Distributed (Systems) Programming
Universal Actors, SALSA, World-Wide Computer

Carlos Varela
Rensselaer Polytechnic Institute

November 6, 2006

Worldwide Computing

- Distributed computing over the Internet.
- Access to large number of processors offsets slow communication and reliability issues.
- Seeks to create a platform for many applications.

Overview of programming distributed systems

- It is harder than concurrent programming!
- Yet unavoidable in today’s information-oriented society, e.g.:
  - Internet
  - Web services
  - Grid computing
- Communicating processes with independent address spaces
- Limited network performance
  - Orders of magnitude difference between WAN, LAN, and single machine communication.
- Localized heterogeneous resources, e.g. I/O, specialized devices.
- Partial failures, e.g. hardware failures, network disconnection
- Openness: creates security, naming, composability issues.

Actors/SALSA Revisited

- Actor Model
  - A reasoning framework to model concurrent computations
  - Programming abstractions for distributed open systems
- SALSA
  - Simple Actor Language System and Architecture
  - An actor-oriented language for mobile and Internet computing
  - Programming abstractions for Internet-based concurrency, distribution, mobility, and coordination

World-Wide Computer (WWC)

- Worldwide computing platform.
- Provides a run-time system for universal actors.
- Includes naming service implementations.
- Remote message sending protocol.
- Support for universal actor migration.

Abstractions for Worldwide Computing

- Universal Actors, a new abstraction provided to guarantee unique actor names across the Internet.
- Theaters, extended Java virtual machines to provide execution environment and network services to universal actors:
  - Access to local resources.
  - Remote message sending.
  - Migration.
- Naming service, to register and locate universal actors, transparently updated upon universal actor creation, migration, recollection.
Universal Naming

- Consists of human readable names.
- Provides location transparency to actors.
- Name to location mappings efficiently updated as actors migrate.

Universal Actor Naming

- UAN servers provide mapping between static names and dynamic locations.
  - Example:

```
uan://wwc.cs.rpi.edu/cvarela/calendar
```

  Name server address and port.

  Actor name.

Universal Actors

- Universal Actors extend the actor model by associating a universal name and a location with the actor.
- Universal actors may migrate between theaters and the name service keeps track of their current location.

Universal Actor Implementation

WWC Theaters

- Theaters provide an execution environment for actors.
- Provide a layer beneath actors for message passing and migration.
- Example locator:

```
rms://wwc.cs.rpi.edu/calendarInstance10
```

  Theater address and port.

  Actor location.
Environmental Actors

- Theaters provide access to environmental actors.
- Environmental actors perform actions specific to the theater and are not mobile.
- Include standard input, output and error stream actors.

Remote Message Sending Protocol

- Messages between remote actors are sent using the Remote Message Sending Protocol (RMSP).
- RMSP is implemented using Java object serialization.
- RMSP protocol is used for both message sending and actor migration.
- When an actor migrates, its locator (UAL) changes but its name (UAN) does not.

Universal Actor Naming Protocol

- UANP includes messages for:
  - Binding actors to UAN, UAL pairs
  - Finding the locator of a universal actor given its UAN
  - Updating the locator of a universal actor as it migrates
  - Removing a universal actor entry from the naming service
- SALSA programmers need not use UANP directly in programs. UANP messages are transparently sent by WWC run-time system.

UANP Implementations

- Default naming service implementation stores UAN to UAL mapping in name servers as defined in UANPs.
  - Name server failures may induce universal actor unreachability.
- Distributed (Chord-based) implementation uses consistent hashing and a ring of connected servers for fault-tolerance. For more information, see:

SALSA Language Support for Worldwide Computing

- SALSA provides linguistic abstractions for:
  - Universal naming (UAN & UAL).
  - Remote actor creation.
  - Message sending.
  - Migration.
  - Coordination.
- SALSA-compiled code closely tied to WWC run-time platform.
Universal Actor Creation

- To create an actor locally
  ```java
  TravelAgent a = new TravelAgent();
  ```
- To create an actor with a specified UAN and UAL:
  ```java
  TravelAgent a = new TravelAgent() at (uan, ual);
  ```
- At current location with a UAN:
  ```java
  TravelAgent a = new TravelAgent() at (uan);
  ```

Message Sending

```java
TravelAgent a = new TravelAgent();
a <- book(flight);
```

Remote Message Sending

- Obtain a remote actor reference by name.
  ```java
  TravelAgent a = (TravelAgent)TravelAgent.getReferenceByName("uan://myhost/tas");
a <- printItinerary();
  ```

