

CustomerQueue.length() ?
```
Code Reuse

- Important for easy scripting
- Class based Inheritance (Java)
  - Subclassing
- Prototype based (JavaScript)
  - No Classes
  - Objects inherit from objects

CustomerQueue

CustomerQueue.length() ?
Code Reuse

- Important for easy scripting
- Class based Inheritance (Java)
  - Subclassing
- Prototype based (JavaScript)
  - No Classes
  - Objects inherit from objects

List

- size=2
- length()

CustomerQueue

- size=4
- next()
- prototype

extension

prototype

CustomerQueue.length()?
Code Reuse

- Important for easy scripting
- Class based Inheritance (Java)
  - Subclassing
- Prototype based (JavaScript)
  - No Classes
  - Objects inherit from objects

List
  - size=2
  - length()

List
  - size=4
  - next()
  - prototype

extension

CustomerQueue

CustomerQueue.length()?
Code Reuse

- Important for easy scripting
- Class based Inheritance (Java)
  - Subclassing
- Prototype based (JavaScript)
  - No Classes
  - Objects inherit from objects

List

CustomerQueue

size=4

next()

prototype

size=2

length()
Prototypes for Objects
Prototypes for Objects

- Singleton classes are wasteful
Prototypes for Objects

- Singleton classes are wasteful
- Modify class then re-instantiate in some languages
Prototypes for Objects

- Singleton classes are wasteful
- Modify class then re-instantiate in some languages
- Emerson objects inherit “live” modifications to their prototype
Inheritance: Bad For Entities
Inheritance: Bad For Entities
Inheritance: Bad For Entities

Building Entity Host

doorOpen() root

name = House

Gallery Entity Host
Inheritance: Bad For Entities

Building
Entity
Host

root

name = House

doorOpen()

Gallery
Entity
Host

Wednesday, October 13, 2010
Inheritance: Bad For Entities

Building
Entity
Host

doorOpen()

root

name = House

Gallery
Entity
Host

root
Inheritance: Bad For Entities

Building
Entity
Host

root

doorOpen()

name = House

Gallery
Entity
Host

root

prototype = House
Inheritance: Bad For Entities

Building
Entity
Host

doorOpen()

name = House

root

Gallery
Entity
Host

prototype = House

root

Wednesday, October 13, 2010
Inheritance: Bad For Entities

Building
Entity
Host

Gallery
Entity
Host

root

name = Gallery

prototype = House

name = House
doorOpen()
Inheritance: Bad For Entities

- **Building**
  - **Entity**
    - **Host**
      - **root**
        - `doorOpen()`
        - `name = House`

- **Gallery**
  - **Entity**
    - **Host**
      - **root**
        - `ArtInventory`
        - `name = Gallery`
        - `prototype = House`
Inheritance: Bad For Entities

Building
Entity
Host

doorOpen

name = Gallery

ArtInventory

prototype = House

name = Gallery

root

name = House

root

root
Inheritance: Bad For Entities

Building
Entity
Host

name = Gallery

prototype = House

doorOpen()

name = House

root

Gallery
Entity
Host

name = Gallery

root

prototype = House

doorOpen

ArtInventory
Inheritance: Bad For Entities

Building
Entity
Host

root

doorOpen()

name = House

prototype =
House

name = Gallery

ArtInventory

root

prototype =
House

Gallery
Entity
Host

Wednesday, October 13, 2010
Inheritance: Bad For Entities
Entity Prototyping
Entity Prototyping

• **Copy Based Prototyping**
  - No prototype lookup
  - Prototype may be on different entity host
  - Look up requires network messaging
  - Copy existing entity and modify
    - State gets copied too
Prototyping for entities
Prototyping for entities

Building
Entity
Host

root

doorOpen()

name = House
Prototyping for entities

Building
Entity
Host

doorOpen()

root

name = House

Gallery

Entity
Host
Prototyping for entities

Building
Entity
Host

doorOpen()

root

name = House

Gallery
Entity
Host

doorOpen()

root

name = House

Wednesday, October 13, 2010
Prototyping for entities

Building
Entity
Host

doorOpen()
name = House

Gallery
Entity
Host

doorOpen()
name = Gallery
Prototyping for entities

Building
Entity
Host

donOpen()

name = House

Gallery
Entity
Host

donOpen()

ArtInventory

name = Gallery
Events

• Multiple event types in virtual worlds
  • Message send/recv, timers, proximity
• Register callbacks for specific events
• Can be painful to handle
Event handling example

- Single Event handler for all message types
- Lots of if-else
- Ugly to fit in incremental model

```java
listen_for_messages(message_handler);

fun message_handler(sender_id, msg) {
    if( msg.name == "loan" )
    {
        loan_art(sender_id, msg);
    }
    else if( msg.name == "buy" )
    {
        buy_art(sender_id, msg);
    }
    else if( msg.name == "take_money" )
    {
        take_money(sender_id, msg);
    }
}
```
Events in Emerson

- Events are described by patterns
- Patterns are objects matched by field
  - name, value and prototype
- Similar to patterns in Erlang
- `[proto] field[.subfield[...]] [: value]`
- `(action:borrow,item_id)`
Messaging Example
Messaging Example

handleLoan <- (action:borrow,'<customer item_id)
Messaging Example

