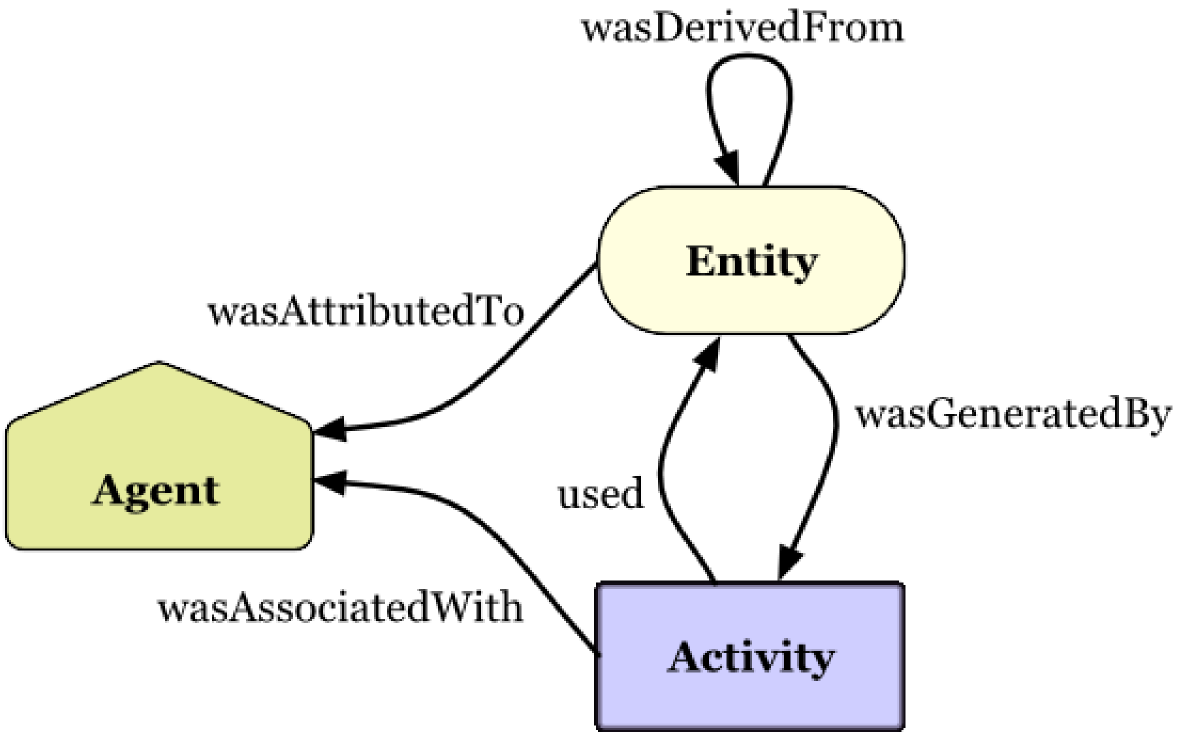
InDiPROV service tutorial

Virtual Reality Group

RWTH-Aachen

Intuitive overview of PROV



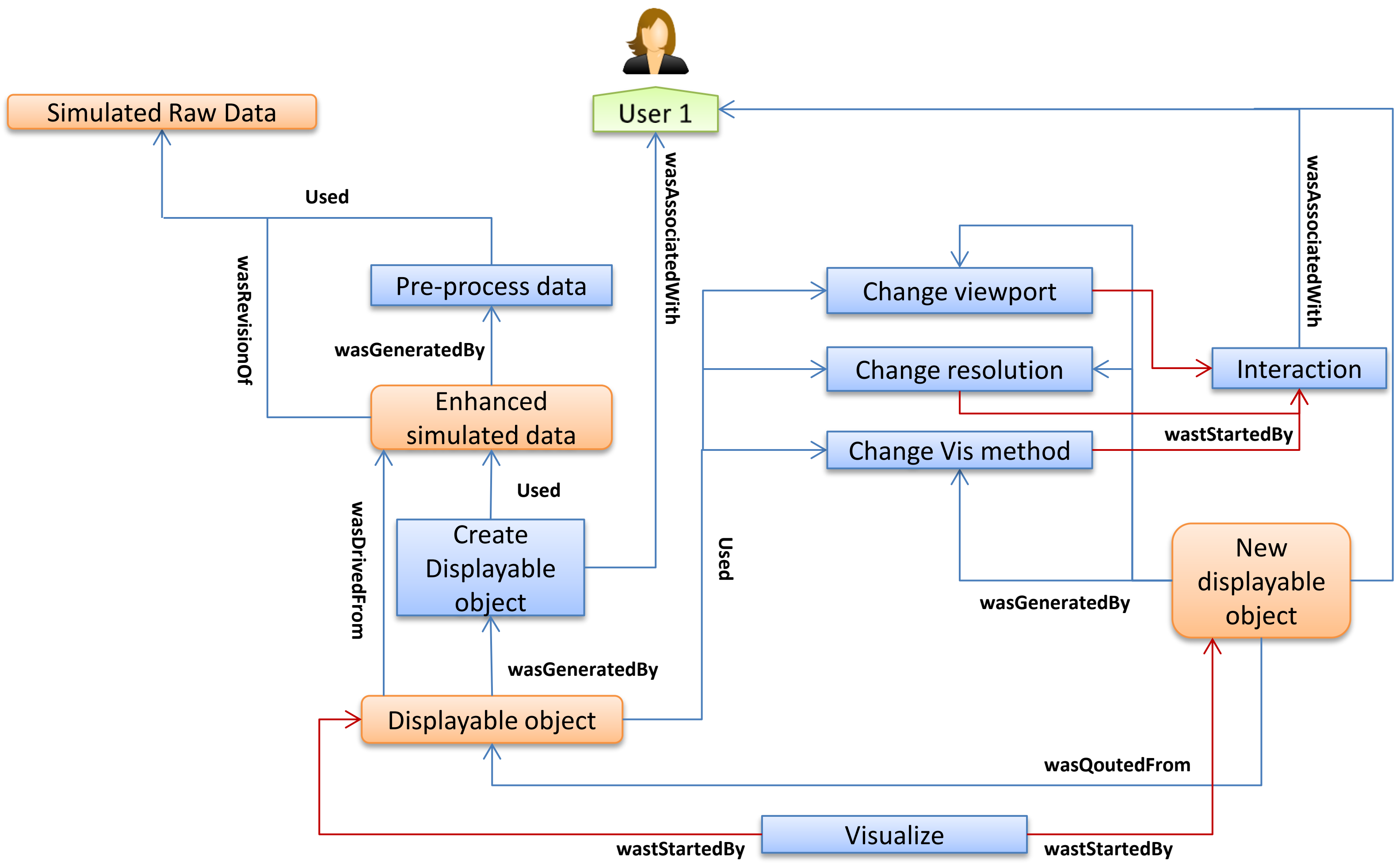
Terminology

- Entity
  - An entity is a physical, digital, conceptual, or other kind of thing with some fixed aspects; entities may be real or imaginary.
- Activity:
  - An activity is something that occurs over a period of time and acts upon or with entities; it may include consuming, processing, transforming, modifying, relocating, using, or generating entities.
- Agent (Person, Organization, Software agent)
  - *An agent is something that bears some form of responsibility for an activity taking place, for the existence of an entity, or for another agent's activity.*
- Generation
  - Generation is the completion of production of a new entity by an activity. This entity did not exist before generation and becomes available for usage after this generation.
- Usage
  - Usage is the beginning of utilizing an entity by an activity. Before usage, the activity had not begun to utilize this entity and could not have been affected by the entity.

Terminology

- Derivation (Revision, Quotation, Primary Source)
  - *A derivation is a transformation of an entity into another, an update of an entity resulting in a new one, or the construction of a new entity based on a pre-existing entity.*
- Attribution
  - *Attribution is the ascribing of an entity to an agent.*
- *Association*
  - *An activity association is an assignment of responsibility to an agent for an activity, indicating that the agent had a role in the activity. It further allows for a plan to be specified, which is the plan intended by the agent to achieve some goals in the context of this activity.*
- *Communication*
  - *Communication is the exchange of some unspecified entity by two activities, one activity using some entity generated by the other.*
- Start
  - *Start is when an activity is deemed to have been started by an entity, known as trigger. The activity did not exist before its start. Any usage, generation, or invalidation involving an activity follows the activity's start. A start may refer to a trigger entity that set off the activity, or to an activity, known as starter, that generated the trigger.*

