Tuple Spaces

George Coulouris, Jean Dollimore, Tim Kindberg and Gordon Blair , "**Distributed Systems Concepts and Design**" Edition 5, © Addison-Wesley 2012

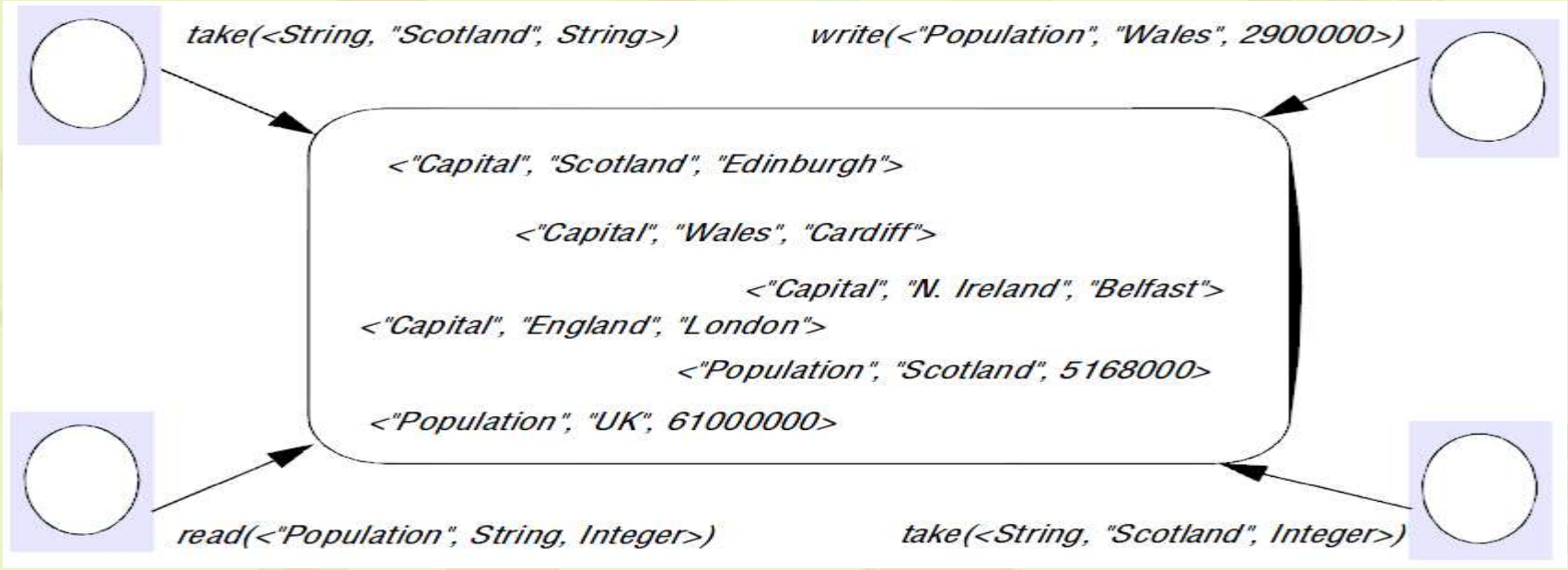
Tuple Spaces

- Processes communicate indirectly by placing tuples in a tuple space, from which other processes can read or remove them.
- Tuples do not have an address but are accessed by pattern matching on content (content-addressable memory)
- Linda programming model has been highly influential and has led developments in distributed programming such as Agora, JavaSpaces from Sun, IBM's Tspaces.
- In the tuple space programming model, processes communicate through a tuple space – a shared collection of tuples

Tuple Spaces - Programming Model

- Tuples in turn consist of a sequence of one or more typed data fields such as <"fred", 1958>, <"sid", 1964> and <4, 9.8, "Yes">.
- Processes share data by accessing the same tuple space
- Operations on Tuple Space : (Write, Read and Take)
- The write operation adds a tuple without affecting existing tuples in
- the space.
- The read operation returns the value of one tuple without affecting the contents of the tuple space.
- The take operation also returns a tuple, but in this case it also removes the tuple from the tuple space

