Configuration

SERVER refers to any class of elements where this operation is defined:

- getId: SERVER $\to \mathbb{N}$.

We impose the restriction that this function is inyective.

ENTITY refers to any class of elements where this operation is defined:

- getId: ENTITY $\to \mathbb{N}$.

We impose the restriction that this function is inyective.

Elements of type FEL are subsets of EVENT and they are operated as priority queues. The priority of a given event is given by it's time of ocurrence and a total relation defined over event types:

- insertEvent: FEL $\to$ (EVENT $\to$ FEL).
- deleteIncoming: FEL $\to$ FEL.
- peekIncoming: FEL $\to$ EVENT.

Elements of type CHANNEL are subsets of $\mathbb{N} \times$ ENTITY and are operated as priority queues:

- insertEntity: CHANNEL $\to (\mathbb{N} \to$ (ELEMENT $\to$ CHANNEL)).
- peekTopEntity: CHANNEL $\to$ ELEMENT.
- deleteTopEntity: CHANNEL $\to$ CHANNEL.

A configuration is a tuple

(Servers, Entities, Channels, isServing, waits, listens, L)

where:

- Servers is a collection of elements of type SERVER.
- Entities is a collection of elements of type ENTITY.
- Channels is a collection of elements of type CHANNEL.
- isServing $\subseteq$ Servers $\times$ Entities is the relationship that indicates which entity is being served by which server.
- waits $\subseteq$ Entities $\times$ Channels is the relationship that indicates in which channel an entity is waiting.
- attends $\subseteq$ Servers $\times$ Channels is the relationship that indicates which channels a given server is currently listening.
- L is an element of type FEL.

Rules of Transition

It would be desirable that each user could decide how the system evolves over time according to the diferent events that occur. This is why we introduce rules of transition, which are made up of two parts: The first part is a first order formula that indicates the conditions that a configuration must satisfy before applying the transition. The second part defines the new configuration of the system after the application. Thus, we can see a rule of transistion as a partial function `Rule`: `CONFIGURATION` $\rightarrow$ `CONFIGURATION`. We will now show examples of such rules for a given system.

Example of a traditional system

We will consider a traditional system made up of n servers and one channel, and where the only events that we consider are arribals and exits. The initial configuration of this system is:

$$\langle \{s_1, \ldots, s_n\}, \emptyset, \{Q\}, \{\texttt{ARRIBAL}, \texttt{OUT}\}, \emptyset, \emptyset, \{(s_1, Q), \ldots, (s_n, Q)\}, \emptyset \rangle$$

$$\frac{\texttt{isEmpty}(L) = \texttt{TRUE} \qquad e\colon \texttt{ENTITY} \land e \notin \texttt{Entities}}{\texttt{Entities}' = \texttt{Entities} \cup \{e\} \land L' = L \cup \{(\texttt{ARRIBAL}, k, e)\}}$$

1 Simulation Engine

Algorithm 1 Simulation Step

```
step(out OpResult res)
{
    Event incomingEv;
    if(!fel.isEmpty())
    {
        ev = fel.getIncoming();
        res = ev.applyChanges(configuration, statistics);
    }
    else
        res = STEP_FAILURE_EMPTYFEL;
    return res;
}
```

2 Statistics Computer

2.1 Algorithms

Algorithm 2 Report of a new entity to the statistics computer

```
reportNewArribal(in IDType id, in int tarr,
                                out ReportResult res)
{
    DSResult dsRes;
    if(simTime <= tarr){
        dsRes = entsDS.insert(id, tarr);
        if(dsRes == SUCCESS){
            totalEnts += 1;
            curEnts += 1;
            simTime = tarr;
            if(maxEnts < curEnts)
                maxEnts = curEnts;
            res = REPORT_SUCCESS;
        }
        else if(dsRes == FAILURE_SPACE)
            res = REPORT_FAILURE_SPACE;
        else
            res = REPORT_FAILURE_REPEATED;
    }
    else
        res = REPORT_FAILURE_TIME;
    return res;
}
```

Algorithm 3 Report of the beginning of a new service to the statistics computer

```
reportNewService(in IDType id, in int tServ,
                                out ReportResult res)
{
    DSResult dsRes;
    int tArr;
    int tWait;
    int tIddle;
    if(simTime <= tServ){
        dsRes = entsDS.getArribal(id, tArr);
        if(dsRes == SUCCESS){
            simTime = tServ;
            curQueueSize -= 1;
            tIddle = tServ - tLast;
            tWait = tServ - tArr;
            if(twait < minWait)
                minWait = tWait;
            if(maxWait < tWait)
                maxWait = tWait;
            totalWait += tWait;
            if(tIddle < minIddle)
                minIddle = tIddle;
            if(maxIddle < tIddle)
                maxIddle = tIddle;
            totalIddle += tIddle;
            res = REPORT_SUCCESS;
        }
        else
            res = REPORT_FAILURE_NOT_FOUND;
    }
    else
        res = REPORT_FAILURE_TIME;
    return res;
}
```

Algorithm 4 Report the entrance of an entity to a queue

```
reportNewEntrance(in IDType id, in int tent,
                                out ReportResult res)
{
    DSResult dsRes;
    if(simTime <= tent){
        dsRes = entsDS.belongs(id);
        if(dsRes == SUCCESS){
            simTime = tent;
            curQueueSize += 1;
            if(maxQueueSize < curQueueSize)
                maxQueueSize = curQueueSize;
            res = REPORT_SUCCESS;
        }
        else
            res = REPORT_FAILURE_NOT_FOUND;
    }
    else
        res = REPORT_FAILURE_TIME;
    return res;
}
```

Algorithm 5 Report the exit of an entity from the system

```
reportNewExit(in IDType id, in int tExt,
                                out ReportResult res)
{
    DSResult dsRes;
    int tTran;
    int tArr
    if(simTime <= tExt){
        dsRes = entsDS.getArribal(id, tArr);
        if(dsRes == SUCCESS){
            entsDS.delete(id);
            simTime = tExt;
            tLast = text;
            curEnts -= 1;
            tTran = tExt - tArr;
            if(tTran < minTran)
                minTran = tTran;
            if(maxTran < tTran)
                maxTran = tTran;
            totalTran += tTran;
            res = REPORT_SUCCESS;
        }
        else
            res = REPORT_FAILURE_NOT_FOUND;
    }
    else
        res = REPORT_FAILURE_TIME;
    return res;
}
```

Channel Implementation

```
addEntity(in Entity e, out ChannelResult res)
{
    DSResult resQ;
    DSResult resW;
    PtrType ptr;

    resW = isWaitingDS.locate(e.getID());
    if(resW == NOT_FOUND){
        resQ = queue.push(e, ptr);
        if(resQ == SUCCESS){
            resW = resW.insert(e.getID(), e, ptr);
            if(resW == SUCCESS)
                res = SUCCESS;
            else
                res = FAILURE_NO_SPACE;

        }
        else
            res = FAILURE_NO_SPACE
    }
    else
        res = FAILURE_REPEATED
    return res;
}
```