ABSTRACT
This paper presents an introduction to DESMO-J, a comprehensive framework developed at the University of Hamburg for rapidly implementing discrete event simulation models in Java. We describe the most important DESMO-J components. Key features include a clear separation of model development and experiment conduction and the support of both event-oriented and process-oriented model descriptions; both views can be combined within a single model. DESMO-J is extensively used in research, teaching and technology transfer at the University of Hamburg and at other universities. However, applications of DESMO-J have been reported primarily in logistics. We present a simulation model from an entirely different area, namely the simulation of an abstract mobile ad hoc network. A MANET is a collection of mobile devices dynamically forming a wireless network. The model concerns the evaluation of a protocol for the decentralized and fair data rate allocation. The implementation and a typical experiment are described, illustrating the use, capabilities and flexibility of DESMO-J.

Keywords: discrete event simulation, simulation framework, Java, mobile ad hoc networks

1. INTRODUCTION
This paper presents an introduction to DESMO-J (Discrete Event Simulation Modelling in Java), a comprehensive framework developed at the University of Hamburg, Germany for rapidly implementing discrete event simulation models in the widely-used object-oriented programming language Java. We first describe DESMO-J’s black box components which provide the simulation infrastructure and the hot spot components which are abstract classes to be implemented by the user such as events and processes which determine the behaviour of the simulator. We then describe the important design features of DESMO-J.

We next present an application of DESMO-J to model the behaviour of mobile ad hoc networks (MANETs) which are networks of wireless devices which do not necessarily require a communications infrastructure. In this way the reader gains an impression of how simulation modelling in DESMO-J is conducted. The abstract MANET model is based on Crowcroft, Gibbens, Kelly and Östring (2004); the focus of the MANET model is the design of a decentralized and fair protocol to allocate resources (battery power and bandwidth) such that nodes are provided with incentives to use their resources to relay transmissions on behalf of other nodes.

The paper is organized as follows. Section 2 provides an introduction to the DESMO-J framework, focusing on its most important features, components and design decisions. Section 3 describes MANETs as an application area. A protocol for the decentralized and fair allocation of resources (bandwidth and battery power) which determines the data rates based on congestion pricing is described. This section also establishes the major features that a simulator requires for the purpose of simulating the abstract MANET model. Section 4 shows how the MANET environment is implemented in DESMO-J; this section is completed by describing a typical MANET simulation experiment and discussing the results. Conclusions are given in Section 5.

2. DESMO-J
DESMO-J is a framework in Java developed at the University of Hamburg for rapidly implementing discrete event simulation models. DESMO-J is licensed under the Apache license version 2.0. The following sections provide a short introduction to DESMO-J covering some general properties, important components and simulation modelling as well as some design decisions. The reader is referred to Page and Kreutzer (2005) and to the web page http://www.desmoj.de from where the binaries, the source code and other resources can be obtained. These resources include an example model, the API documentation and a tutorial; the latter eases getting started with simulation modelling in DESMO-J: it contains a step-by-step implementation of a small container terminal model using either the event-oriented or the process-oriented view.
2.1. General
A framework can be defined as a collection of different components with coherent software architecture and a defined cooperative behaviour serving a particular task. As a framework at the level of a general purpose programming language, DESMO-J requires the user to code discrete event simulation models in Java. Although simulators at the tool or model level (see Page and Kreutzer 2005, Chapter 9.1) may provide a quicker means of developing standard models such as queueing/server systems, the flexible approach adopted by DESMO-J ensures that complex systems can be adequately mapped into a simulation model as no simulator-specific programming languages or restricted APIs constrain model implementation. This facilitates the modelling of sophisticated strategies and complex system behaviour.

The advantages of choosing Java also include addressing a wide-spread community of programmers, clear and robust object-oriented programming without pitfalls like dangling pointers and memory leaks, as well as platform independence. At the same time, the potential disadvantages of Java, most notably that runtime performance is worse in terms of execution efficiency than for example C++, have been lessened by improvements in just-in-time compilation and the optimization of garbage collection.

