Distributed and Mobile Systems with SALSA

AMST does not explicitly model actor locations and mobility. SALSA introduces the notions of:

• *universal actor names*, that enable:
  • transparent actor migration and
  • location-independent communication.

• *theaters*, or Java virtual machines extended to provide an 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, garbage collection.
Universal actor names

Universal actor locations

Message names

Values

Expressions

Actor creation

Reference from UAN

Programs

Non-garbage actors
Distributed Programming with Universal Actors

- SALSA concurrent programs can be directly used for distributed and mobile systems programming.
  - All continuation-passing coordination abstractions have the same semantics.
- SALSA’s run-time system consists of a *name service* and *theaters*, which are Java virtual machines extended with actor creation, migration, and communication services.
- SALSA’s actors can be assigned *Universal Actor Names* (UAN). UANs are strings that serve as unique identifiers to enable location-independent communication with universal actors, mediated by the naming service.
Distributed Programming with Universal Actors

For example, a calendar actor behavior may be written as follows:

```java
behavior Calendar implements ActorService{
    Appointment[] getAppointments(){
        ...
    }
}
```

The ActorService empty interface is used to annotate behaviors whose instances are not to be garbage collected.

- The reason they should not be collected is that it is possible to create references to instances of behaviors implementing this interface from their UANs.
Distributed Programming with Universal Actors

A calendar instance can be created as follows:

```java
Calendar myCalendar = new Calendar() at (uan, host);
```

- `uan` represents the unique name of the calendar actor, which has Uniform Resource Identifier (URI) syntax, for example:
  - `uan://wcl.cs.rpi.edu/~cvarela/calendar`, and

- The host represents the initial location of this actor, which is typically a string with a Domain Name System (DNS) domain name optionally followed by a listening port, for example
  - `t1.wcl.cs.rpi.edu:4040`.

- The naming service (in this example, a daemon running at `wcl.cs.rpi.edu` and listening on default port `3030`) is in charge of checking for uniqueness of the UANs.
Distributed Programming with Universal Actors

Any SALSA program can now obtain references to this universal actor and send messages to it, in the following manner:

```
Calendar calendar = reference uan;
token appointments = calendar <- getAppointments();
```

In SALSA 1.1, this is done as follows:\(^1\)

```
Calendar calendar = (Calendar) Calendar.getReferenceByName(uan);
token appointments = calendar <- getAppointments();
```

\(^1\): the reference syntax is not yet implemented.
Mobile Systems Programming with Universal Actors

Notice that the calendar actor can be migrated at any point to another theater by sending it a `migrate` message:

```plaintext
  calendar <- migrate(newHost);
```
Example: An Applet Server

Example  An applet server can be encoded in SALSA as follows:

```java
behavior Applet {...}

behavior AppletServer implements ActorService {
    ...
    Applet getApplet(String uan, String host){
        return new Applet() at (uan, host);
    }
}
```

For each request, the applet server first creates a new applet actor with the given universal actor name at the given remote host, and then returns its reference to the applet client.
SALSA 1.1 Syntax for UANs and UALs

In SALSA 1.1, Universal Actor Names have the form:

uan://wwc.cs.rpi.edu:3030/cvarela/calendar
uan://wwc.cs.rpi.edu/cvarela/calendar
uan://128.113.126.39/cvarela/calendar

In SALSA 1.1, Universal Actor Locations use the URI or Domain Name (or IP address) syntax, e.g.:

rmsp://wwc.cs.rpi.edu:4040/
rmsp://192.168.1.39:4040/
wwc.cs.rpi.edu:4041
wwc.cs.rpi.edu
192.168.1.39

The port numbers 3030 and 4040 are used by default for names and locations respectively, if no port number is specified.
**SALSA: Programming with Universal Actors**

<table>
<thead>
<tr>
<th>AMST</th>
<th>SALSA</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>new(b)</code></td>
<td><code>new \( B(\mathcal{E}, \ldots, \mathcal{E}) \)</code></td>
<td>Create new actor</td>
</tr>
<tr>
<td></td>
<td><code>new \( B(\mathcal{E}, \ldots, \mathcal{E}) \) \text{ at } (U)</code></td>
<td>[Universal] actor creation</td>
</tr>
<tr>
<td></td>
<td><code>new \( B(\mathcal{E}, \ldots, \mathcal{E}) \) \text{ at } (U, L)</code></td>
<td>[Remote] actor creation</td>
</tr>
</tbody>
</table>

