Proceedings of the 2005 Winter Simulation Conference
M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, eds.

FLIGHT SCHEDULING AT KLM

Peter H.M. Jacobs
Alexander Verbraeck

Department Technology, Policy and Management
Delft University of Technology
P.O. Box 5105, 2600GA Delft
THE NETHERLANDS

Jeroen B.P. Mulder

Department of Decision Support,
Corporate Information Office
KLM, P.O. Box 7700, Luchthaven Schiphol
THE NETHERLANDS

ABSTRACT

We present a specification of the optimization of KLM's flight schedule in this paper using object-oriented simulation services on top of the Java-based distributed simulation environment DSOL. The paper shows the added value of a service-based architecture for linking the simulation of KLM's operations, the operational modules for day-to-day flight optimization, the interfaces to input data, and the output visualization tools. All services are loosely coupled to the other services. The contracts as defined in the interfaces of the services allow for easy extensions of the functionality of for instance the optimization modules, without the need for making changes in other places of the model. Links to external data sources can also be easily provided. In the project, it was clearly demonstrated that the usefulness, usability, and usage of the distributed, service-based architecture for KLM is much higher than earlier implementations of their simulation-based optimization models.

1 INTRODUCTION

Royal Dutch Airlines, or KLM, is the Netherlands' largest airliner and as such one of the largest in Europe; it was founded in 1919 and since then it has experienced constant growth. In the fiscal year 2003/2004, KLM employed 34.529 people and maintained a fleet of 188 aircrafts (KLM 2004). KLM executes a schedule connecting over 400 cities in 85 countries on 6 continents. More than 23 million passengers and some 529,000 tons of freight were transported over their network in 2003/2004 (KLM 2004). KLM has recently merged with Air France; the new airline is the biggest in Europe and number three in the world.

KLM adapts its flight schedule at least four times per year to accommodate changes in demand. Schedules are developed by the Network department, a business unit with a strong commercial focus, whose main interest is to maximize profit by maximizing the number of passengers, flights, and flight connections. Within KLM, Operations Control is the business unit responsible for operating the schedule. Its interest lies in executing a feasible schedule which implies less passengers and fewer flights. A sub department of Operations Control, called Plan Acceptance Management, assesses the feasibility of a new schedule, and reports to Operations Control on whether to accept it. Due to opposing interests of Network and Operations Control, objectivity and rationality in the plan acceptance process are highly valued. To achieve such rationality, the decision support department of KLM was requested to develop a simulation model that would support gaining insight into the operational consequences of a new flight schedule. This simulation model, delivered in 2003, and specified in Arena, is called *OPiuM*, and is used ever since. The name OPiuM comes from the abbreviation OPM for Operational Plan Management, the KLM department name of the client of the project.

This model reflects KLM's philosophy of evaluating individual business units based on their performance in the execution of sub-processes. The process of an individual plane is divided into four sub-processes, which are referred to as *process building blocks* and are presented in Figure 1. A department responsible for the execution of a sub-process is called a capacity service provider. KLM's management made capacity service providers individually responsible for measuring service time distributions, for their building block.

All flights of a schedule are simulated in OPiuM. Disturbances in OPiuM are the result of the difference between a value drawn from a statistical distribution and a scheduled process time. The statistical distributions used in OPiuM are based on the service time distributions, provided by the capacity service providers. Whenever there is a disturbance in the simulated schedule, OPiuM optimizes the remainder of the schedule by evaluating a number of potential measures. Such measures include accepting the disturbance, swapping planes and canceling the flight. Penalties, awarded to all these measures, are used to evaluate the remainder of the schedule.