Several example applications of DESMO-J have been published including multi-agent city couriers (Knaak, Meyer and Page 2004), harbour logistics (Page and Neufeld 2003), vehicular traffic (Wittmann, Göbel, Möller and Schroer 2007) and material flow analysis (Wohlgemuth, Page, Mäusbacher and Staudt-Fischbach 2004).

The clear design and the ease and flexibility of simulation modelling provided by DESMO-J have resulted in this simulator being intensively used in simulation courses for students of Computer Science and Information Systems at the University of Hamburg and elsewhere, in applied research projects as well as for industrial simulation studies in technology transfer projects (Page and Kreutzer 2006). The business process modelling tool iProcess Suite (TIBCO Software Inc., http://www.tibco.com) relies on DESMO-J as a simulation engine.

2.2. Components and Modelling
DESMO-J provides two groups of components (Java classes). Black box components are “closed” classes that are ready to use: they provide a complete environment to execute discrete event simulations. This environment includes the core simulation infrastructure consisting of an event list storing pending events and processes, the scheduler controlling model execution, the simulation clock as well as an Experiment class to integrate these components. The environment also includes facilities for collecting and reporting statistical data, random number generation, a variety of probability distributions as well as generic model components like queues.

Given the availability of ready-to-use black boxes, the user can concentrate on implementing the properties and the behaviour of the system being modelled. For this purpose, DESMO-J provides a second class of components referred to as white box components or hot spots. White box components are abstract Java classes whose implementation has to be completed by the user.

The DESMO-J user can choose between an event-oriented view and a process-oriented view of describing a model. The event-oriented perspective (“bird’s eye view”) requires implementing subclasses of the hot spot Event defining the changes that are to be applied to or triggered by an entity at a certain instant in time. Similarly, subclasses of ExternalEvent represent state changes that are not associated with a specific entity, but with the model as a whole, for instance external arrivals. Entities are objects which are derived from the class Entity. Examples of possible state changes conducted by events and external events in their so-called event routines are: attributes modified, entities enqueued or dequeued, events scheduled into the event list for later execution, or events in the event list being brought forward or postponed (rescheduled) or cancelled.

In contrast, a process-oriented model definition involves describing the activities conducted by every entity over time during its life cycle; this yields a “worm’s eye view” representing the model behavior from the perspective of the entities. Process life cycle definitions require subclassing SimProcess. In addition to modifying attributes and queueing operations, a life cycle may include waiting for a fixed amount of time or for an indeterminate period until activation by another process, and activating or interrupting other processes. Note that events are executed at specific instants in time without interruption while processes (potentially) persist as time passes, handing over program control amongst each other.

An important feature of DESMO-J is that both views can be combined in a single model, for example an event activating a process that in turn schedules or cancels another event. The user is free to use whichever modelling view is better suited for certain model entities and their behaviour.

Finally, regardless of the modelling view used for model description, any model requires a subclass of the hot spot Model in which all model components are created, and dynamic behaviour is triggered by scheduling initial events or activating processes.

2.3. Design Decisions
Fig. 1 shows the hierarchy of some important DESMO-J classes. NamedObject (implicitly derived from java.lang.Object) as the root for all DESMO-J classes ensures that all objects in DESMO-J have a meaningful name. The separation of the model, which is composed from objects derived from ModelComponent, and the experiment (according to the Façade design pattern, the class Experiment initializes and manages the simulation infrastructure).
keeps model development and experimentation independent of each other. This for example allows for a series of experiments, where each experiment is represented by different Experiment objects, to be conducted with the same model Model object.

