Usually over a distributed system, certain resources such as printers are shared over a network. Many of these resources require that only one or a limited number of users can be simultaneously accessing the resource. In this project, we are trying to use the tuple spaces concept to handle this situation. This project aims to develop a distributed tuple space, which would be manipulated by mobile agents for locating/allocating resources.

Each host would have a part of the distributed tuple space. When a machine needs a resource, it sends a mobile agent to look for the resource, if its not found in tuple space in the local machine. The mobile agent goes to every host and looks for an available resource (represented by a tuple). When it finds the matching tuple, the agent reports back to the node, which requested the resource with the address of the resource, and other details required. While migrating back, the agent consumes the tuple representing the resource so that it is no longer free. Once the resource has been used, it can be released by releasing the appropriate tuple into the tuple space again.

This tuple space system has been implemented using JINI JavaSpaces. The JINI Javaspaces was found to be suited for implementing the tuple space system that was proposed. We used Java Object serialization and Sockets for developing the mobile agent part of the system. By developing this system our objective was to learn about the tuple spaces and mobile agent implementation.
**Project Description:**

<table>
<thead>
<tr>
<th>Terms Used</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java Space</td>
<td>Tuple Space. Since Java space is used, the tuple space will be referred to as Java Space</td>
</tr>
<tr>
<td>Entry</td>
<td>Tuple</td>
</tr>
<tr>
<td>Resource</td>
<td>The tuple representing the resource.</td>
</tr>
<tr>
<td>Waiting Query</td>
<td>The query tuple written into the space if the matching resource is not found</td>
</tr>
<tr>
<td>Take operation</td>
<td>Remove the tuple from the space</td>
</tr>
</tbody>
</table>

**Entries Used by the System:**

**Resource**
The *Resource* Entry contains the following:
- Type of the resource. E.g. Printer, CDWriter
- Name of the resource. E.g. HPDeskJet, Beenut.
- Address of the Machine to which the resource belongs.

**Waiting Query**
The *Waiting Query* Entry contains the following:
- The Resource requested
- Timestamp in which the resource requested was not found in the local space where the client resides.

**Components of the System:**

**Tuple Space Manager (Class TSM)**
The *tuple space manager* is the front end of the system, which the client sees. The TSM aggregates within itself the components for the communication with the clients, managing the actual tuple space, allocating process space and communication with TSM’s at the other hosts. The actual components of the TSM are
- Java Space manager – manages the java space.
- Agent handler - component meant to give a process space for the execution of the mobile agent
- Mobile Agent Receiver – component which receives mobile agents on a standard port and passes it on to the TSM
- Request Receiver – component that receives requests from the clients.
- Receiver – component which receives messages from the TSM in the other machines

**Java Space Manager (Class JSM)**
The *Java space manager* manages the Java Space present in the local machine. It is used by the TSM for the following:
- To find and load the Java Space running in the local machine when the name of the Java Space running is given.
- To find the resource in the local space.
• To write the WaitingQuery into the space if the Resource searched is not found.
• To remove the Resource matching a timestamp when the TSM receives a multicast message

**Space Writer (Class SpaceWriter)**

The *Space Writer* finds the Java space running in the local machine (when the name of the java space is given) and writes some Resources into that space.

**Agent Handler (Class AgentHandler)**

The *Agent handler* is the component used by the TSM to give some process space to the mobile agent without sacrificing its own process space. The Agent handler accepts the mobile interface object from the TSM and invokes the resume execution method of the mobile agent. Once the mobile agent finishes execution, the Agent Handler is automatically removed. A new agent handler is created for each mobile agent, which comes in.

**Mobile Agent Receiver (Class MAReceiver)**

This is the component of the TSM, which keeps listening on a port for incoming mobile agents. The port on which the receiver keeps listening for mobile agents is hard coded and kept fixed to facilitate/standardize the interface for the TSM, mobile agent coordination. Once a mobile agent receiver has got a mobile agent, it passes it on to the TSM

**Request Receiver (Class ReqReceiver)**

This is the component of the TSM, which keeps listening on a port for incoming requests from the clients. When the receiver gets the request, it is passed onto the TSM. The port on which the receiver keeps listening for requests is hard coded and kept fixed to facilitate/standardize the interface for the TSM, client coordination.

