

Progetto di Simulazione di Sistemi

Samuele Evangelisti

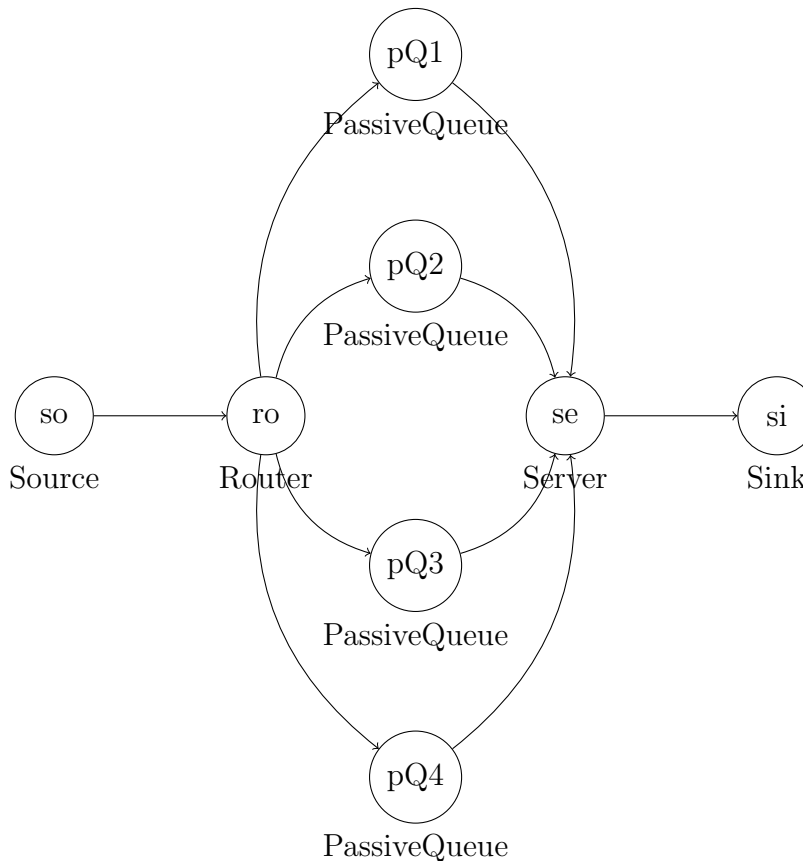
a.a. 2019/2020

Contents

1	Modello	2
1.1	Rete	2
1.1.1	network.ned	2
1.2	Configurazione	3
1.2.1	omnetpp.ini	3
A	Codice Sorgente	5
A.1	Job (pssqueueinglib)	5
A.1.1	Job.h	5
A.1.2	Job.cc	7
A.2	Source (pssqueueinglib)	9
A.2.1	Source.ned	9
A.2.2	Source.cc	10
A.3	Router (pssqueueinglib)	12
A.3.1	Router.ned	12
A.3.2	Router.h	13
A.3.3	Router.cc	13
A.4	SelectionStrategies (pssqueueinglib)	15
A.4.1	SelectionStrategies.h	15
A.4.2	SelectionStrategies.cc	17
A.5	Server (pssqueueinglib)	22
A.5.1	Server.ned	22
A.5.2	Server.h	23
A.5.3	Server.cc	24

1 Modello

1.1 Rete



1.1.1 network.ned

```
//  
// This program is free software: you can redistribute it and/or modify  
// it under the terms of the GNU Lesser General Public License as published by  
// the Free Software Foundation, either version 3 of the License, or  
// (at your option) any later version.  
//  
// This program is distributed in the hope that it will be useful,  
// but WITHOUT ANY WARRANTY; without even the implied warranty of  
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
// GNU Lesser General Public License for more details.  
//  
// You should have received a copy of the GNU Lesser General Public License  
// along with this program. If not, see http://www.gnu.org/licenses/.  
//  
import org.omnetpp.queueing.PassiveQueue;  
import org.omnetpp.queueing.Router;  
import org.omnetpp.queueing.Server;  
import org.omnetpp.queueing.Sink;  
import org.omnetpp.queueing.Source;  
  
//  
// TODO documentation  
//
```

```

network Network
{
    @display("bg=677,409");
    submodules:
        source: Source {
            @display("p=53,191");
        }
        router: Router {
            @display("p=173,191");
        }
        passiveQueue: PassiveQueue {
            @display("p=324,63");
        }
        passiveQueue1: PassiveQueue {
            @display("p=321,152");
        }
        passiveQueue2: PassiveQueue {
            @display("p=321,238");
        }
        passiveQueue3: PassiveQueue {
            @display("p=321,325");
        }
        server: Server {
            @display("p=479,191");
        }
        sink: Sink {
            @display("p=613,191");
        }
    connections:
        source.out --> router.in++;
        router.out++ --> passiveQueue.in++;
        router.out++ --> passiveQueue1.in++;
        router.out++ --> passiveQueue2.in++;
        router.out++ --> passiveQueue3.in++;
        passiveQueue.out++ --> server.in++;
        passiveQueue1.out++ --> server.in++;
        passiveQueue2.out++ --> server.in++;
        passiveQueue3.out++ --> server.in++;
        server.out --> sink.in++;
}

```

Listing 1: "network.ned"