![DESMO-J Class Hierarchy](image)

Figure 1: The DESMO-J class hierarchy

Model components are partitioned into two groups. First, components can be derived from Schedulable, representing objects which the discrete event scheduler can process yielding the desired model behaviour. Note that SimProcess is derived from Entity, allowing for scheduling events to occur for a given entity as well as for a process, which facilitates combining event- and process-oriented models. Second, static model components are derived from Reportable, denoting their ability to report statistical data generated by an experiment, for example a Queue reporting its average length and waiting time. DESMO-J also emphasizes the encapsulation and re-usability of model components: a Model class, itself derived from ModelComponent, can be used within another model, thus forming hierarchical structures of submodels.

Other features and design decisions of DESMO-J include data being collected automatically if desired (based on the Observer design pattern), a GUI for model and experiment parameterization and experiment execution as well as white- and black box components for higher-level process synchronization such as processes waiting for resources from a pool which are acquired/released mutually exclusively.

### 3. Mobile Ad Hoc Networks

The applications of DESMO-J already published as quoted in Section 2.1 may give rise to the impression that DESMO-J is primarily used in logistics simulations. However, logistics is by far not the only field where discrete event simulation can be successfully applied. This section presents mobile ad hoc networks (MANETs) as an entirely different application area. The focus of the simulation is to evaluate a protocol for resource (battery power and bandwidth) and data rate allocation that is decentralized and fair. The protocol description was initially presented in Göbel and Krzesinski (2008b). The section concludes by establishing the requirements a simulator has to meet in order to adequately model the abstract MANET environment based on Crowcroft, Gibbens, Kelly and Östring (2004).

#### 3.1. Definition

Mobile ad hoc networks (MANETs) are formed by mobile nodes using wireless communication. Each node can be source and a destination of data transmissions and also functions as a router. This cooperation ensures that nodes that are out of each other's radio range can transmit to each other without relying on an existing infrastructure or on the network topology being known in advance. A survey of MANETs focusing on Quality of Service aspects is given in Reddy, Karthigeyan, Manoj and Murthy (2006). A typical MANET application might be passengers at an airport terminal carrying wireless-enabled devices which allow multi-hop routes to be established to connect these devices to each other or to an Internet access point in case the transmission range does not suffice to directly connect to the target.

#### 3.2. Resource Allocation and Fairness

Suppose the nodes that form a MANET are owned by a single authority. Then it is in the authority's interest that the nodes make their resources (battery power and bandwidth) available to forward data transmissions between origin and destination nodes that are not in direct radio contact with each other and which are dependent on one or more relays to forward their transmissions. However, if the nodes belong to different authorities, then a relay might lack an incentive to drain its battery power and allocating its bandwidth by altruistically supporting data transmissions of which it is neither the source nor the destination. The functioning of a MANET depends on the participation of the nodes. However, a node will only participate if fairness in resource allocation is enforced: if a node acts as a relay (transit) node and grants its resources to enable the data transmissions of other nodes, then the relay node will expect the same service in return when it attempts to send its own transmissions on multi-hop routes. Moreover, should a node never need to use another node's resources, then that node will demand monetary compensation for the use of its resources.

Crowcroft, Gibbens, Kelly and Östring (2004) investigate an abstract fluid-level MANET model, for which they present a mechanism of resource pricing and data rate control based on congestion prices that are assigned to the resources at every node. Prices are measured in terms of a virtual currency referred to as credits; the power congestion price $\mu_j^c(t)$ that node $j$ charges expressed in credits per unit power (credits/W) satisfies the differential equation (DE)

$$\frac{d\mu_j^c(t)}{dt} = \frac{\kappa \mu_j^c(t)}{\Gamma_j} (\gamma_j(t) - \Gamma_j)$$

with initial value $\mu_j^c(0) = 1$ where $\kappa$ is a constant of dimension seconds$^{-1}$, $\gamma_j(t)$ is the power in use at
node \( j \) at time \( t \) and \( \Gamma_j \) is target power node \( j \) is willing to make available to the network. Likewise, the bandwidth congestion price \( \mu^\delta_j(t) \) expressed in credits per unit flow (credits/Mb) satisfies the DE

\[
\frac{d}{dt} \mu^\delta_j(t) = \frac{k \mu^\delta_j(t)}{C_j} (c_j(t) - C_j)
\]

with initial value \( \mu^\delta_j(0) = 1 \) where \( c_j(t) \) is the bandwidth in use at node \( j \) at time \( t \) and \( C_j \) is the target bandwidth of node \( j \) not to be exceeded by the MANET traffic.