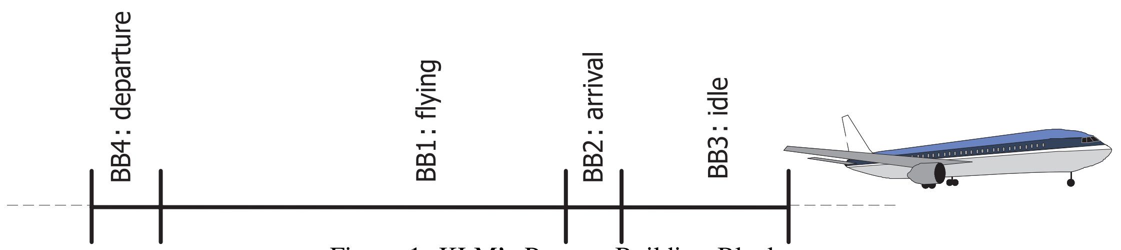


Figure 1: KLM's Process Building Blocks

Although the current specification of OPiuM in Arena is considered to be a great success - managers and employees must be constantly warned not to over trust the outcome of the model - KLM believed there was added value to be obtained from a renewed specification of the OPiuM model in DSOL. DSOL is a Java based, open source, simulation suite developed by Delft University of Technology (Jacobs, Lang, and Verbraeck 2002).

2 WHY A RENEWED SPECIFICATION IN DSOL?

The challenges noticed by KLM with respect to the usefulness, usability and usage of their current specification of OPiuM forms the topic of this section.

One of the most noticeable problems was that while the optimizer requires operations research strategies to deliver the required output, Arena is clearly not designed for such algorithmic specifications. As a result OPiuM is mostly specified in Microsoft Visual Basic and Arena is, besides providing an event calendar and thus the simulated time, almost circumvented as a simulation language.

Another problem is Arena's model execution environment. Although OPiuM was developed by highly educated simulation experts, the users of OPiuM, i.e. employees of Plan Acceptance Management, are not highly educated in this area, and while they are experts in the domain of flight scheduling, the execution and development environment of Arena is considered to be too complex and as such lacks usability qualities for those that must use it.

Due to Arena being non-standard software at KLM, and because of the complicated structure and deployment of Arena licenses, in addition to the specific versions of the libraries used to accomplish the interaction between Arena and Microsoft Visual Basic, the Decision Support department developed OPiuM and by default became responsible for its installation, maintenance and service for end-users.

Arena does not provide any support for the collaborative model specification, nor does it provide support for distributed, concurrent model execution.

Arena is difficult to integrate with external distributed data sources. Since capacity service providers must individually publish service times, support for integrating information would be very much appreciated.

An overall conclusion with respect to the current specification in Arena was that KLM foresees that usage of OPiuM may well shift to a more operational mode, in which daily problems are evaluated, if the current specification has not reached its limits with respect to scalability and performance.

3 CONCEPTUALIZATION

We will now introduce the conceptual diagrams of the case, to provide insight into the processes of executing a flight schedule. If a flight is delayed, Operations Control can take several measures to optimize the remainder of the schedule. To evaluate the consequences of a potential measure, a value function is assigned to each measure, expressing a sanction, in minutes delay, for a particular flight, at a particular moment in time. The following measures are used in the model.

Swapping two fleetlines: one of the measures is to swap two fleetlines. A fleetline is the sequence of flights scheduled during the schedule period to be performed by one aircraft. Swapping fleetlines implies that the scheduled flights for a particular plane for the remainder of the rotation are swapped with those scheduled for an alternative plane. A rotation is the sequence of flights from Schiphol to Schiphol. Usually this involves two flights, or legs, but sometimes it involves three or more. The swap measure is illustrated in Figure 2. If a delay causes time pressure, and thus an inevitable delay, on the next flight, swapping a rotation between two planes may well be a rational measure to take. Swaps are only performed at Schiphol airport and preferably between two planes of the same sub type. This is called a *registration swap*. Only if such plane is not available, are planes of another type considered. The sanctions for this measure are expressed in minutes and based on the differences in passenger capacity of the aircrafts involved and on the crew related cost consequences. There is a base sanction plus a leg-sanction for every swapped leg, except for registration swaps, where only the base sanction applies.