1.2 Configurazione

1.2.1 omnetpp.ini

```

[General]
network = Network

record-eventlog = true
repeat = 20
seed-set = 33
sim-time-limit = 100s

# Esponenziale di media 1/lambda = {0.5, 0.714285714286, 0.833333333333, 1}; lambda =
{2.0, 1.4, 1.2, 1.0}

```

```

simtime-resolution = ps
Network.source.interArrivalTime = exponential(0.5s)
# Uniforme su [a, b] = {[4.0, 6.0], [3.0, 7.0]}
Network.source.jobRelativeDeadline = uniform(4.0s, 6.0s)

# Code K = {1, 2, 4}
Network.router.queueNumber = 1
Network.router.routingAlgorithm = "pssRandom"

# Esponenziale negativa di media 1/mu = {0.333333333333, 0.25}; mu = {3.0, 4.0}
Network.server.serviceTime = exponential(0.333333333333s)
Network.server.fetchingAlgorithm = "exhaustiveService"
Network.server.checkJobDeadline = true

```

Listing 2: "omnetpp.ini"

A Codice Sorgente

Per l'implementazione delle funzionalità richieste dal modello di simulazione si è proceduto modificando alcune classi e alcuni moduli della libreria *queueinglib*. La nuova libreria è stata chiamata *pssqueueinglib* ed è stata utilizzata nel progetto al posto della libreria *queueinglib*.

Di seguito vengono riportate le modifiche apportate, nei moduli e nelle librerie il codice aggiunto è riportato sotto il commento

```
// progettoss
```

A.1 Job (pssqueueinglib)

Aggiunte:

- *simtime_t absoluteDeadline*: attributo che contiene la deadline assoluta del *Job* (*Job.h*)
- *void setAbsoluteDeadline(simtime_t absoluteDeadline)*: metodo per impostare la deadline assoluta del *Job* (*Job.h*, *Job.cc*)
- *simtime_t getAbsoluteDeadline()*: metodo per ottenere la deadline assoluta del *Job* (*Job.h*, *Job.cc*)

A.1.1 Job.h

```
//  
// This file is part of an OMNeT++/OMNEST simulation example.  
//  
// Copyright (C) 2006–2015 OpenSim Ltd.  
//  
// This file is distributed WITHOUT ANY WARRANTY. See the file  
// 'license' for details on this and other legal matters.  
//  
  
#ifndef __QUEUEING_JOB_H  
#define __QUEUEING_JOB_H  
  
#include <vector>  
#include "Job_m.h"  
  
namespace queueing {  
  
class JobList;  
  
/**  
 * We extend the generated Job_Base class with support for split-join, as well  
 * as the ability to enumerate all jobs in the system.  
 *  
 * To support split-join, Jobs manage parent-child relationships. A  
 * relationship is created with the makeChildOf() or addChild() methods,  
 * and lives until the parent or the child Job is destroyed.  
 * It can be queried with the getParent() and getNumChildren()/getChild(k)  
 * methods.  
 *  
 * To support enumerating all jobs in the system, each Job automatically  
 * registers itself in a JobList module, if one exist in the model.  
 */
```

```

* (If there's no JobList module, no registration takes place.) If there
* are more than one JobList modules, the first one is chosen.
* JobList can also be explicitly specified in the Job constructor.
* The default JobList can be obtained with the JobList::getDefaultInstance()
* method. Then one can query JobList for the set of Jobs currently present.
*/
class QUEUEING_API Job: public Job_Base
{
    friend class JobList;
protected:
    Job *parent;
    std::vector<Job*> children;
    JobList *jobList;
    virtual void setParent(Job *parent); // only for addChild()
    virtual void parentDeleted();
    virtual void childDeleted(Job *child);
    // progettoss
    simtime_t absoluteDeadline;
public:
    /**
     * Creates a job with the given name, message kind, and jobList. If
     * jobList==nullptr, the default one (or none if none exist) will be chosen.
     */
    Job(const char *name=nullptr, int kind=0, JobList *table=nullptr);

    /** Copy constructor */
    Job(const Job& job);

    /** Destructor */
    virtual ~Job();

    /** Duplicates this job */
    virtual Job *dup() const override {return new Job(*this);}

    /** Assignment operator. Does not affect parent, children and jobList. */
    Job& operator=(const Job& job);

    /** @name Parent-child relationships */
    //@{
    /** Returns the parent job. Returns nullptr if there's no parent or it no longer
        exists. */
    virtual Job *getParent();

    /** Returns the number of children. Deleted children are automatically removed
        from this list. */
    virtual int getNumChildren() const;

    /** Returns the kth child. Throws an error if index is out of range. */
    virtual Job *getChild(int k);

    /** Marks the given job as the child of this one. */
    void addChild(Job *child);

    /** Same as addChild(), but has to be invoked on the child job */
    virtual void makeChildOf(Job *parent);
    //@}

    /** Returns the JobList where this job has been registered. */
    JobList *getContainingJobList() {return jobList;}

```



```

        // progettoss
        void setAbsoluteDeadline(simtime_t absoluteDeadline);

        simtime_t getAbsoluteDeadline();

};

}; // namespace

#endif