In addition to creating local actors using `new`, you can use `at` to create:

- **universal** actors by giving them a valid UAN, which must be unique—it will not change over the actor’s life-time, and

- **remote** universal actors by also specifying a location where the actor is to start
  - the location must correspond to an Internet site with a theater program listening for incoming actors and messages.
  - the theater’s CLASSPATH must include behavior \( B \).
Reference Cell Service in SALSA

```plaintext
module mcell;
behavior Cell implements ActorService {
    Object content;
    Cell(Object initialContent){
        content = initialContent;
    }
    Object get(){
        standardOutput<-println("Returning cell value:"+content);
        return content;
    }
    void set(Object newContent){
        standardOutput<-println("Setting cell value to:"+
                                newContent);
        content = newContent;
    }
}

• References to standardOutput actor are dynamically bound at the actor’s location—they change as the actor migrates.
```
module mcell;
behavior DistCellTester {
    void act( String[] args ) {
        if (args.length != 2)
            standardError <- println("Usage: salsa mcell.DistCellTester <UAN> <UAL>");
        return;
    }
    Cell c = new Cell(0) at (new UAN(args[0]), new UAL(args[1]));
    standardOutput <- print("Initial Value:") @
        c <- get() @ standardOutput <- println( token );
}

• Creates a distributed cell actor with command-line given name (UAN) and location (UAL).
• A name server and a theater (actor VM) must be running at the appropriate Internet hosts.
Distributed Reference Cell Client

module mcell;

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

• Gets a reference to the distributed cell actor from the command-line given name (UAN). Notice access is location-independent.
Mobile Reference Cell Example

module mcell;
behavior MovingCellTester{
    void act(String args[]){
        if (args.length != 3) {
            standardOutput<-println("Usage: salsa mcell.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<-print("New Value at New Location:") @
        c<-get() @
        standardOutput<-println(token);
    }
}
Mobile Reference Cell Example

The mobile reference cell receives as command-line arguments:

- the cell’s universal actor *name*, which remains the same even as the cell migrates to different hosts,
- the *initial* universal actor *location*, or *host* where the cell is to be remotely created, and
- the *final* universal actor location, or host where the cell is to migrate to.

It *creates* the cell with value ‘‘Hello’’, at the initial location, *accesses* it and *changes* its value to ‘‘World’’; then, it *migrates* the cell to the final location, and *sets* its value to ‘‘New World’’ at its new location.
Address Book Service

```java
module addressbook;
import java.util.*;

behavior AddressBook implements ActorService {
  private 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 -Duan=<UAN> -Dual=<UAL> addressbook.AddressBook" );
      return;
    }
  }
}
```

- uan and ual system property values can be used to give an initial UAN and UAL to the SALSA-created bootstrap actor.
**Address Book Client: Add User**

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

- The 3 command-line arguments are: uan, name, and email.
- The address book’s uan is used to get a reference to the actor.
- A user with the given name and email is added to the address book.
Address Book Client: Get Email

```java
module addressbook;
behavior GetEmail {
    void act(String args[]){
        if (args.length != 2) {
            standardOutput<-println("Usage: salsa addressbook.GetEmail <AddressBookUAN> <Name>" );
            return;
        }
        getEmail(args[0], args[1]);
    }
    void getEmail(String uan, String name){
        try{
            AddressBook book = (AddressBook)
                AddressBook.getReferenceByName(new UAN(uan));
            standardOutput<-print(name + "'s email: ") @
                book<-getEmail(name) @
                standardOutput<-println(token);
        } catch(MalformedUANException e){
            standardOutput<-println(e);
        }
    }
}
```
module addressbook;
behavior MigrateBook {
  void act(String args[]){
    if (args.length != 2) {
      standardOutput<-println(
        "Usage: salsa addressbook.MigrateBook <AddressBookUAN> <NewUAL>"
      );
      return;
    }
    AddressBook book = (AddressBook)
      AddressBook.getReferenceByName(new UAN(args[0]));
    book<-migrate(args[1]);
  }
}

• The command-line arguments are: uan, ual.
• The address book’s uan is used to get a reference to the actor
• the address book is migrated to a new host denoted by ual.