These DEs leave the congestion prices constant when the current utilization of a resource matches the target utilization. The prices increase when the resources are over-utilized and the prices decrease when the resources are under-utilised. The DEs must be evaluated frequently to ensure the power \( \gamma_j(t) \) and bandwidth in use \( c_j(t) \) will only temporarily exceed the target resource utilization: Over-utilization will cause the congestion prices to increase, which will lower the flow rates. Crowcroft, Gibbens, Kelly and Östring (2004) suggest numerically evaluating the DEs every \( \Delta = 0.01 \) seconds which will result in target resource utilization being exceeded infrequently and by small amounts.

Each node has a credit balance \( b_j(t) \). Data transmissions are connected on the least-cost route, assuming a route to the destination exists. During the data transmission, the originating node pays credits for the power and bandwidth congestion costs incurred at the downstream nodes along its route according to prices determined by the DEs described above. In return, the source node receives credits for data transmissions that it relays and for which it provides its resources for transmitting or receiving. During each second the nodes are allowed to spend a fraction \( \alpha b_j(t) \) of their current credit balance \( b_j(t) \) to pay for the resources that they will use at the downstream nodes along their route. The data rate allocated to an originating node will therefore increase/decrease as its credit balance increases/decreases. A data transmission terminates after the desired amount of data has been sent. A transmission is discarded if it has not completed after a maximum transmission duration.

The scheme provides the required incentive to collaborate when the nodes do not belong to a single authority: the nodes need to acquire credits by providing their resources to other nodes in order to be able to send traffic themselves. However, some nodes may not be able to attract sufficient transit traffic to cover the costs of transmitting their own traffic. A node at the edge of the network may hardly be used as transit at all. In contrast, a node at the centre of the network may face such a high demand to act as a relay that it earns more credits than it is able to spend.

This results in a flow of credits from (1a) those nodes that are located in disadvantageous positions and from (1b) those nodes that transmit more data than the average node, to (2a) those nodes whose credit balance benefits from a favourable location and to (2b) those nodes with low amounts of data to transmit. Credit discounting was therefore suggested in Crowcroft, Gibbens, Kelly and Östring (2004) allowing nodes whose credit balance is less than a certain amount (the target credit balance) to create a certain amount of credits per unit of time, while those nodes whose credit balance exceeds the target balance must destroy surplus credits at the same rate. Since the creation and destruction of credits improves or degrades the amount of resources a node can obtain subsequently, compensation is required: As proposed in detail in Göbel and Krzesinski (2008b), nodes creating credits are monetarily charged (e.g. a certain amount of dollars per credit). The payments are deposited in a fund which is used to compensate those nodes which destroy credits. Note that evaluating net payments provide a means of monetarily evaluating node locations.

### 3.3. Local Control

The performance of the abstract MANET model proposed by Crowcroft, Gibbens, Kelly and Östring (2004) depends on the following factors (not necessarily exhaustive):

- propagating pricing information subject to delays
- parameters of the data rate allocation scheme, most important the rate of creating/destroying credits at under/overprovisioned nodes
- assigning different amounts of energy and bandwidth to the nodes
- various types of node movements (see Camp, Boleng and Davies 2002 for an overview)
- radio interference (investigated in Göbel, Krzesinski and Mandjes 2007).

Crowcroft, Gibbens, Kelly and Östring (2004) demonstrate that in the absence of radio interference their mechanism of resource pricing and data rate control achieves optimal resource utilization (and near maximum data rates subject to the constraints is resources the node are willing to provide to the network) and global stability of the system. However, although no central controller is needed, their scheme assumes that the resource utilization and the prices at each node are instantaneously available at all nodes in the network. For that reason, their results represent an upper bound on the performance of a fluid-flow network where data rates are determined based on (partially) outdated prices that have to be propagated from node to node along each route subject to delays in order make them available locally.