```

A.1.2 Job.cc

```

//
// This file is part of an OMNeT++/OMNEST simulation example.
//
// Copyright (C) 2006–2015 OpenSim Ltd.
//
// This file is distributed WITHOUT ANY WARRANTY. See the file
// 'license' for details on this and other legal matters.
//

#include <algorithm>
#include "Job.h"
#include "JobList.h"

namespace queueing {

Job::Job(const char *name, int kind, JobList *jobList) : Job_Base(name, kind)
{
    parent = nullptr;
    if (jobList == nullptr && JobList::getDefaultInstance() != nullptr)
        jobList = JobList::getDefaultInstance();
    this->jobList = jobList;
    if (jobList != nullptr)
        jobList->registerJob(this);
}

Job::Job(const Job& job)
{
    setName(job.getName());
    operator=(job);
    parent = nullptr;
    jobList = job.jobList;
    if (jobList != nullptr)
        jobList->registerJob(this);
}

Job::~~Job()
{
    if (parent)
        parent->childDeleted(this);
    for (int i = 0; i < (int)children.size(); i++)
        children[i]->parentDeleted();
    if (jobList != nullptr)
        jobList->deregisterJob(this);
}

```

```

}

Job& Job::operator=(const Job& job)
{
    if (this == &job)
        return *this;
    Job_Base::operator=(job);
    // leave parent and jobList untouched
    return *this;
}

Job *Job::getParent()
{
    return parent;
}

void Job::setParent(Job *parent)
{
    this->parent = parent;
}

int Job::getNumChildren() const
{
    return children.size();
}

Job *Job::getChild(int k)
{
    if (k < 0 || k >= (int)children.size())
        throw cRuntimeError(this, "child_index_%d_out_of_bounds", k);
    return children[k];
}

void Job::makeChildOf(Job *parent)
{
    parent->addChild(this);
}

void Job::addChild(Job *child)
{
    child->setParent(this);
    ASSERT(std::find(children.begin(), children.end(), child) == children.end());
    children.push_back(child);
}

void Job::parentDeleted()
{
    parent = nullptr;
}

void Job::childDeleted(Job *child)
{
    std::vector<Job *>::iterator it = std::find(children.begin(), children.end(), child);
    ASSERT(it != children.end());
    children.erase(it);
}

void Job::setAbsoluteDeadline(simtime_t absoluteDeadline)

```

```

{
    this→absoluteDeadline = absoluteDeadline;
}

simtime_t Job::getAbsoluteDeadline()
{
    return absoluteDeadline;
}

}; // namespace

```

A.2 Source (pssqueueinglib)

Aggiunte:

- *double jobRelativeDeadline @unit(s) = default(0s)*: parametro per impostare la deadline relativa dei *Job* (*Source.ned*)

Modifiche:

- *Job *SourceBase::createJob()*: il *Job* viene configurato con la sua deadline assoluta (*Source.cc*)

A.2.1 Source.ned

```

//
// This file is part of an OMNeT++/OMNEST simulation example.
//
// Copyright (C) 2006–2015 OpenSim Ltd.
//
// This file is distributed WITHOUT ANY WARRANTY. See the file
// 'license' for details on this and other legal matters.
//

package org.omnetpp.queueing;

//
// A module that generates jobs. One can specify the number of jobs to be generated,
// the starting and ending time, and interval between generating jobs.
// Job generation stops when the number of jobs or the end time has been reached,
// whichever occurs first. The name, type and priority of jobs can be set as well.
// One can specify the job relative deadline.
//
simple Source
{
    parameters:
        @group( Queueing );
        @signal[ created ]( type="long" );
        @statistic[ created ]( title="the number of jobs created"; record=last;
            interpolationmode=none );
        @display( "i=block/source" );
        int numJobs = default(-1); // number of jobs to be generated (-1
            means no limit)
        volatile double interArrivalTime @unit(s); // time between generated jobs
        string jobName = default("job"); // the base name of the generated job (
            will be the module name if left empty)
        volatile int jobType = default(0); // the type attribute of the created
            job (used by classifiers and other modules)

```

```

        volatile int jobPriority = default(0); // priority of the job
        double startTime @unit(s) = default(interArrivalTime); // when the module sends
            out the first job
        double stopTime @unit(s) = default(-1s); // when the module stops the job
            generation (-1 means no limit)
        // progettos
        double jobRelativeDeadline @unit(s) = default(0s); // job relative deadline
    gates:
        output out;
}

```

Listing 3: "Source.ned"

A.2.2 Source.cc

```

//
// This file is part of an OMNeT++/OMNEST simulation example.
//
// Copyright (C) 2006–2015 OpenSim Ltd.
//
// This file is distributed WITHOUT ANY WARRANTY. See the file
// 'license' for details on this and other legal matters.
//

#include "Source.h"
#include "Job.h"

namespace queueing {

void SourceBase::initialize()
{
    createdSignal = registerSignal("created");
    jobCounter = 0;
    WATCH(jobCounter);
    jobName = par("jobName").stringValue();
    if (jobName == "")
        jobName = getName();
}

Job *SourceBase::createJob()
{
    char buf[80];
    sprintf(buf, "%.60s-%d", jobName.c_str(), ++jobCounter);
    Job *job = new Job(buf);
    job->setKind(par("jobType"));
    job->setPriority(par("jobPriority"));
    job->setAbsoluteDeadline(simTime() + par("jobRelativeDeadline"));
    return job;
}

void SourceBase::finish()
{
    emit(createdSignal, jobCounter);
}

//——
Define_Module(Source);