Scenario 1: Interactive visualization of simulated data



Scenario 1: Using InDiProv service

Initialize / load Work Flow (WF)

```
int wfid= provclient->createWF(“WorkFlowName","WFPassword");
if (provclient->loadWF(wfid , wfpass)==wfid)
```

Entity : label, location, type, value

```
int entityID= provclient->setEntity("Simulated_Raw_Data“ ,“../path/to/file/raw.dat",
        “stored_data",“Null");
```

Activity: startTime, endTime, label, location, type

```
int activityID= provclient->setActivity(“18:30:31.1234 13.07.2015“ , “18:30:51.1234 13.07.2015” ,
        "Pre-process data",“../code/visualization.py",“Computer process");
```

Agent: label, location, type

```
int agentID= provclient->setAgent(“user_1",“VR-Aachen",“Human");
```

Usage: actID, entID, usedTime, label, location, role, type

```
int usedID= provclient->setused(activityID, entityID, “18:30:35.1234 13.07.2015”,"UsedLabel“,
        "UsedLocation", "UsedRole","UsedType");
```

Generation: entID, actID, generateTime, label, location, role, type

```
int wasGeneratedByID= provclient->setwasGeneratedBy(entityID, activityID, “18:30:48.1234
        13.07.2015”,“generationLabel","generationLocation", "generationRole","generationType");
```

Scenario 1: Using InDiProv service

Derivation: genEntID, usdEntID, actID, genID, usgID, label, type

```
int wasDerivedFromID = provclient->setwasDerivedFrom (entityID, entityID, activityID,
            wasGeneratedByID, usedID, "wasDerivedFromLabel","wasDerivedFromType");
```

Attribution: entID, agentID, label, type

```
int wasAttributedToID= provclient->setwasAttributedTo (entityID, agentID, "wasAttributedToLabel",
            "wasAttributedToType")
```

Association: actID, agentID, planID, label, role, type

```
int wasAssociatedWithID= provclient->setwasAssociatedWith(activityID, agentID,-1, "
            wasAssociatedWithLabel", "wasAssociatedWithRole","wasAssociatedWithType");
```

Communication : informed, informant, label, type

```
int wasInformedByID= provclient->setwasInformedBy (activityID1, activityID2, "informing other
            clients","wasInformedByType");
```

Start : actID, triggerID, starterID, time, label, location, role, type

started by entity

```
int wasStartedByID= provclient->wasStartedBy(activityID, triggerID, -1 ,"18:31:20.1234
            13.07.2015","StartedByLabel", "StartedByLocation", "StartedByRole", "StartedByType");
```

Or started by another activity

```
int wasStartedByID= provclient->wasStartedBy(activityID, -1, starterID ,"18:31:20.1234
            13.07.2015","StartedByLabel", "StartedByLocation", "StartedByRole", "StartedByType");
```