An example of a signalling protocol is presented in Göbel and Krzesinski (2008b) to conduct this exchange of information. Two types of signals are introduced:
• An update cost (UC) signal is sent periodically from the destination to the origin of the data transmission: this signal gathers the local resource prices which are valid at the instant the signal arrives at each node on the route.
• Once the update cost signal reaches the origin, the source updates the data rate of the transmission as well as its credit balance and returns an update flow (UF) signal to the destination: this signal disburses the credits to be paid to each transit node on the multi-hop route for the data transmission during the past period.

Special UF signals announcing a preliminary data rate of 0 are sent to initialize a flow or to reroute a flow (rerouting will occur if a relay node along a multi-hop route leaves the network because its battery power is exhausted) and to terminate a data transmission. These signals do not represent a reply to an UC signal. Fig. 2 summarizes the protocol – see Göbel and Krzesinski (2008b) for more details.

![Figure 2: The protocol](image)

3.4. Simulation Requirements
To evaluate the impact of decentralized (i.e. signal-based) data rate allocation on the performance of the decentralized network in terms of data throughput, resource utilization and transmissions discarded, a simulator for the abstract MANET environment is required that must meet the following criteria:

• The simulator must adequately map the abstract MANET system described above into a valid simulation model. To do this, the simulator must be sufficiently flexible: it must not enforce undesired abstraction or idealization by not supporting the simulation of parts of the model, nor must it impose an overhead by requiring unnecessary details to be simulated. This implies that the simulator supports modelling using the event-oriented view (see Section 2.2) since the model behaviour depends upon triggers (for example a data transmission starting, an UC signal processed, a node departing) occurring at certain instants in time. Conversely, there are no complex time consuming activities for which the process-oriented view would be better suited.

• The simulator must support the automated collection, aggregation and reporting of relevant data.
• The simulator must support the separation of the MANET simulation logic and the experiment scenarios such as the spatial configuration of the nodes, resource provisioning at the nodes and the movement patterns of the mobile nodes. This eases making changes to the network topology configuration.
• Modifications and extensions to the model are facilitated thanks to the object-oriented programming paradigm. Modifications like adjusting movement patterns are conducted by refining the relevant component already defined in the simulation model. An example extension is given in (Göbel and Krzesinski 2008a) where the MANET simulator components were augmented to model a mobile sensor network where area coverage currently attained is measured, which the autonomously moving nodes seek to maximize.

DESMO-J meets these criteria. Flexibility, ease of modification and extension are inherited from using Java as a programming language. DESMO-J imposes no restrictions and hardly any overhead to the definition of entities (derived from Entity) and behaviours (method eventRoutine() of subclasses Event and ExternalEvent): Since derived from Named Object, a name must be assigned to entities and events. Additional overhead may be incurred by including the entities and events in the trace output, or by increased simulation run-time performance, for example each Schedulable holding a reference to its current event list entry, if currently scheduled. DESMO-J also provides comprehensive reports which are generated automatically after simulation experiment completion. Different experiment scenarios are described by parsing XML input files which contain the different network configurations.

As far as disadvantages are concerned, we have already mentioned Java's bias in run-time performance (Section 2.1). Moreover, DESMO-J does not (yet) provide a native visualization facility. For the purpose of MANET simulation development, visualization is not strictly necessary, yet it could be useful: although visual impressions are no substitute for empirically well-founded analysis, visual metaphors allow for rapid communication and comprehension of model states and behaviours. In particular, erroneous or implausible behaviour may be easier to detect visually than by analyzing aggregated statistical data (see Page and Kreutzer 2005, Chapter 9.6). In this regard, the DESMO-J user can implement his/her own visualization using Java's built-in graphics libraries or (more conveniently) use existing frameworks serving this purpose; our MANET simulator uses the
framework JFreeChart (http://www.jfree.org/jfreechart) to visualize mobile node movement.