Reference Cell Service Example

```java
module examples.cell;
behavior Cell implements ActorService{
  Object content;
  Cell() {
    content = initialValue;
  }
  Object get() {
    standardOutput <- println("Returning:"+content);
    return content;
  }
  void set(Object newValue) {
    standardOutput <- println("Setting:"+newValue);
    content = newValue;
  }
}
```

Reference Cell Client Example

```java
module examples.cell;
behavior GetCellValue{
  void act(String[] args) {
    if (args.length != 1){
      standardOutput <- println("Usage:
          salsa examples.cell.GetCellValue <CellUAN>");
      return;
    }
    Cell c = (Cell)Cell.getReferenceByName(new UAN(args[0]));
    standardOutput <- print("Cell Value") @
      c <- get() @
      standardOutput <- println(token);
  }
}
```

Migration

- Obtaining a remote actor reference and migrating the actor.
  ```java
  TravelAgent a = (TravelAgent)TravelAgent.getReferenceByName
    ("uan://myhost/tas");
a <- migrate("cnmp://yourhost/travel") @
a <- printItinerary();
  ```
Moving Cell Tester Example

```java
module examples.cell {
    behavior MovingCellTester {
        void act(String[] args) {
            if (args.length != 3) {
                standardOutput <- println("Usage: salsa examples.cell. MovingCellTester <UAN> <UAL1> <UAL2>");
                return;
            }
            Cell c = new Cell("Hello") at (new UAN(args[0]), new UAL(args[1]));
            standardOutput <- print("Initial Value: @");
c <- get(); standardOutput <- println(token);
            c <- set("World"); standardOutput <- print("New Value: @");
c <- get(); standardOutput <- println(token);
            c <- migrate(args[2]);
c <- set("New World");
            standardOutput <- println("New Value at New Location: @");
        }
    }
}
```

Agent Migration Example

```java
behavior Migrate {
    void print() {
        standardOutput <- println("Migrate actor is here.");
    }
    void act(String[] args) {
        if (args.length != 3) {
            standardOutput <- println("Usage: salsa migration.Migrate <UAN> <srcUAL> <destUAL>");
            return;
        }
        UAN uan = new UAN(args[0]);
        UAL ual = new UAL(args[1]);
        Migrate migrateActor = new Migrate() at (uan, ual);
        migrateActor <- print();
        migrateActor <- migrate(args[2]);
        migrateActor <- print();
    }
}
```

Compilation and Execution

```
$ java salsaCompiler demo/Migrate.salsa
SALSA Compiler Version 1.0: Reading from file demo/Migrate.salsa . . .
SALSA Compiler Version 1.0: SALSA program parsed successfully.
SALSA Compiler Version 1.0: SALSA program compiled successfully.
$ javac demo/Migrate.java
$ java demo.Migrate
Usage: java migration.Migrate <uan> <srcUAL> <destUAL>
```

Migration Example

- The program must be given valid universal actor name and locators.
  - Appropriate name services and theaters must be running.
- After remotely creating the actor. It sends the print message to itself before migrating to the second theater and sending the message again.

```
UAN Server

<table>
<thead>
<tr>
<th>Host</th>
<th>Location</th>
<th>OS/JVM</th>
<th>Processor</th>
</tr>
</thead>
<tbody>
<tr>
<td>solar.isr.co.jp</td>
<td>Tokyo, Japan</td>
<td>Solaris 2.6 JDK 1.1.6</td>
<td>Sparc 20</td>
</tr>
<tr>
<td>vulcain.ecoledoc.lip6.fr</td>
<td>Paris, France</td>
<td>Linux 2.2.5 JDK 1.2pre2</td>
<td>Pentium II 350Mhz</td>
</tr>
<tr>
<td>yangtze.cs.uiuc.edu</td>
<td>Urbana IL, USA</td>
<td>Solaris 2.5.1 JDK 1.1.6</td>
<td>Ultra 2</td>
</tr>
</tbody>
</table>
```

World Migrating Agent Example

- Compile Migrate.salsa file into Migrate.java.
- Compile Migrate.java file into Migrate.class.
- Execute Name Server
- Execute Theater 1 and Theater 2 Environments
- Execute Migrate in any computer
Address Book Service

```java
module examples.addressbook;

behavior Addressbook implements ActorService {
    Hashtable samemail;
    Addressbook() {
        samemail = new Hashtable();
    }
    String getemail(String email) {
        boolean address(String name, String email) {
            }
        }
        void act(String[] args) {
            // code...
        }
    }

    void getUser(String[] args) {
        // code...
    }
    void addUser(String[] args) {
        // code...
    }
    void migrate(String[] args) {
        // code...
    }
}

C. Varela 31
```