Scenario 1: Retrieving provenance data

Get information in JSON format

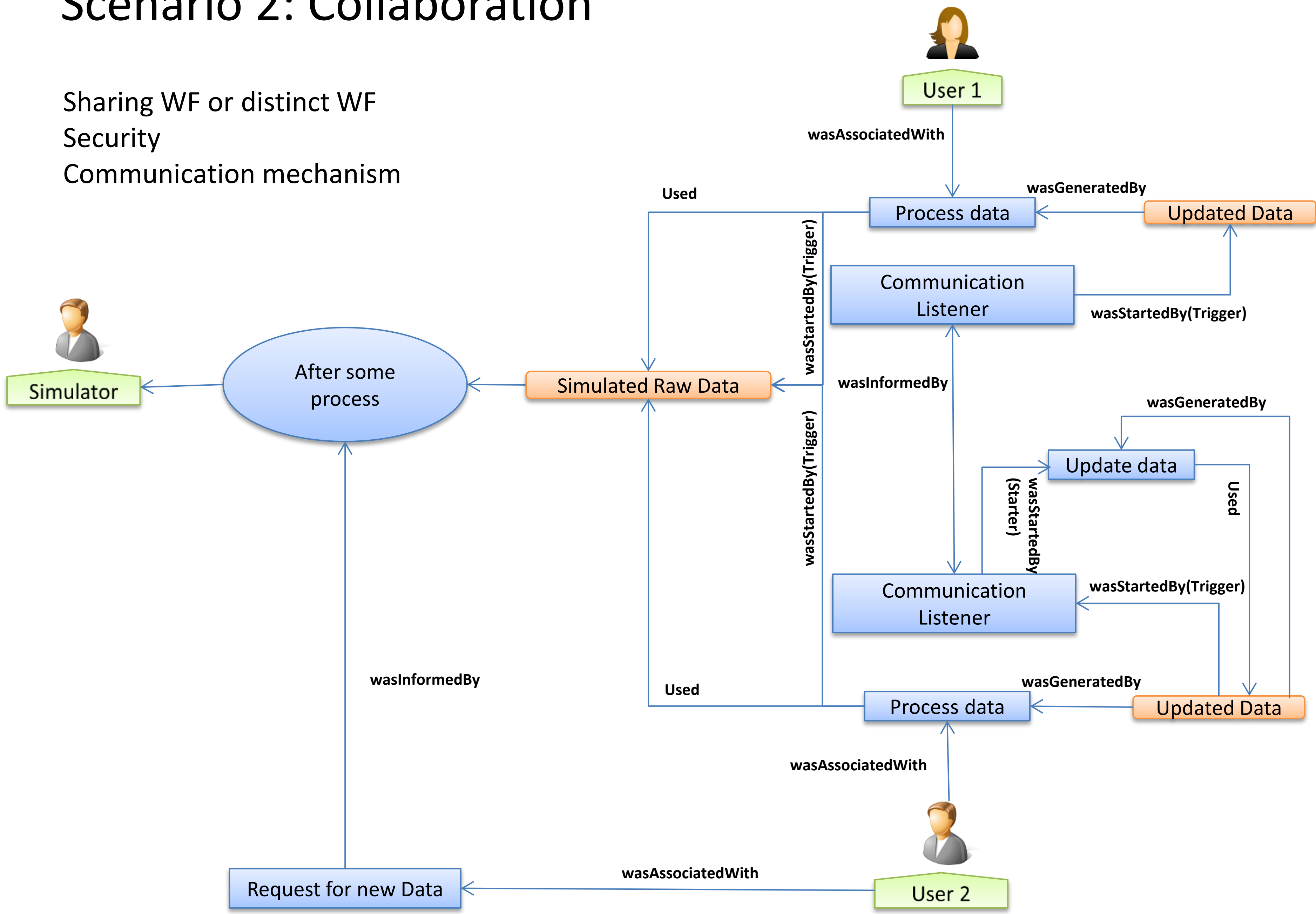
```
string getEntities();
string getActivities();
string getAgents();
string getUseds();
string getWasGeneratedBys();
string getWasDerivedFroms();
string getWasAttributedTos();
string getWasAssociatedWiths();
```