```

```

void Source::initialize()
{
    SourceBase::initialize();
    startTime = par("startTime");
    stopTime = par("stopTime");
    numJobs = par("numJobs");

    // schedule the first message timer for start time
    scheduleAt(startTime, new cMessage("newJobTimer"));
}

void Source::handleMessage(cMessage *msg)
{
    ASSERT(msg->isSelfMessage());

    if ((numJobs < 0 || numJobs > jobCounter) && (stopTime < 0 || stopTime > simTime()))
    {
        // reschedule the timer for the next message
        scheduleAt(simTime() + par("interArrivalTime").doubleValue(), msg);

        Job *job = createJob();
        send(job, "out");
    }
    else {
        // finished
        delete msg;
    }
}

//——
Define_Module(SourceOnce);

void SourceOnce::initialize()
{
    SourceBase::initialize();
    simtime_t time = par("time");
    scheduleAt(time, new cMessage("newJobTimer"));
}

void SourceOnce::handleMessage(cMessage *msg)
{
    ASSERT(msg->isSelfMessage());
    delete msg;

    int n = par("numJobs");
    for (int i = 0; i < n; i++) {
        Job *job = createJob();
        send(job, "out");
    }
}

}; //namespace

```

Listing 4: "Source.cc"

A.3 Router (pssqueueinglib)

Aggiunte:

- *int queueNumber = default(sizeof(out)-1)*: numero di code da utilizzare (*Router.ned*)
- *ALG_PSSRANDOM*: algoritmo che consente di inoltrare i messaggi solo alle prime n code in maniera casuale (*Router.h*)
- *int queueNumber*: numero di code da utilizzare (*Router.h*)

Modifiche:

- *string routingAlgorithm @enum("random", "roundRobin", "shortestQueue", "minDelay", "pssRandom") = default("random")*: "pssRandom" permette di inoltrare i messaggi solo alle prime n code in maniera casuale (*Router.ned*)
- *void Router::initialize()*: inizializzazione dell'algoritmo di instradamento e del numero di code da utilizzare (*Router.cc*)
- *void Router::handleMessage(cMessage *msg)*: implementazione dell'algoritmo di instradamento *ALG_PSSRANDOM* (*Router.cc*)

A.3.1 Router.ned

```
//
// This file is part of an OMNeT++/OMNEST simulation example.
//
// Copyright (C) 2006–2015 OpenSim Ltd.
//
// This file is distributed WITHOUT ANY WARRANTY. See the file
// 'license' for details on this and other legal matters.
//

package org.omnetpp.queueing;

//
// Sends the messages to different outputs depending on a set algorithm.
// Sends the messages to first queueNumber-th queues.
//
// @author rhornig, Samuele Evangelisti
// @todo minDelay not implemented
//
simple Router
{
    parameters:
        @group( Queueing );
        @display( "i=block/routing" );
        string routingAlgorithm @enum( "random", "roundRobin", "shortestQueue", "minDelay", "pssRandom" ) = default( "random" );
        volatile int randomGateIndex = default( intuniform( 0, sizeof( out ) - 1 ) ); // the destination gate in case of random routing
        // progetto
        int queueNumber = default( sizeof( out ) - 1 ); // queue number limit
    gates:
        input in [ ];
        output out [ ];
```

```
}
```

Listing 5: "Router.ned"

A.3.2 Router.h

```
//  
// This file is part of an OMNeT++/OMNEST simulation example.  
//  
// Copyright (C) 2006–2015 OpenSim Ltd.  
//  
// This file is distributed WITHOUT ANY WARRANTY. See the file  
// 'license' for details on this and other legal matters.  
//  
  
#ifndef _QUEUEING_ROUTER_H  
#define _QUEUEING_ROUTER_H  
  
#include "QueueingDefs.h"  
  
namespace queueing {  
  
// routing algorithms  
enum {  
    ALG_RANDOM,  
    ALG_ROUND_ROBIN,  
    ALG_MIN_QUEUE_LENGTH,  
    ALG_MIN_DELAY,  
    ALG_MIN_SERVICE_TIME,  
    // progetto  
    ALG_PSSRANDOM  
};  
  
/**  
 * Sends the messages to different outputs depending on a set algorithm.  
 * Sends the messages to first queueNumber-th queues.  
 */  
class QUEUEING_API Router : public cSimpleModule  
{  
    private:  
        int routingAlgorithm; // the algorithm we are using for routing  
        int rrCounter;        // msgCounter for round robin routing  
        // progetto  
        int queueNumber;  
    protected:  
        virtual void initialize() override;  
        virtual void handleMessage(cMessage *msg) override;  
};  
  
}; //namespace  
  
#endif
```

Listing 6: "Router.h"