When comparing a general-purpose simulator like DESMO-J to dedicated telecommunication network simulators like OMNeT++ (http://www.omnetpp.org) or ns2 (http://www.isi.edu/nsnam/ns), we need to emphasize that such dedicated network simulators can emulate (mobile) network behaviour at the protocol/signal level.

Observe that in the abstract MANET model of Crowcroft, Gibbens, Kelly and Östring (2004) user data transmission are modelled by assigning flows (a macroscopic fluid-level model), abstracting from simulation at packet level: Estimation of the relevant performance indicators like data rates, resource utilization and the ratio of successful to unsuccessful transmissions does not require such a detailed low-level model. UC and UF signals, though, are explicitly included in model since they are necessary to determine data rates (data rates may change on receipt), yet this does not imply that these control signals themselves have to be modelled bit by bit at the protocol level: only the impact caused by the receipt of the control signals (data rate adjustments, another control signal being sent after a delay, yielding price propagation subject to delays) has to be represented in the model.

In summary, we consider DESMO-J to be an adequate choice for simulating the abstract MANET model because the requirements as established above are met, while the low-level protocol details (which are the strength of dedicated telecommunication network simulators) are not necessary.

4. MANET SIMULATION WITH DESMO-J
This section demonstrates how the abstract MANET environment described in Section 3 is implemented in DESMO-J. In particular, the Event classes that determine the behaviour of the network are described in detail using UML activity diagrams. The section is completed by describing a sample experiment and discussing the results.

4.1. Simulator Development
A simulation implementation in DESMO-J that uses the event-oriented view requires at least three types of components:

A. Entities
Entities are the relevant objects that the system consists of and that determine the behaviour of the system have to be identified. Note that these entities do not necessarily persist permanently: they may be created and discarded at some instant in time. In the MANET model, the entities to be included are mobile node, transmission, UC signal and UF signal.

For each of these entity types, a subclass of Entity has to be derived; since both signal types share some attributes, like storing the route to traverse, they should be derived from a common superclass, which in turn is a subclass of Entity. Table 1 summarizes their most important attributes. Other entity types are not strictly necessary, though such design decisions typically leave a degree of freedom to the user, who might decide to encapsulate a route into an entity of its own instead of route attributes being represented as arrays of nodes in transmission and signal entities. Likewise, the credits possessed by the nodes and their power and bandwidth resources are assigned to nodes in terms of numerical attributes.

<table>
<thead>
<tr>
<th>Entity type</th>
<th>Selected attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mobile node</td>
<td>Position; movement pattern; power, bandwidth available, in use, prices; current transmissions; credit balance</td>
</tr>
<tr>
<td>Transmission</td>
<td>Route; data rate; data already sent, left to transmit</td>
</tr>
<tr>
<td>UC signal</td>
<td>Route; costs gathered</td>
</tr>
<tr>
<td>UF signal</td>
<td>Route; data rate; credits transferred</td>
</tr>
</tbody>
</table>

B. Events
Once the entities are defined, the model behaviour is described in terms of subclasses of Event, representing events that refer to a certain entity (see Section 2.2) such as a particular transmission terminating, and in terms of subclasses of External Event, representing events not referring to a certain existing entity, like a new node joining the network. Six of the eight types of events listed below are described in some detail using the UML activity diagram notation in Fig. 3, subject to two stereotypes: ≪create≫ refers to the creation of an entity, while ≪schedule≫ represents operations that modify the simulator’s event list (scheduling, rescheduling or canceling an event); see Knaak and Page (2006) for more details about the usage of UML activity diagrams for discrete event simulation modelling. The two events not shown in Fig. 3, update positions and update statistics are sufficiently simple that the UML activity diagrams are omitted.