Tuple Spaces - Programming Model



- The *take operation take(<String, "Scotland", String>) will match <"Capital", "Scotland", "Edinburgh">,*
- whereas *take(<String, "Scotland", Integer>) will match <"Population", "Scotland", 5168000>.*
- The *write operation write(<"Population", "Wales, 2900000>) will insert a* new tuple in the tuple space
- *read(<"Population", String, Integer)* can match the equivalent tuples for the populations of the UK, Scotland or indeed Wales

Tuple Spaces - Programming Model

- To enable processes to synchronize their activities, the read and take operations both block until there is a matching tuple in the tuple space.
- A tuple specification includes the number of fields and the required values or types of the fields.
- For example, *take(<String, integer>) could* extract either *<"fred", 1958> or <"sid", 1964>;*
- *take(<String, 1958>) would extract only <"fred", 1958> of those two.*
- No direct access to tuples in tuple space is allowed and processes have to replace tuples in the tuple space instead of modifying them.
- Thus, tuples are *immutable.*

Tuple Spaces - Programming Model

- A set of processes maintains a shared counter in tuple space.
- The current count (say, 64) is in the tuple *<"counter", 64>.*

Tuple space name is *myTS:*

To Increment counter:

- *<s, count> := myTS.take(<"counter", integer>);*
- *myTS.write(<"counter", count+1>);*

Properties of Tuple Spaces

- *Space uncoupling:* A tuple placed in tuple space may originate from any number of sender processes and may be delivered to any one of a number of potential recipients.
- This property is also referred to as distributed naming in Linda.
- *Time uncoupling:* A tuple placed in tuple space will remain in that tuple space until removed (potentially indefinitely), and hence the sender and receiver do not need to overlap in time

Replication In Tuple Spaces

write

1. The requesting site multicasts the *write* request to all members of the view;
2. On receiving this request, members insert the tuple into their replica and acknowledge this action;
3. Step 1 is repeated until all acknowledgements are received.

read

1. The requesting site multicasts the *read* request to all members of the view;
2. On receiving this request, a member returns a matching tuple to the requestor;
3. The requestor returns the first matching tuple received as the result of the operation (ignoring others);
4. Step 1 is repeated until at least one response is received.

take

Phase 1: Selecting the tuple to be removed

1. The requesting site multicasts the *take* request to all members of the view;
2. On receiving this request, each replica acquires a lock on the associated tuple set and, if the lock cannot be acquired, the *take* request is rejected;
3. All accepting members reply with the set of all matching tuples;
4. Step 1 is repeated until all sites have accepted the request and responded with their set of tuples and the intersection is non-null;
5. A particular tuple is selected as the result of the operation (selected randomly from the intersection of all the replies);
6. If only a minority accept the request, this minority are asked to release their locks and phase 1 repeats.

Phase 2: Removing the selected tuple

1. The requesting site multicasts a *remove* request to all members of the view citing the tuple to be removed;
2. On receiving this request, members remove the tuple from their replica, send an acknowledgement and release the lock;
3. Step 1 is repeated until all acknowledgements are received.

Summary of Indirect Communication

	Groups	*Publish-subscribe systems*	*Message queues*	*DSM*	*Tuple spaces*
Space-uncoupled	Yes	Yes	Yes	Yes	Yes
Time-uncoupled	Possible	Possible	Yes	Yes	Yes
Style of service	Communication-based	Communication-based	Communication-based	State-based	State-based
Communication pattern	1-to-many	1-to-many	1-to-1	1-to-many	1-1 or 1-to-many
Main intent	Reliable distributed computing	Information dissemination or EAI; mobile and ubiquitous systems	Information dissemination or EAI; commercial transaction processing	Parallel and distributed computation	Parallel and distributed computation; mobile and ubiquitous systems
Scalability	Limited	Possible	Possible	Limited	Limited
Associative	No	Content-based publish-subscribe only	No	No	Yes


Thank You
fppt.com