A.3.3 Router.cc

```

//
// This file is part of an OMNeT++/OMNEST simulation example.
//
// Copyright (C) 2006–2015 OpenSim Ltd.
//
// This file is distributed WITHOUT ANY WARRANTY. See the file
// 'license' for details on this and other legal matters.
//

#include "Router.h"

namespace queueing {

Define_Module(Router);

void Router::initialize()
{
    const char *algName = par("routingAlgorithm");
    if (strcmp(algName, "random") == 0) {
        routingAlgorithm = ALG_RANDOM;
    }
    else if (strcmp(algName, "roundRobin") == 0) {
        routingAlgorithm = ALG_ROUND_ROBIN;
    }
    else if (strcmp(algName, "minQueueLength") == 0) {
        routingAlgorithm = ALG_MIN_QUEUE_LENGTH;
    }
    else if (strcmp(algName, "minDelay") == 0) {
        routingAlgorithm = ALG_MIN_DELAY;
    }
    else if (strcmp(algName, "minServiceTime") == 0) {
        routingAlgorithm = ALG_MIN_SERVICE_TIME;
    }
    else if (strcmp(algName, "pssRandom") == 0) {
        routingAlgorithm = ALG_PSS_RANDOM;
    }
    rrCounter = 0;
    int qn = par("queueNumber").intValue() - 1;
    if (qn < 0 || qn > gateSize("out") - 1)
        throw cRuntimeError("Invalid_queue_number");
    else
        queueNumber = qn;
}

void Router::handleMessage(cMessage *msg)
{
    int outGateIndex = -1; // by default we drop the message

    switch (routingAlgorithm) {
        case ALG_RANDOM:
            outGateIndex = par("randomGateIndex");
            break;

        case ALG_ROUND_ROBIN:
            outGateIndex = rrCounter;
            rrCounter = (rrCounter + 1) % gateSize("out");
            break;
    }
}

```



```

    case ALG_MIN_QUEUE_LENGTH:
        // TODO implementation missing
        outGateIndex = -1;
        break;

    case ALG_MIN_DELAY:
        // TODO implementation missing
        outGateIndex = -1;
        break;

    case ALG_MIN_SERVICE_TIME:
        // TODO implementation missing
        outGateIndex = -1;
        break;

    case ALG_PSSRANDOM:
        outGateIndex = intuniform(0, queueNumber);
        break;

    default:
        outGateIndex = -1;
        break;
}

// send out if the index is legal
if (outGateIndex < 0 || outGateIndex >= gateSize("out"))
    throw cRuntimeError("Invalid output gate selected during routing");

send(msg, "out", outGateIndex);
}

}; //namespace

```

Listing 7: "Router.cc"

A.4 SelectionStrategies (pssqueueinglib)

Aggiunte:

- *class QUEUEING_API ExhaustiveServiceSelectionStrategy : public SelectionStrategy*: implementa la *SelectionStrategy* per ottenere un exhaustive service (*SelectionStrategies.h*, *SelectionStrategies.cc*)

Modifiche:

- *SelectionStrategy *SelectionStrategy::create(const char *algName, cSimpleModule *module, bool selectOnInGate*: inizializzazione di *ExhaustiveServiceSelectionStrategy* (*SelectionStrategies.cc*)

A.4.1 SelectionStrategies.h

```

//
// This file is part of an OMNeT++/OMNEST simulation example.
//
// Copyright (C) 2006–2015 OpenSim Ltd.
//
// This file is distributed WITHOUT ANY WARRANTY. See the file
// 'license' for details on this and other legal matters.

```

```

//

#ifndef _QUEUEING_SELECTIONSTRATEGIES_H
#define _QUEUEING_SELECTIONSTRATEGIES_H

#include "QueueingDefs.h"

namespace queueing {

/**
 * Selection strategies used in queue, server and router classes to decide
 * which module to choose for further interaction.
 */
class QUEUEING_API SelectionStrategy : public cObject
{
protected:
    bool isInputGate;
    int gateSize; // the size of the gate vector
    cModule *hostModule; // the module using the strategy
public:
    // on which module's gates should be used for selection
    // if selectOnInGate is true, then we will use "in" gate otherwise "out" is used
    SelectionStrategy(cSimpleModule *module, bool selectOnInGate);
    virtual ~SelectionStrategy();

    static SelectionStrategy * create(const char *algName, cSimpleModule *module,
        bool selectOnInGate);

    // which gate index the selection strategy selected
    virtual int select() = 0;
    // returns the i-th module's gate which connects to our host module
    cGate *selectableGate(int i);
protected:
    // is this module selectable according to the policy? (queue is selectable if
    // not empty, server is selectable if idle)
    virtual bool isSelectable(cModule *module);
};

/**
 * Priority based selection. The first selectable index will be returned.
 */
class QUEUEING_API PrioritySelectionStrategy : public SelectionStrategy
{
public:
    PrioritySelectionStrategy(cSimpleModule *module, bool selectOnInGate);
    virtual int select() override;
};

/**
 * Random selection from the selectable modules, with uniform distribution.
 */
class QUEUEING_API RandomSelectionStrategy : public SelectionStrategy
{
public:
    RandomSelectionStrategy(cSimpleModule *module, bool selectOnInGate);
    virtual int select() override;
};

/**