Using a reserved aircraft: this is an aircraft that is kept idle for a longer period of time (several hours to days). A reserved is used to schedule a standby aircraft and crew, which are only used whenever a problem in the schedule occurs. When a flight must be executed but the originally

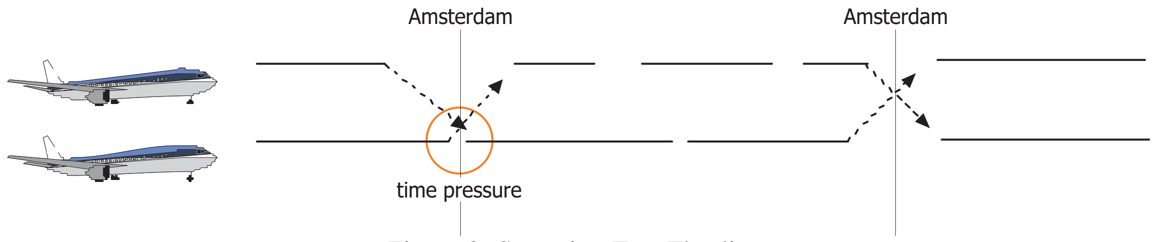


Figure 2: Swapping Two Fleetlines

assigned aircraft is unavailable a reserve aircraft may be used. The sanction and value function of a reserve are equal to the swap measure.

Reducing maintenance time: maintenance can be shortened by approximately 15% of the regular maintenance time by increasing the amount of assigned resources, i.e. engineering staff and equipment. The sanction for this measure depends on the type of airplane.

Canceling a flight: a cancel-measure implies that an entire rotation is canceled, i.e. it will not be executed. The sanction is evaluated per leg and is very high, which is not surprising since cancelation is the most radical solution.

Operations Control is responsible for the execution of flights and thus for the optimization of the schedule. A sequence diagram of this `executeFlight` operation is presented in Figure 3.

The following characteristics of this sequence diagram are worth discussing. One, on the execution of a flight, an external optimizer is requested to optimize the schedule. Two, this optimizer is defined as an interface. By using an interface, we adhere to the principle of design by contract and as such emphasize on a loosely coupled relation, i.e. we place the emphasize on the replaceability of a particular implementation. A domain specific, proprietary, external service can easily be used to do the actual optimization. Three, the actual execution is implemented in the `Aircraft` which adheres to the principle of separation of concerns.

Although the use of an external, well validated optimizer is encouraged, a reference implementation of the interface is presented in Figure 4. The following characteristics of this implementation are worth discussing. One, the optimizer has no relation with the DSOL simulator. This ensures that the algorithm used to optimize a flight schedule can be shared between the simulation model, i.e. OPiuM, and the information systems supporting daily flight execution. The value of this openness is that the optimizer can be used for the simulation model, i.e. OPiuM, as well as for the information systems used by Operations Control in their daily work. It furthermore opens the door of using OPiuM in a more operational daily environment. Two, measures follow a transaction model, which supports an initial try and a final perform. Three, both value functions assessing the consequences of individual measures and the reference time providers, i.e. `NormTimeProvider` are interfaces that emphasize on the replaceability of a particular implementations.

The actual execution of a flight is the responsibility of an aircraft, see the sequence diagram of Figure 3. The `Aircraft` class is presented in Figure 5. The following characteristics are presented in this Figure. One, static parsing methods in both the `Aircraft` and the `SubType` directly parse the *Flash* schedule presented in Figure 6 (*Flash* is the name of the KLM proprietary software used to develop flight schedules.). Two, the `State` class represents the states reflecting the sub-processes of an plane. The sequence is specified in the `nextState` operation.

KLM's schedule is conceptualized in Figure 7. The `Fleetline` holds a number of *FlightLists* each holding a number of *Flights*. DSOL's event package enables flights to fire events whenever a delay is incurred, and to compute the performance indicators of a schedule, DSOL's statistical objects are asynchronously subscribed to these events.

4 SPECIFICATION

We present the specification of the OPiuM model in DSOL in this section. We merely introduce those aspects that are relevant for understanding the extent to which the DSOL specification of OPiuM differs from the prior OPiuM model specification in Arena.

4.1 Input Specification

DSOL supports input and output of the Flash file format to ensure that the usability of OPiuM and Flash for the end-user is not affected.

