Middleware-Konzepte

Tuple Spaces

Dr. Gero Mühl

Kommunikations- und Betriebssysteme
Fakultät für Elektrotechnik und Informatik
Technische Universität Berlin
Agenda

> Introduction
> Linda Tuple Spaces
> JavaSpaces
Coordination

> **Coordination** is prevalent in many areas
  > Economy, Biology, Psychology, Informatics, in daily life…

> „Coordination is managing dependencies between activities“

„Coordination is the additional information processing performed when multiple, connected actors pursue goals that a single actor pursuing the same goal would not perform.“, (Malone and Crowston)
Linda

> Linda is a so-called *coordination language*
> Developed in the early 80s at the University of Yale
> Application viewed as a collection of cooperating processes
> Eases developing parallel and distributed programs

> **Tuple space** is a multiset of tuples
  > **Tuples** are sequences of *typed fields*
    > E.g., <“Hans”, 42>
  > Processes communicate *only* by writing and reading tuples in and out of the shared tuple space
  ⇒ Space similar to a shared memory or a relational database
Linda (contd.)

> Communication is *indirect* and *anonymous*  
  ⇒ *Space decoupling*
> Also called *generative communication* because the lifetime of a tuple is independent of the process that has written it to the tuplespace ⇒ *Time decoupling*
Tuples Space Operations

> **out**  \(\text{out}(\text{"Peter"}, \text{57})\)
> Puts a tuple into the space; non-blocking

> **eval**  \(\text{eval}(5, f(5))\)
> Similar to **out**
> Creates a concurrent process called **active tuple** that evaluates the function and inserts the resulting (passive) tuple into the space
Tuples Space Operations (contd.)

> **in** 
\[\text{in}(“Peter”, \ ?age)\]
> Retrieves and removes a matching tuple from the space
> Blocks if there is no matching tuple
> Order in which tuples are retrieved is non-deterministic
> A **template**, which consists of **actuals** and **formals**, is used for matching
> A tuple **matches** a template iff
  1. Both have the same number of fields
  2. The actuals in the template equal the values in the tuple
  3. The types of all formals equals the type of the proper values

> **rd** 
\[\text{rd}(“Peter”, \ ?age)\]
> The same as **in** except that the tuple remains in the space
Implementing Message Passing and Semaphores

> Message passing
  > Trivial with *out* and *in*
  > Sender: *out*(receiver_id, sender_id, message)
  > Receiver: *in*(receiver_id, ?sender_id, ?message)

> Semaphores
  > Tuple with one field representing the unique *id* of the semaphore can be used
  > Initialization: *out*(semid)
  > Give: *out*(semid)
  > Take: *in*(semid)
Implementing Remote Procedure Calls (RPCs)

Client

out(P, req, cid, out-actuals)
in (P, rep, cid, ?in-formals)

Server

in (P, req, ?who, ?in-formals)
"body of procedure P"
out(P, rep, who, out-actuals)

> Client and server are decoupled in time and space
> More specific example:

Client

out("Square", req, "me", 10)
in ("Square", rep, "me", ?r)

Server

in("Square", req, ?who, ?x)
r := x*x
out("Square", rep, who, r)
Distributed Data Structures

> Data structures can be accessed by multiple processes simultaneously.
> E.g., an array can be represented as a set of 3-tuples
  > \((name, index, value)\)
  > Elements can be changed in the following way
    > \(in("A", i, ?v)\)
    > \(v := \ldots\)
    > \(out("A", i, v)\)
> A matrix can be represented as a set of 4-tuples
  > \((name, line, column, value)\)
> Access to data items (i.e., tuples) is serialized.
> No concurrent access to individual tuples if \(in/out\) is used but
  *deadlocks* can occur if processes run concurrently.
Linda Implementations

> C, C++, Java, Modula2, Pascal, Ada, Prolog, Fortran, Lisp, Eiffel, Smalltalk, ML ...

> Although programming model is simple, there exist many non-trivial problems, especially with distributed tuple spaces
  > E.g., where are the tuples stored?

> Load balancing

> `eval()` implementation

> Optimizations
Extensions to Linda

- Linda has led to many similar coordination languages
- They often define additional operations
  - `inp()` non-blocking `in();` returns whether tuple was found
  - `rdp()` non-blocking `rd();` returns whether tuple was found
  - `copy-collect(ts1, ts2, template)` copies tuples form one space to another
- Blocking operations with timeouts
- Multiple tuplespaces
- Security mechanisms
- Events
- More expressive matching algorithms
- Fault tolerance
- Transactions
- Persistence
Linda vs. Database

> Linda and databases have much in common
  > Tuple-oriented model
  > Relation very similar to a subspace

> Linda puts emphasize on coordination
  > Simple matching using templates
    > Normally either exact match or “don’t care” for each field
  > Tuples cannot directly be manipulated in the space
  > Processes can be created easily
  > Linda can be implemented on top of a database
Linda vs. Database (contd.)