```

```

* Uses Round Robin selection, but skips any module that is not available currently.
*/
class QUEUEING_API RoundRobinSelectionStrategy : public SelectionStrategy
{
    protected:
        int lastIndex; // the index of the module last time used
    public:
        RoundRobinSelectionStrategy(cSimpleModule *module, bool selectOnInGate);
        virtual int select() override;
};

/**
* Chooses the shortest queue. If there are more than one
* with the same length, it chooses by priority among them.
* This strategy is for output only (i.e. for router module).
*/
class QUEUEING_API ShortestQueueSelectionStrategy : public SelectionStrategy
{
    public:
        ShortestQueueSelectionStrategy(cSimpleModule *module, bool selectOnInGate);
        virtual int select() override;
};

/**
* Chooses the longest queue (where length>0 of course).
* Input strategy (for servers).
*/
class QUEUEING_API LongestQueueSelectionStrategy : public SelectionStrategy
{
    public:
        LongestQueueSelectionStrategy(cSimpleModule *module, bool selectOnInGate);
        virtual int select() override;
};

// progettos
/**
* End all the tasks in a queue, then chooses cyclically the next one.
* Input strategy (for servers).
*/
class QUEUEING_API ExhaustiveServiceSelectionStrategy : public SelectionStrategy
{
    private:
        int actualInputGate; // actual input gate
    public:
        ExhaustiveServiceSelectionStrategy(cSimpleModule *module, bool selectOnInGate);
        virtual int select() override;
};

}; //namespace

#endif

```

Listing 8: "SelectionStrategies.h"

A.4.2 SelectionStrategies.cc

```

//
// This file is part of an OMNeT++/OMNEST simulation example.

```

```

//
// Copyright (C) 2006–2015 OpenSim Ltd.
//
// This file is distributed WITHOUT ANY WARRANTY. See the file
// 'license' for details on this and other legal matters.
//

#include "SelectionStrategies.h"
#include "PassiveQueue.h"
#include "Server.h"

namespace queueing {

SelectionStrategy::SelectionStrategy(cSimpleModule *module, bool selectOnInGate)
{
    hostModule = module;
    isInputGate = selectOnInGate;
    gateSize = isInputGate ? hostModule->gateSize("in") : hostModule->gateSize("out");
}

SelectionStrategy::~SelectionStrategy()
{
}

SelectionStrategy *SelectionStrategy::create(const char *algName, cSimpleModule *module,
bool selectOnInGate)
{
    SelectionStrategy *strategy = nullptr;

    if (strcmp(algName, "priority") == 0) {
        strategy = new PrioritySelectionStrategy(module, selectOnInGate);
    }
    else if (strcmp(algName, "random") == 0) {
        strategy = new RandomSelectionStrategy(module, selectOnInGate);
    }
    else if (strcmp(algName, "roundRobin") == 0) {
        strategy = new RoundRobinSelectionStrategy(module, selectOnInGate);
    }
    else if (strcmp(algName, "shortestQueue") == 0) {
        strategy = new ShortestQueueSelectionStrategy(module, selectOnInGate);
    }
    else if (strcmp(algName, "longestQueue") == 0) {
        strategy = new LongestQueueSelectionStrategy(module, selectOnInGate);
    }
    else if (strcmp(algName, "exhaustiveService") == 0) {
        strategy = new ExhaustiveServiceSelectionStrategy(module, selectOnInGate);
    }

    return strategy;
}

cGate *SelectionStrategy::selectableGate(int i)
{
    if (isInputGate)
        return hostModule->gate("in", i)->getPreviousGate();
    else
        return hostModule->gate("out", i)->getNextGate();
}

```

```

bool SelectionStrategy::isSelectable(cModule *module)
{
    if (isInputGate) {
        IPassiveQueue *pqueue = dynamic_cast<IPassiveQueue *>(module);
        if (pqueue != nullptr)
            return pqueue->length() > 0;
    }
    else {
        IServer *server = dynamic_cast<IServer *>(module);
        if (server != nullptr)
            return server->isIdle();
    }

    throw cRuntimeError("Only IPassiveQueue (as input) and IServer (as output) is supported by this Strategy");
}

//

```

```

PrioritySelectionStrategy::PrioritySelectionStrategy(cSimpleModule *module, bool
selectOnInGate) :
    SelectionStrategy(module, selectOnInGate)
{
}

int PrioritySelectionStrategy::select()
{
    // return the smallest selectable index
    for (int i = 0; i < gateSize; i++)
        if (isSelectable(selectableGate(i)->getOwnerModule()))
            return i;

    // if none of them is selectable return an invalid no.
    return -1;
}

//

```

```

RandomSelectionStrategy::RandomSelectionStrategy(cSimpleModule *module, bool
selectOnInGate) :
    SelectionStrategy(module, selectOnInGate)
{
}

int RandomSelectionStrategy::select()
{
    // return the smallest selectable index
    int noOfSelectables = 0;
    for (int i = 0; i < gateSize; i++)
        if (isSelectable(selectableGate(i)->getOwnerModule()))
            noOfSelectables++;

    int rnd = hostModule->intuniform(1, noOfSelectables);

    for (int i = 0; i < gateSize; i++)

```

```

        if (isSelectable(selectableGate(i)->getOwnerModule()) && (--rnd == 0))
            return i;

    return -1;
}

//

