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

Carlos Varela
Rensselaer Polytechnic Institute

November 3, 2009

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.
- Provide unique identifiers for actors on their creation.
Environment Actors

• Theaters provide access to environment actors.
• Environment 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 location (host:port) changes but its name (UAN) does not.

Universal Actor Naming Protocol

• UANP includes messages for:
  – Binding actors to UAN, host:port 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 host/port mapping in name servers as defined in UANs.
  – 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 (UANs).
  – 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, host, port);
  ```
- 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 = reference TravelAgent("uan://myhost/ta");
a <<< printItinerary();
  ```

Reference Cell Service Example

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

Reference Cell Client Example

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

Migration

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

```java
module examples.cell {
    behavior MovingCellTester {
        void act(String[] args) {
            if (args.length != 5) {
                standardOutput <- println("Usage: java examples.cell.MovingCellTester <UAN> <host1> <port1> <host2> <port2>");
                return;
            }

            Cell c = new Cell("Hello") at (new UAN(args[0]), args[1], Integer.parseInt(args[2]));

            standardOutput <- print("Initial Value: @");
            c <- get();
            standardOutput <- println(token);;

            c <- set("World");
            standardOutput <- print("New Value: @");
            c <- get();
            standardOutput <- println(token);

            c <- migrate(args[3], Integer.parseInt(args[4]));
            c <- set("New World");
            standardOutput <- print("New Value at New Location: @");
            c <- get();
            standardOutput <- println(token);
        }
    }
}
```

Agent Migration Example

```java
module Migrate {
    void print() {
        standardOutput <- println("Migrate actor is here.");
    }

    void act(String[] args) {
        if (args.length != 5) {
            standardOutput <- println("Usage: java migration.Migrate <UAN> <src_host> <src_port> <dest_host> <dest_port>");
            return;
        }

        UAN uan = new UAN(args[0]);
        Migrate migrateActor = new Migrate() at (uan, args[1], Integer.parseInt(args[2]));

        migrateActor <- print();
        migrateActor <- migrate(args[3], Integer.parseInt(args[4]));
        migrateActor <- print();
    }
}
```

Compilation and Execution

```
$ java salsa.core.compiler.SalsaCompiler Migrate.salsa
SALSA Compiler Version 2.0: Reading from file Migrate.salsa . . .
SALSA Compiler Version 2.0: SALSA program parsed successfully.
SALSA Compiler Version 2.0: SALSA program compiled successfully.
$ javac Migrate.java
$ java Migrate
Usage: java Migrate <uan> <host1> <port1> <host2> <port2>
```

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.

World Migrating Agent Example

```
```

The actor will print "Migrate actor is here." at theater 1 then at theater 2.
module examples.addressbook;

behavior AddressBook implements ActorService {
  Hashtable name2email;

  AddressBook() {
    name2email = new HashTable();
  }

  String getName(String email) {
  ...
  }

  String getEmail(String name) {
  ...
  }

  boolean addUser(String name, String email) {
  ...
  }

  void act(String[] args) {
    if (args.length != 0) {
      standardOutput.println("Usage: salsa -Dactor_uan=<uan>
        -Dactor_host=<host> -Dactor_port=<port>
        examples.addressbook.AddressBook");
    }
  }
}

---

module examples.addressbook;

behavior AddUser {
  void act(String[] args) {
    if (args.length != 3) {
      standardOutput.println("Usage: salsa examples.addressbook.AddUser <BookUAN> <Name> <Email>");
      return;
    }
    AddressBook book = reference AddressBook(new UAN(args[0]));
    book.addUser(args[1], args[2]);
  }
}

---

module examples.addressbook;

behavior GetEmail {
  void act(String[] args) {
    if (args.length != 2) {
      standardOutput.println("Usage: salsa examples.addressbook.GetEmail <BookUAN> <Name>");
      return;
    }
    getEmail(args[0], args[1]);
  }

  void getEmail(String uan, String name) {
    AddressBook book = reference AddressBook(uan);
    standardOutput.println(name + "’s email:");
    standardOutput.println(book.getEmail(name));
  }
}

---

module examples.addressbook;

behavior MigrateBook {
  void act(String[] args) {
    if (args.length != 3) {
      standardOutput.println("Usage: java examples.addressbook.Migrate <BookUAN> <post> <port>");
      return;
    }
    AddressBook book = reference AddressBook(new UAN(args[0]));
    book.migrate(args[1], Integer.parseInt(args[2]));
  }
}

---

Exercises

1. How would you implement the join continuation linguistic abstraction considering different potential distributions of its participating actors?
2. Download and execute the Agent.salsa example.
3. Modify the lock example in the SALSA distribution to include a wait/notify protocol, as opposed to “busy-waiting” (or rather “busy-asking”).
4. VRH Exercise 11.11.3 (pg 746). Implement the example using SALSA/WWC.