**Receiver (Class Receiver)**

This is the component of the TSM, which keeps listening on a port for incoming messages from the clients. When the receiver gets the request, it is passed onto the TSM. The port on which the receiver keeps listening for requests is hard coded and kept fixed to facilitate/standardize the interface for the TSM, client coordination.

**System Flow**

1. The Tuple Space Manager (TSM) is invoked with the name of the space and the addresses of the TSM's at the other hosts.
2. It starts up the Communication Interface component of the TSM which includes the Request Receiver which receives the request from the clients, the Mobile Agent receiver which receives the mobile agent from
the other TSM’s, and the Receiver which is more a broadcast receiving port which receive information from the other TSM’s.

3. The Java Space Manager (JSM) is given a reference to the local Java Space of the host.

4. When the TSM receives a request object (sent by the client) from the ReqReceiver,
   a. it asks its JSM to search for the resource in its own tuple space. Now there are two scenarios which arise from this
      1. If the local space has the resource, it is taken out and given to the client at the port specified in the request object and the TSM keeps listening for other information
      2. If the local space does not have the resource, then a waiting query is generated and a mobile agent (MA) is created. The MA is given the waiting query and the request as well as the locations of the TSM’s on the other hosts. It is then dispatched to the first TSM on the list

5. When the TSM receives a MA (sent by a TSM) from the MAREceiver then
   a. It allocates an Agent Handler to that MA.
   b. The Agent handler thread is invoked and the mobile agent resumes execution (by the invocation of the ResumeExec function) in the space of the Agent handler.
   c. A check is done to determine if the mobile agent is returning from an operation or if it is still on its mission of looking for the resource tuple.
      1. If the mobile agent is still looking for the resource then the TSM (using the JSM) looks up its own tuple space. If found it is given to the mobile agent which is sent back to its parent else the mobile agent is dispatched to the next TSM in its itinerary after writing a waiting query in its Java Space (the significance of this will be explained later).
      2. If the mobile agent has returned from its task, then depending on the results it either gives back client based on information in the request object the resource information or tells the client that the resource is not available. In case the tuple was found a broadcast is sent to all the other TSM asking them to remove the waiting query based on the timestamp (at present).

6. When the mobile agent receives a message with a timestamp it knows that it has to remove the waiting query with that timestamp and does that with the JSM component.
Summary
Learnt from the Project
Concepts learnt:
• Tuple spaces
• Mobile Agents
• Designing a solution for a distributed system
• Implementing a solution for distributed system
• Message passing protocols

Techniques learnt:
• JINI Java Spaces
• Mobile Agent Implementation
• Java Object Serialization

Extensions
In this existing system, if the resource is not found in any of the remote locations (including the local space), the mobile agent looking for the resource will go back to the parent TSM, where the client resides. The mobile agent will leave a WaitingQuery at all the remote Java spaces where it has completed unsuccessful search for the resource. The parent TSM will inform the client about the unavailability of the resource, after the agent comes back. The actual system was designed to include a delivery agent, which will follow up this sequence of finding/allocating the resource to the client. Whenever a client completes using a resource, it will leave back the resource into the space where it belongs (can be simulated using the SpaceWriter). When this resource comes back to the space, if there are any WaitingQuery(s) for this resource, a delivery agent will be invoked, which will visit all the TSMs whose client are waiting for the resource. The delivery agent will visit all the hosts based on the order of the timestamp found in the WaitingQuerys for that resource in the local space. When a client receives the resource, the waiting query left for that resource in all the TSMs would be removed from their spaces using a broadcast message from the TSM where the client resides. Implementing this would result in realizing a transparent distributed tuple space, which would enable allocating remote/local resources, if available and if not, providing the capability for delivering the resource to the client when the resource becomes available.