```

```

RoundRobinSelectionStrategy::RoundRobinSelectionStrategy(cSimpleModule *module, bool
selectOnInGate) :
    SelectionStrategy(module, selectOnInGate)
{
    lastIndex = -1;
}

int RoundRobinSelectionStrategy::select()
{
    // return the smallest selectable index
    for (int i = 0; i < gateSize; ++i) {
        lastIndex = (lastIndex+1) % gateSize;
        if (isSelectable(selectableGate(lastIndex)->getOwnerModule()))
            return lastIndex;
    }

    // if none of them is selectable return an invalid no.
    return -1;
}

//

```

```

ShortestQueueSelectionStrategy::ShortestQueueSelectionStrategy(cSimpleModule *module,
bool selectOnInGate) :
    SelectionStrategy(module, selectOnInGate)
{
}

int ShortestQueueSelectionStrategy::select()
{
    // return the smallest selectable index
    int result = -1; // by default none of them is selectable
    int sizeMin = INT_MAX;
    for (int i = 0; i < gateSize; ++i) {
        cModule *module = selectableGate(i)->getOwnerModule();
        int length = (check_and_cast<IPassiveQueue *>(module))->length();
        if (isSelectable(module) && (length < sizeMin)) {
            sizeMin = length;
            result = i;
        }
    }
    return result;
}

//

```

```

LongestQueueSelectionStrategy::LongestQueueSelectionStrategy(cSimpleModule *module, bool
    selectOnInGate) :
    SelectionStrategy(module, selectOnInGate)
{
}

int LongestQueueSelectionStrategy::select()
{
    // return the longest selectable queue
    int result = -1; // by default none of them is selectable
    int sizeMax = -1;
    for (int i = 0; i < gateSize; ++i) {
        cModule *module = selectableGate(i)->getOwnerModule();
        int length = (check_and_cast<IPassiveQueue *>(module))->length();
        if (isSelectable(module) && length > sizeMax) {
            sizeMax = length;
            result = i;
        }
    }
    return result;
}

//

ExhaustiveServiceSelectionStrategy::ExhaustiveServiceSelectionStrategy(cSimpleModule *
    module, bool selectOnInGate) :
    SelectionStrategy(module, selectOnInGate)
{
    actualInputGate = 0;
}

int ExhaustiveServiceSelectionStrategy::select()
{
    // previously selected queue is not empty
    if (isSelectable(selectableGate(actualInputGate)->getOwnerModule()))
        return actualInputGate;
    // scan cyclically the next non empty queue
    else {
        for (int i = 1; i < gateSize; i++) {
            int gn = (actualInputGate + i) % gateSize;
            if (isSelectable(selectableGate(gn)->getOwnerModule())) {
                actualInputGate = gn;
                return gn;
            }
        }
    }

    // if none of them is selectable return an invalid no.
    return -1;
}

}; //namespace

```

Listing 9: "SelectionStrategies.cc"

A.5 Server (pssqueueinglib)

Aggiunte:

- *@signal[droppedForDeadline](type="long")*: *signal* per la registrazione dei *Job* scartati a causa di un inizio di servizio successivo alla loro *absoluteDeadline* (*Server.ned*)
- *@statistic[droppedForDeadline](title="drop event for deadline reached";record=vector?,count;interpolationmode=sample-and-hold)*: statistica relativa ai *Job* scartati a causa di un inizio di servizio successivo alla loro *absoluteDeadline* (*Server.ned*)
- *bool checkJobDeadline = default(true)*: specifica se sia necessario controllare *Job.absoluteDeadline* prima dell'inizio del servizio (*Server.ned*)
- *simsignal.t droppedForDeadlineSignal*: *signal* per la registrazione dei *Job* scartati a causa di un inizio di servizio successivo alla loro *absoluteDeadline* (*Server.h*)
- *bool checkJobDeadline*: specifica se sia necessario controllare *Job.absoluteDeadline* prima dell'inizio del servizio (*Server.h*)

Modifiche:

- *void Server::initialize()*: configurazione dei nuovi valori aggiunti (*Server.cc*)
- *void Server::handleMessage(cMessage *msg)*: modifiche al metodo per implementare il controllo di *Job.absoluteDeadline* e richiedere un nuovo *Job* in caso quello attuale venga scartato

A.5.1 Server.ned

```
//
// This file is part of an OMNeT++/OMNEST simulation example.
//
// Copyright (C) 2006–2015 OpenSim Ltd.
//
// This file is distributed WITHOUT ANY WARRANTY. See the file
// 'license' for details on this and other legal matters.
//

package org.omnetpp.queueing;

//
// Queue server. It serves multiple input queues (PassiveQueue), using a preset
// algorithm. Inputs must be connected to Passive Queues (PassiveQueue)
//
simple Server
{
    parameters:
        @group( Queueing );
        @display( "i=block/server" );
        @signal[ busy ]( type="bool" );
        @statistic[ busy ]( title="server busy state"; record=vector?, timeavg;
            interpolationmode=sample-and-hold );
        // progettos
        @signal[ droppedForDeadline ]( type="long" );
        @statistic[ droppedForDeadline ]( title="drop event for deadline reached"; record=
            vector?, count; interpolationmode=none );
}
```



```

        string fetchingAlgorithm @enum("priority","random","roundRobin","longestQueue","
            exhaustiveService") = default("priority");
        // how the next job will be choosen from the attached queues
        volatile double serviceTime @unit(s); // service time of a job
        // progettoss
        bool checkJobDeadline = default(false);
    gates:
        input in[];
        output out;
}

