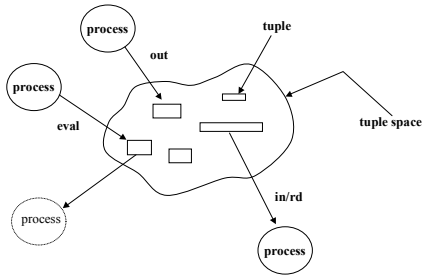
Tuple Space Model

Tuple Space Concepts



Tuple Space Operations

tuple: a series of typed fields
examples: ("label", 10, 2.15)
(5, "term")
(100)

Operations

- **out(t)** insert the tuple t into the tuple space (non-blocking)
- **in(t)** find and remove a "matching" tuple from the tuple space; block until a matching tuple is found
- **rd(t)** like in(t) except that the tuple is not removed
- **eval(t)** add the active tuple t to the tuple space

Tuple Matching

Let t(i) denote the ith field in the tuple t.

A tuple t given in a in(t) or rd(t) operation "matches" a tuple t' in the tuple space iff:

1. t and t' have the same number of fields, and
2. for each field
   if t(i) is a value then t(i) = t'(i)
   or
   if t(i) is of the form ?x then t'(i) is a valid value for the type of variable x

If more than one tuple in the tuple space matches, then one is selected nondeterministically.

As a result of tuple matching if t(i) is of the form ?x, then x := t'(i)

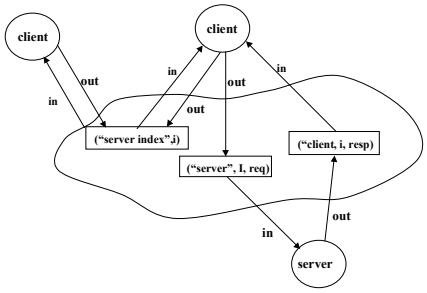
Examples of Tuple Matching

The tuple defined by:

int i;
float f;
("label", ? i, ? f, 10)

Matches these:	Does not match any of these:
("label" , 20, 1.5, 10) and i := 20; f:= 1.5;	("label, 20, 1.5)
	("label", 20, 1.5, 10, 2)
	("other", 20, 1.5, 10)
("label", 0, 2.7, 10) and i:=0; f:=2.7	("label, 20, 1.5, 5)
	("label", "20", 1.5, 10)
	("label", 20, "1.5", 10)

Client-Server Example



Client-Server Example

```
server()
{ int index = 1;
  request req;
  response resp;
  . . .
  while(1) {
    in("server", index, ? req);
    //compute resp
    out("client", index, resp);
    index = index + 1;
  }
}
```

```
client()
{ int index;
  request req;
  response resp;
  . . .
  in("server index", ?index);
  out("server index:, index+1);
  . . .
  out("server", index, req);
  in("client", index, resp);
}
```


Uses of Tuple Spaces

As a coordination language: added to existing programming languages to facilitate distributed and parallel programming

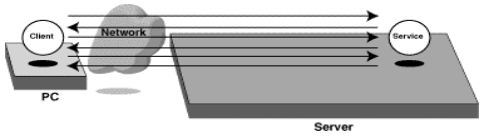
As a distributed registry of names, events, information among loosely coupled processes

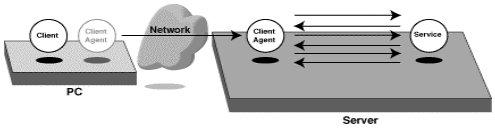
Agent Model

Characteristics of Mobile Agents

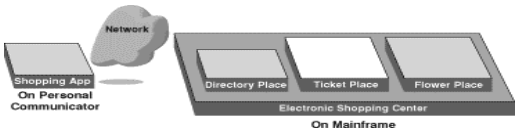
- **Encapsulated**: code, data, itinerary, activity, etc.
- **Autonomous**: decisions on what to do, where to go and when to go.
- **Asynchronous**: has its own thread of execution
- **Local interaction** : with other mobile agents or stationary objects locally.
- **Disconnected operation**: in the absence of a network connection
- **Parallel execution**: among agent dispatched to different sites

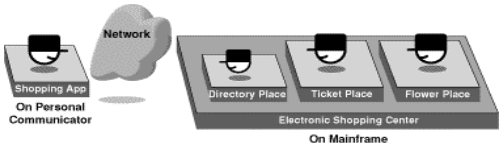
Mobile Agents



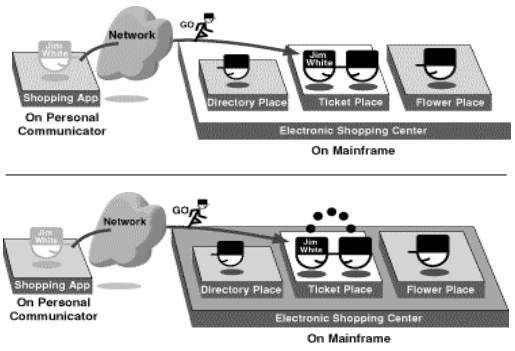


Places and Agents

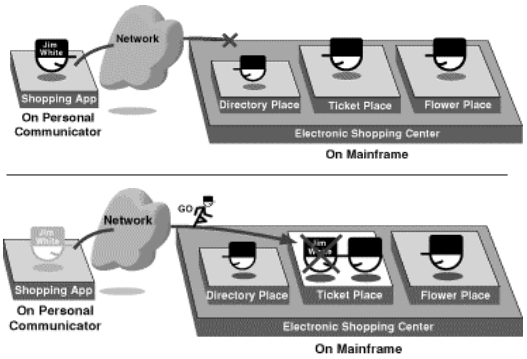




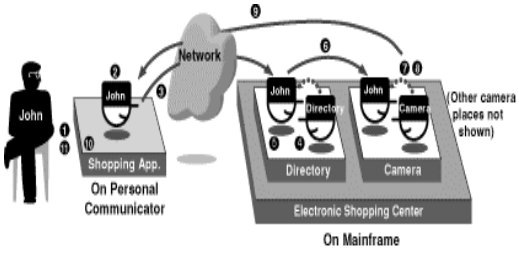
Travel and Meeting



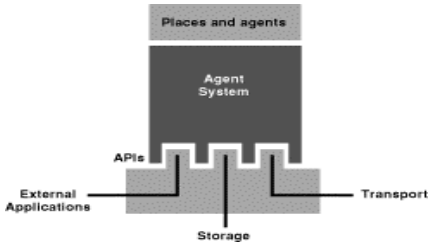
Authorization and Permission



Example



Architecture



Lange's Seven Good Reasons for Using Mobile Agents

- reduce network load
- overcome network latency
- encapsulate protocols
- execute asynchronously and autonomously
- adapt dynamically
- naturally heterogeneous
- robust and fault-tolerant