- *Transmission starts* (TS, Fig. 3a, Event refers to the node of origin) initiates a new data transmission. If a route exists, the transmission is initiated, which requires sending an UF signal, see Fig. 2. The next transmission is scheduled.
- *Transmission completed* (TC, Fig. 3b, Event refers to a transmission) terminates a data transmission, which requires sending an UF signal.
- *Node arrival* (NA, Fig. 3c, External Event) creates a new node that joins the network. Its first data transmission is scheduled as well as the arrival of the next node.
Node departure (ND, Fig. 3d, External Event) removes a node from the network. Transmissions that are relayed through this node are rerouted (if possible) and transmissions that originate at this node are cancelled. The next node departure is scheduled.

Update cost arrival (UCA, Fig. 3e, Event) refers to an UC signal arriving at a node. Provided the relevant transmission is neither completed nor cancelled, local cost data are collected and forwarded. The destination schedules the next UC signal. If the UC signal arrived at the

Figure 3: UML activity diagrams describing the behaviour of the MANET model (Knaak and Page 2006)
origin, the data rate is adjusted and an UF signal is sent.

- **Update flow arrival** (UFA, Fig. 3f, Event) refers to an UF signal denoting an UF signal arriving at a node. The local credit balance is updated, the origin is debited and the relays and the receiver are credited. In addition, credit discounting (Section 3.2) is applied. Further signals (forwarding this UF signal towards the destination or issuing an UC signal at the destination in reply to an initializing/rerouting UF signal) are sent if applicable.

- **Update positions** (UP, not shown, External Event) updates the positions of the nodes according to their movement patterns and schedules the next UP event.

- **Update statistics** (US, not shown, External Event) updates various statistics and schedules the next US event.

C. Model
A class derived from Model is required to setup the parameters and an initial configuration of the nodes (as parsed from the XML experiment scenario file) and to trigger the dynamic behaviour of the model in terms of the events being scheduled, namely the first transmission starting (TS) on each node. The events responsible for the arrival and departure of the nodes (if desired) and updating the positions and the statistics are entered into the event list.

4.2. Experiment Configuration
For a sample experiment, consider a 12×12 Manhattan grid on a 320m×320m plane. Mobile nodes, which may be interpreted as pedestrians carrying mobile networking devices, traverse this grid, proceeding from one randomly selected target (street intersection) to the next, subject to varying speeds and to varying delays once a target is reached or (for a shorter period) before other intersections are traversed.

Once per second (exponentially distributed), each mobile node \( \Gamma_j = 0.5 \text{ W} \), \( C_j = 10 \text{ Mb} \) initiates a data transmission of on average 2Mb (exponentially distributed). The target is either another mobile node (75%) or an external agency which is referred to as the outside world (25%). Nodes also may receive data transmissions from the outside world. Data transmissions to and from the outside world are routed via access points (APs) which is part of a fixed infrastructure (FI). The APs are connected to the outside world and the APs are logically fully meshed, so that a wired (AP to AP) hop can be used for data transmissions from one mobile node to another. Whether or not to include the FI in mobile to mobile routes and which AP to use for a data transmission to/from the outside world is determined by least-cost routing considerations. Note that unlike the mobile nodes and the outside world, the APs do not originate data transmissions.

The provider of the FI has to determine the number and the spatial distribution of the APs and how much transmission power and bandwidth to allocate to them. Fig. 4 shows that the APs may for example be located at the centres of hexagonal cells whose radius is equal to the transmission range of the mobile nodes, which is assumed to be 50m. This hexagonal pattern ensures that every mobile node is always in radio contact with at least one access point: it can be proved that no pattern exists that provides this property using fewer access points (Lim and Lee 1997).

![Hexagonal cells, radius 50 m](image_url)

Assume the FI provides considers two alternative FI configurations: The *dense* FI configuration uses hexagonal cells of radius 50 m to ensure that every mobile node is always in direct radio contact with at least one AP. This requires 15 APs. Initial simulation studies show that an AP resource configuration of \( \Gamma_j = 5 \text{ W} \), \( C_j = 50 \text{ Mb} \) is sufficient to carry all data transmissions. The *sparse* FI configuration allocates (approximately) the same power and bandwidth resources at only 8 APs (\( \Gamma_j = 10 \text{ W} \), \( C_j = 100 \text{ Mb} \)). The resulting hexagonal cells of radius 65 m do not guarantee that every mobile node is always in direct radio contact with at least one AP.