Although capacity service providers publish service times individually and autonomously, Decision Support is currently obliged to download such information and to export it into Microsoft Access files which can be read by Arena. The DSOL specification of OPiuM supports both the Microsoft Access, and any remote text-based proprietary document format. As a consequence updated information can be automatically downloaded over ftp, http or nfs. Java's JDBC and JNDI standards further result in support for remote relational and directory-based databases. This support is illustrated in the model definition of the exper-

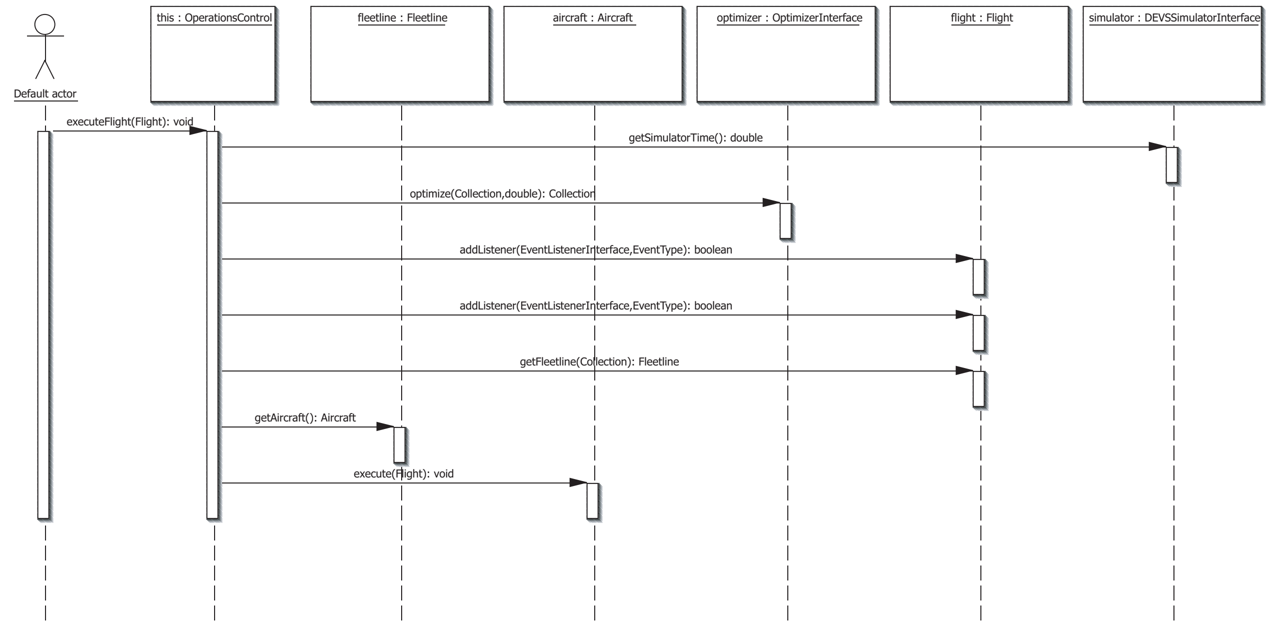

Default actor
this : OperationsControl
fleetline : Fleetline
aircraft : Aircraft
optimizer : OptimizerInterface
flight : Flight
simulator : DEVSSimulatorInterface
executeFlight(Flight): void
getSimulatorTime(): double
optimize(Collection,double): Collection
addListener(EventListenerInterface,EventType): boolean
addListener(EventListenerInterface,EventType): boolean
getFleetline(Collection): Fleetline
getAircraft(): Aircraft
execute(Flight): void