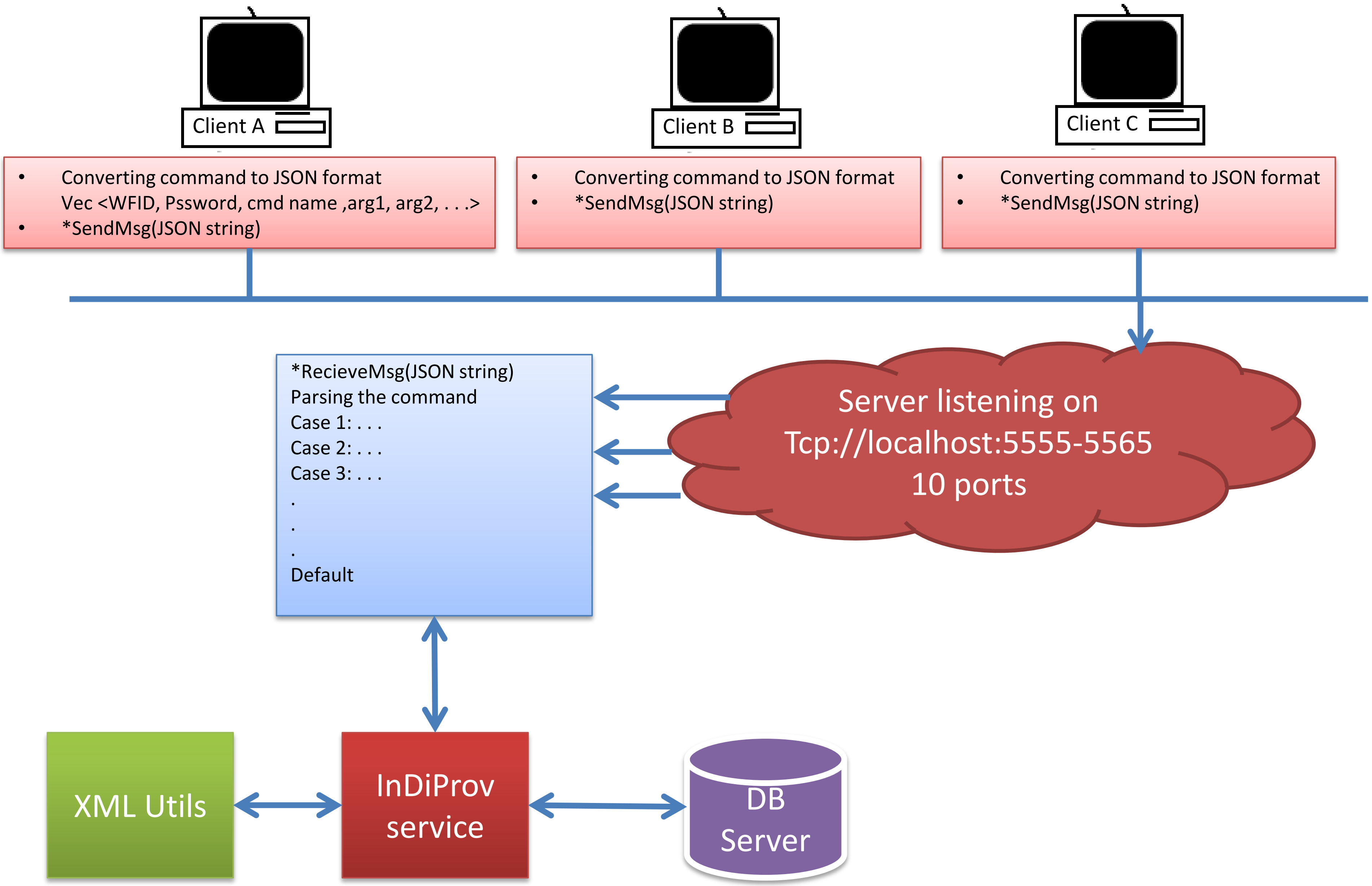
Deserialization and vectorization

```
vector<ProvUtils::Entity> deSerializeEntities(char *entitiesStr);
vector<ProvUtils::Activity> deSerializeActivities(char *activitiesStr);
vector<ProvUtils::Agent> deSerializeAgents(char *agentsStr);
vector<ProvUtils::Used> deSerializeUseds(char *usedsStr);
vector<ProvUtils::WasGeneratedBy> deSerializeWasGeneratedBys(char *wasGeneratedBysStr);
vector<ProvUtils::WasDerivedFrom> deSerializeWasDerivedFroms(char *wasDerivedFromsStr);
vector<ProvUtils::WasAttributedTo> deSerializeWasAttributedTos(char *wasAttributedToStr);
vector<ProvUtils::WasAssociatedWith> deSerializeWasAssociatedWiths(char *wasAssociatedWithsStr);
```

Scenario 2: Collaboration

Sharing WF or distinct WF
Security
Communication mechanism




Client A
Client B
Client C
• Converting command to JSON format
Vec <WFID, Pssword, cmd name ,arg1, arg2, . . .>
• *SendMsg(JSON string)
• Converting command to JSON format
• *SendMsg(JSON string)
• Converting command to JSON format
• *SendMsg(JSON string)
*RecieveMsg(JSON string)
Parsing the command
Case 1: . . .
Case 2: . . .
Case 3: . . .
.
.
.
Default
Server listening on
Tcp://localhost:5555-5565
10 ports
XML Utils
InDiProv service
DB Server