> Databases emphasize storing, manipulating, and querying data (most common is the relational data model ⇒ RDBMS)
  > Rich query language (e.g., use of join operator, range queries)
  > Tuples can directly be manipulated (update statements)
  > Tuples can be inserted and deleted (insert and delete statements)
  > No native blocking select statements
    > Can possibly be emulated with stored procedures and triggers
  > Many optimizations used
    (query optimization, indexing techniques)
  > Good transaction support (data locking, deadlock detection etc.)
Discussion of Linda

> Simple model $\rightarrow$ easy to learn
> Well suited to implement problems in the area of distributed and parallel computing
> Scalability problems due to the $in()$ operation!
JavaSpaces

> Part of Jini

http://www.sun.com/jini

> Current Specification: Java Spaces2.2
## Linda vs. JavaSpaces

<table>
<thead>
<tr>
<th>Linda</th>
<th>JavaSpaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tuplespace</td>
<td>JavaSpace</td>
</tr>
<tr>
<td>Tuple</td>
<td>Entry</td>
</tr>
<tr>
<td>Formal</td>
<td>Wildcard</td>
</tr>
<tr>
<td>Actual</td>
<td>Value</td>
</tr>
<tr>
<td>out</td>
<td>write</td>
</tr>
<tr>
<td>rd</td>
<td>read</td>
</tr>
<tr>
<td>in</td>
<td>take</td>
</tr>
</tbody>
</table>
Linda vs. JavaSpaces (contd.)

> JavaSpaces have no `eval()` operation
> New operations
  > `readIfExists()` non-blocking `read()`
  > `takeIfExists()` non-blocking `take()`
  > `notify()` listener is notified if matching tuple is inserted
> For blocking operations, a timeout can be specified
> Applications can access multiple spaces simultaneously
Implementation of Entries

> Entries are Java objects implementing the `Entry` interface
>   > All fields must be `public`
>   > Fields can be arbitrary objects but no primitive types
>   > Entry must have an empty constructor
> > Templates are also entries, but some fields may be null

```java
public interface Entry extends java.io.Serializable {
}

public class SharedVar implements Entry {
    public String name;
    public Integer value;
    public SharedVar() {} // Empty constructor
    public SharedVar(String n, Integer v) {
        name = n;
        value = v;
    }
}
```
Entry Matching

> Space is searched for entries that
  > have the same type (or subtype) as the template *and*
  > which fields comply with the fields of the template
> The value of every field in the template which is not *null* must be *equal* to the value of the corresponding field of the entry
> If a value of a field in the template is *null*, the value of the corresponding field of the entry does not matter (*wildcard*)
> Fields are tested for equality using their serialized form (array of bytes)
  > Allows to test fields for equality without requiring a properly implemented `equals()` method
  > Prohibits more expressive tests (e.g., less than)
Entry Matching Details

> Every Entry is converted to an EntryRep
> For every public field, an EntryRep stores the field’s value in a MarshalledObject and its name in a String
> An MarshalledObject essentially contains a field serialized to an byte[]
> Testing two instances of MarshalledObject for equality
  > If the length of the byte arrays are not equal, they are not equal
  > Otherwise, the byte arrays are compared byte by byte

```java
public class EntryRep implements java.io.Serializable {
    // marshalled fields (null == wildcard)
    MarshalledObject[] values;
    // marshalled field names
    String[] names;
    ...
}
```
JavaSpaces Usage Example

```java
JavaSpace space = SpaceAccessor.getSpace();

SharedVar sharedVar = new SharedVar("v1", new Integer(10));
space.write(sharedVar, null, Lease.FOREVER); // no TX

SharedVar template = new SharedVar();
Template.name = "v1";
SharedVar result =
    (SharedVar)space.read(template, null, Long.MAX_VALUE);
Integer value = result.value;

SharedVar result =
    (SharedVar)space.take(template, null, Long.MAX_VALUE);
result.value++;
space.write(result, null, Lease.FOREVER);
```
Using Jini Services: Leasing, Transactions, Events

> JavaSpaces are build upon Jini
  > Space registered as Jini service
  ⇒ Jini lookup service used to lookup space

> **Leasing mechanism**
  > A *lifetime* can be associated with entries
  > Entries must be periodically refreshed
  > Otherwise they are removed from the space

> **Transactions**
  > Multiple operations across multiple spaces can be bundled into an atomic unit of work

> **Distributed events**
  > *notify()*
  > Implemented using Java RMI ⇒ Not very scalable!
Bibliography