Address Book Add User Example

```java
module examples.addressbook;

behavior Addusers {
    void act(String[] args) {
        if (args.length > 2) {
            standardOutput.println("Usage: salsa examples.addressbook Addressbook -addUser <name> <email>");
            return;
        }
        Addressbook book = new Addressbook();
        book.adduser(args[1], args[2]);
    }
}

C. Varela 32
```

Address Book Get Email Example

```java
module examples.addressbook;

behavior Getemail {
    void act(String[] args) {
        if (args.length > 2) {
            standardOutput.println("Usage: salsa examples.addressbook Addressbook -getEmail <name>");
            return;
        }
        standardOutput.println("Usage: salsa examples.addressbook Addressbook -getEmail <name>");
    }
    void getUser(String[] name) {
        Addressbook book = new Addressbook();
        book.getemailByName(name);
        standardOutput.println(book.toString());
    }
}

C. Varela 33
```

Address Book Migrate Example

```java
module examples.addressbook;

behavior Migrates {
    void act(String[] args) {
        if (args.length > 2) {
            standardOutput.println("Usage: salsa examples.addressbook Addressbook -migrate <name> <email>");
            return;
        }
        Addressbook book = new Addressbook();
        book.migrate(args[1]);
    }
}

C. Varela 34
```

Middleware for Autonomous Computing

- **Middleware:**
  - A software layer between distributed applications and operating systems.
  - Supports applications that run on separate machines or grids.
  - Reduces the complexity of distributed systems.

- **Internet Operating System (IOS):**
  - A virtual layer that hides the underlying network from applications.
  - Allows applications to operate on different platforms and network architectures.

- **Quality of Service:**
  - Assures that networked systems meet the requirements of applications.

- **Log Management:**
  - Manages log files generated by different applications.

- **Network Addressing:**
  - Assigns unique addresses to networked systems.

- **Security:**
  - Ensures that networked systems are secure from unauthorized access.

- **Fault-Tolerance:**
  - Design and implementation of fault-tolerant systems.

- **Load Balancing:**
  - Distributes network traffic among multiple servers.

- **Concurrency Control:**
  - Synchronizes access to shared resources.

- **Distributed Transaction Management:**
  - Manages transactions in distributed systems.

- **Virtual Environment:**
  - Provides a virtual environment for applications to run in.

- **ACID Properties:**
  - Atomicity, Consistency, Isolation, Durability.

- **Distributed Locking:**
  - Ensures that multiple applications can access shared resources in a coordinated manner.

- **Virtual Memory:**
  - Stores data in virtual memory that is not physically present.

- **Object-Oriented Middleware:**
  - Provides a framework for building distributed applications.

- **Peer-to-Peer Networking:**
  - Enables peer-to-peer communication between networked systems.

- **Pluggable Transport:**
  - Allows applications to use different communication protocols.

- **Naming Service:**
  - Provides a mechanism for naming network resources.

- **Message Passing:**
  - Enables communication between networked systems.

- **Network Monitoring:**
  - Tracks network performance and identifies bottlenecks.

**SALSA application layer**
- Programming language constructs for user communication, regulation, and coordination.

**IOS middleware layer**
- A Resource Profiling Component
  - Captures information about actor and network topologies and available resources.
- A Decision Component
  - Takes migration, split/merge, or replication decisions based on profiled information.
- A Protocol Component
  - Performs communication between nodes in the middleware system.

**World-Wide Computer Architecture**

- **SALSA application layer**
  - Programming language constructs for user communication, regulation, and coordination.

- **IOS middleware layer**
  - A Resource Profiling Component
    - Captures information about actor and network topologies and available resources.
  - A Decision Component
    - Takes migration, split/merge, or replication decisions based on profiled information.
  - A Protocol Component
    - Performs communication between nodes in the middleware system.

- **WWC run-time layer**
  - Theories provide runtime support for actor execution and access to local resources.
  - Pluggable transport, naming, and messaging services.
Actor Garbage Collection

- Implemented since SALSA 1.0 using pseudo-root approach.
- Includes distributed cyclic garbage collection.
- For more details, please see:


Distributed Systems Visualization

- Generic online Java-based distributed systems visualization tool
- Uses a declarative Entity Specification Language (ESL)
- Instruments byte-code to send events to visualization layer.
- For more details, please see:


Exercises

66. How would you implement the join continuation linguistic abstraction considering different potential distributions of its participating actors?

67. Download and execute the Agent.salsa example.

68. *Modify the lock example to include a wait/notify protocol, as opposed to "busy-waiting" (or rather "busy-asking").

69. *VRH Exercise 11.11.3 (pg 746). Implement the example using SALSA/WWC.