4.3. Results
Simulation runs using both the dense and the sparse FI configurations described above were executed 10 times for 10,000 seconds each, so that each node attempts approximately 10,000 transmissions, using different random number generator seeds. Table 2 shows some important performance indicators and their 95% confidence intervals.

<table>
<thead>
<tr>
<th>fixed infrastructure</th>
<th>avg. FI resource usage</th>
<th>avg. transmission throughput (Mb/s)</th>
<th>transmissions completed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>power (W)</td>
<td>bandwidth (Mb)</td>
<td>via FI</td>
</tr>
<tr>
<td>dense</td>
<td>21.2 ± 0.2</td>
<td>593.8 ± 6.9</td>
<td>195.4 ± 1.7</td>
</tr>
<tr>
<td>sparse</td>
<td>24.3 ± 0.1</td>
<td>584.3 ± 2.3</td>
<td>190.5 ± 0.6</td>
</tr>
</tbody>
</table>
We observe that in both FI configurations, the power and bandwidth of the APs namely $\sum C_j \geq 75 \text{W}$, $\sum C_j \geq 750 \text{Mb}$ are not fully utilized. This indicates that the resources provided by the service provider are sufficient, so that the traffic demand and the transmission capabilities of the mobile nodes are the bottlenecks that prevent the nodes from completing all their data transmissions successfully. The dense configuration achieves a slightly larger throughput for the calls routed via the FI than in the sparse configuration: this corresponds to more bandwidth in use. Nevertheless, the power used is smaller: on average, the mobile nodes are located closer to the APs, which decreases the transmission energy required.

Note that the dense FI configuration also increases the throughput of the data transmission that are not routed via the FI. This is because, in comparison to the sparse FI configuration, less transmission capacity is required at the mobile nodes (which are bottlenecks) to relay FI-based transmissions to and from the APs. Both the transmissions routed via the FI and the transmissions not routed via the FI benefit from this additional capacity.

Which FI configuration should the service provider adopt? This question cannot be answered without further economic details – yet it seems unlikely that the marginal increase in data throughput and the savings in energy consumption, which both translate to increased revenue, compensate for the cost of deploying and maintaining almost twice as many APs. This leads to a recommendation of the sparse FI configuration.

5. CONCLUSIONS
This paper has introduced DESMO-J, a framework for discrete event simulation in Java, whose key features include flexibility and joint event- and process-oriented model implementation. As an example of a simulation application besides the traditional areas of logistics such as queue-based production systems and transportation, a MANET simulation is used to present how to design and implement a simulator based on DESMO-J. The essential features of the MANET model, including the underlying economics, the decentralized protocol of data rate allocation and a relatively simple simulation experiment. Further information and more comprehensive experiments can be found in Göbel and Krzesinski (2008b).


REFERENCES


**AUTHORS BIOGRAPHY**

**J. Göbel** holds a diploma in Information Systems from the University of Hamburg, Germany. He now works as scientific assistant and PhD candidate at the Center of Architecture and Design of IT-Systems at the University of Hamburg; research interests focus on discrete event simulation and self-organizing transport networks.

**A.E. Krzesinski** obtained the MSc from the University of Cape Town and the PhD from Cambridge University, England. He is a Professor of Computer Science at the University of Stellenbosch, South Africa. His research interests centre on the performance evaluation of communication networks.

**B. Page** holds degrees in Applied Computer Science from the Technical University of Berlin, Germany, and from Stanford University, USA. As professor for Applied Computer Science at the University of Hamburg he researches and teaches in the field of Computer Simulation as well as in Environmental Informatics.