Figure 3: Sequence Diagram of the `ExecuteFlight` Operation

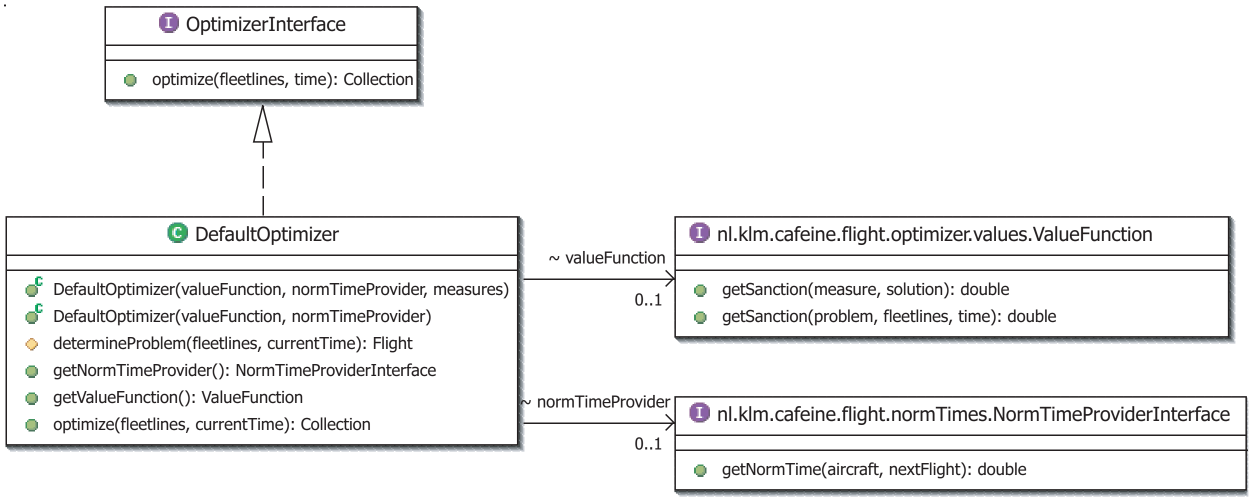

OptimizerInterface
optimize(fleetlines, time): Collection
DefaultOptimizer
DefaultOptimizer(valueFunction, normTimeProvider, measures)
DefaultOptimizer(valueFunction, normTimeProvider)
determineProblem(fleetlines, currentTime): Flight
getNormTimeProvider(): NormTimeProviderInterface
getValueFunction(): ValueFunction
optimize(fleetlines, currentTime): Collection
~ valueFunction
0..1
nl.klm.cafeine.flight.optimizer.values.ValueFunction
getSanction(measure, solution): double
getSanction(problem, fleetlines, time): double
~ normTimeProvider
0..1
nl.klm.cafeine.flight.normTimes.NormTimeProviderInterface
getNormTime(aircraft, nextFlight): double

Figure 4: A Reference Implementation of the `OptimizerInterface`

iment definition presented in Figure 8. The connection arguments for connecting to a remote database, e.g. Microsoft Access or Oracle, are supplied in the text based `database.properties` file.

4.2 Output Specification

The DSOL specification of OPiuM supports a number of output modes. Besides charts and Microsoft Excel files, the DSOL specification of OPiuM supports animation on top of a geographical information system, i.e. Gisbeans (Jacobs and Jacobs 2004). Several characteristics of DSOL's animation capabilities are presented in Figure 9.

DSOL supports multiple, remote animation screens which are concurrently subscribed to the simulation model. This validates the value of DSOL's strong support for remote asynchronous communication. It furthermore validates the value of a loosely coupled, pull-based approach to animation.

The geographical information system, Gisbeans, supports layered based rendering; detailed information is only shown at detailed zoom levels. The animation of the planes is state dependent, i.e. the actual delay is represented under the plane. Clicking on a specific plane enables users to drill down and actually to get operational insight into the status of the plane, e.g. the position and the estimated delay.