```

Listing 10: "Server.ned"

A.5.2 Server.h

```

//
// This file is part of an OMNeT++/OMNEST simulation example.
//
// Copyright (C) 2006–2015 OpenSim Ltd.
//
// This file is distributed WITHOUT ANY WARRANTY. See the file
// 'license' for details on this and other legal matters.
//

#ifndef _QUEUEING_SERVER_H
#define _QUEUEING_SERVER_H

#include "IServer.h"

namespace queueing {

class Job;
class SelectionStrategy;

/**
 * The queue server. It cooperates with several Queues that which queue up
 * the jobs, and send them to Server on request.
 *
 * @see PassiveQueue
 */
class QUEUEING_API Server : public cSimpleModule, public IServer
{
private:
    simsignal_t busySignal;
    bool allocated;

    SelectionStrategy *selectionStrategy;

    Job *jobServiced;
    cMessage *endServiceMsg;

    // progettoss
    simsignal_t droppedForDeadlineSignal;
    bool checkJobDeadline;

public:
    Server();
    virtual ~Server();

```

```

protected:
    virtual void initialize() override;
    virtual int numInitStages() const override {return 2;}
    virtual void handleMessage(cMessage *msg) override;
    virtual void refreshDisplay() const override;
    virtual void finish() override;

public:
    virtual bool isIdle() override;
    virtual void allocate() override;
};

}; //namespace

#endif

```

Listing 11: "Server.h"

A.5.3 Server.cc

```

//
// This file is part of an OMNeT++/OMNEST simulation example.
//
// Copyright (C) 2006–2015 OpenSim Ltd.
//
// This file is distributed WITHOUT ANY WARRANTY. See the file
// 'license' for details on this and other legal matters.
//

#include "Server.h"
#include "Job.h"
#include "SelectionStrategies.h"
#include "IPassiveQueue.h"

namespace queueing {

Define_Module(Server);

Server::Server()
{
    selectionStrategy = nullptr;
    jobServiced = nullptr;
    endServiceMsg = nullptr;
    allocated = false;
}

Server::~Server()
{
    delete selectionStrategy;
    delete jobServiced;
    cancelAndDelete(endServiceMsg);
}

void Server::initialize()
{
    busySignal = registerSignal("busy");
    emit(busySignal, false);
}

```

```

endServiceMsg = new cMessage("end-service");
jobServiced = nullptr;
allocated = false;
selectionStrategy = SelectionStrategy::create(par("fetchingAlgorithm"), this, true);
if (!selectionStrategy)
    throw cRuntimeError("invalid_selection_strategy");
droppedForDeadlineSignal = registerSignal("droppedForDeadline");
checkJobDeadline = par("checkJobDeadline").boolValue();
}

void Server::handleMessage(cMessage *msg)
{
    if (msg == endServiceMsg) {
        ASSERT(jobServiced != nullptr);
        ASSERT(allocated);
        simtime_t d = simTime() - endServiceMsg->getSendingTime();
        jobServiced->setTotalServiceTime(jobServiced->getTotalServiceTime() + d);
        send(jobServiced, "out");
        jobServiced = nullptr;
        allocated = false;
        emit(busySignal, false);

        // examine all input queues, and request a new job from a non empty queue
        int k = selectionStrategy->select();
        if (k >= 0) {
            EV << "requesting_job_from_queue_" << k << endl;
            cGate *gate = selectionStrategy->selectableGate(k);
            check_and_cast<IPassiveQueue *>(gate->getOwnerModule())->request(gate->
                getIndex());
        }
    }
    else {
        if (!allocated)
            error("job_arrived, but the sender did not call allocate() previously");
        if (jobServiced)
            throw cRuntimeError("a_new_job_arrived_while_already_servicing_one");
        jobServiced = check_and_cast<Job *>(msg);
        simtime_t serviceTime = par("serviceTime");
        if (checkJobDeadline) {
            if (jobServiced->getAbsoluteDeadline() < simTime()) {
                EV << "Dropped!" << endl;
                if (hasGUI())
                    bubble("Dropped!");
                emit(droppedForDeadlineSignal, 1);
                delete msg;
                allocated = false;
                jobServiced = nullptr;
                int k = selectionStrategy->select();
                if (k >= 0) {
                    EV << "requesting_job_from_queue_" << k << endl;
                    cGate *gate = selectionStrategy->selectableGate(k);
                    check_and_cast<IPassiveQueue *>(gate->getOwnerModule())->request(
                        gate->getIndex());
                }
            }
            else {
                scheduleAt(simTime()+serviceTime, endServiceMsg);
                emit(busySignal, true);
            }
        }
    }
}

```

```

        }
    }
    else {
        scheduleAt(simTime()+serviceTime, endServiceMsg);
        emit(busySignal, true);
    }
}

void Server::refreshDisplay() const
{
    getDisplayString().setTagArg("i2", 0, jobServiced ? "status/execute" : "");
}

void Server::finish()
{
}

bool Server::isIdle()
{
    return !allocated; // we are idle if nobody has allocated us for processing
}

void Server::allocate()
{
    allocated = true;
}

}; //namespace

```

Listing 12: "Server.cc"