```plaintext
handleLoan <- (action: borrow, item_id) 

// handleLoan
```
handleLoan <- (action:borrow, item_id)<-customer

// handleLoan
receipt = new Receipt(status = 'OK')
Messaging Example

handleLoan <- (action: borrow, item_id) <- customer

// handleLoan
receipt = new Receipt(status = 'OK')
receipt -> customer
Message Handlers
Message Handlers

LONDON

Art Gallery Entity

Artinventory

root

getArtDetails
Message Handlers

LONDON

Artinventory

getArtDetails

handleLoan

root

Art Gallery Entity
Message Handlers

LONDON

Art Gallery Entity

Artinventory

getArtDetails

handleLoan

Customer Entity

Customer Entity Host
Message Handlers

LONDON

Art Gallery Entity

Artinventory

root

getArtDetails

handleLoan

Customer Entity Host

Customer Entity

Wednesday, October 13, 2010
Message Handlers

- LONDON

- action: borrow
  item_id = 65

- Customer Entity Host

- Art Gallery Entity
  - Artinventory
    - root
    - getArtDetails
    - handleLoan

- Customer Entity
Message Handlers

LONDON

action=borrow
item_id = 65

Artinventory
getArtDetails
handleLoan

customer Entity Host

Art Gallery Entity

Artinventory
getArtDetails
handleLoan

Customer Entity

Wednesday, October 13, 2010
Message Handlers

LONDON

action=borrow
item_id = 65

Customer Entity

Art Gallery Entity

Artinventory

root

getArtDetails

handleLoan

Customer Entity Host

Art Gallery Entity

Artinventory

root
Summary
Summary

- Emerson: Scripting for Federated, Seamless and Scalable VW
- Federation: Entity Isolation
- Scalability: Asynchronous Messaging
Summary

• Emerson: Scripting for Federated, Seamless and Scalable VW
• Federation: Entity Isolation
• Scalability: Asynchronous Messaging
• Easy scripting
  • Reuse prototypes, Incremental Scripting
Summary

- Emerson: Scripting for Federated, Seamless and Scalable VW
- Federation: Entity Isolation
- Scalability: Asynchronous Messaging
- Easy scripting
  - Reuse prototypes, Incremental Scripting
- Event Handling: Pattern based
Current State

- Prototype based on V8 JavaScript interpreter
- Sirikata virtual world (www.sirikata.com)
- Language Library
Future

• Need to extend Emerson for
  • Persistence features
  • Transactions
  • Access control and ownership issues
Thank You
Live Programming

• Entities can dynamically execute scripts
  • `eval` as in JavaScript
• Modification of state and behavior without termination
• Access control to prevent executing arbitrary scripts
Live modification

Artist Entity Host

Gallery Entity

Artist Entity

root

inventory

getArtDetails()

addArt()
Live modification

function deleteArt()
Live modification

function deleteArt()

Artifact Entity Host

Gallery Entity

root

inventory

getArtDetails()

addArt()
Live modification

function deleteArt()

Gallery Entity

Artist Entity Host

Artist Entity

root

inventory

getArtDetails()

addArt()

deleteArt()
Still evolving...

• developing more of the syntactic features
• Writing programs to find common cases and embed these in the language as syntax
• Exposing more of the underlying system functionality into the language
• Language Library
Presences
Presences

• Entities hold references to presences of their own and other entities
Presences

- Entities hold references to presences of their own and other entities
- Communication through presences
Presences

• Entities hold references to presences of their own and other entities
• Communication through presences
• Multiple presences to bridge worlds
  • same entity can service multiple worlds
Emerson
Emerson

- Interpreted language
- similar to JavaScript
Emerson

• Interpreted language
  • similar to JavaScript

• Event-driven execution model
  • Each entity executes single script
  • Script consists of short event handlers
Live Programming

• Entities can dynamically execute scripts
• More in the paper
Scripting in VW
Scripting in VW

• Add behavior to graphical entities
  • Entities execute a program (Scripted Entities)
  • Eg. Lua (WoW), LSL (Second Life), UScript (Unreal)
  • Bulletin Boards, Intelligent Fighters
Summary

• Entities, Presences, Objects

• Code Reuse
  • prototypes for objects
  • copy and modify for entities

• Incremental development by executing arbitrary scripts

• Patterns for events with failure callbacks
Art Gallery
Art Gallery

LONDON

Entity Host

Artist Entity Host
Art Gallery

LONDON

VirtualWorld

Entity

Artist Entity Host

Artist Entity

Entity Host

Wednesday, October 13, 2010
Art Gallery

LONDON
Art Gallery

LONDON

VirtualWorld

Artist Avatar (Presence)

Entity

Entity Host

Artist Entity Host

34

Wednesday, October 13, 2010
Art Gallery

VirtualWorld

Entity Host

Art Gallery (Presence)

Entity

Artist Avatar (Presence)

Artist
Entity

Entity

Object

Art Gallery Entity

Entity Host

Artist Entity Host

getArtDetails()

inventory

root