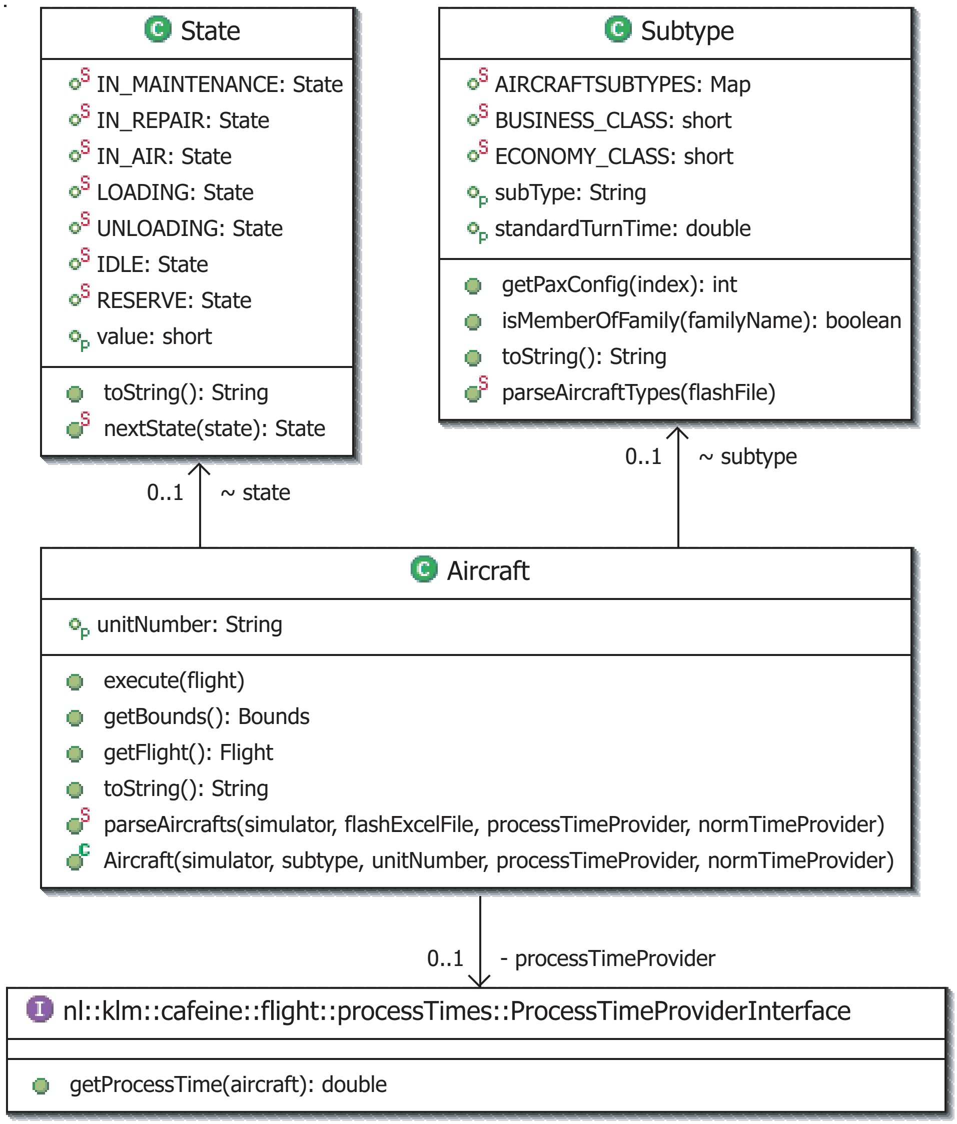


Figure 5: Class Diagram of `Aircraft`

5 CONCLUSIONS

We will now evaluate the current specification and the extent to which we have accomplished the requirements presented in section 2. The first requirement was to overcome the circumvention of the simulation language to specify the algorithms of schedule optimization. Without further elaboration, we may well conclude that using the Java programming language for the specification of the model fulfills this requirement. The open architecture of DSOL furthermore prevents a doubtful boundary between parts that are available to designers and parts that are shielded. DSOL for example provides full access to the event list of the discrete simulator.

A second requirement was to distinguish end-users from simulation model builders and to target specific support environments for their tasks. In DSOL, model builders are supported with state-of-the-art software engineering tools such as an integrated development environment, e.g. Eclipse, and a Java project management tool, e.g. Maven. End-users are supported with a web-portalled environment which is tailored for specific usage.

A third requirement was to deliver a suite of KLM standard information system services that allowed KLM's IT department to take responsibility for and control installation and end-user support. Since KLM has standardized all in-house developments in the Java programming language, we may well conclude that our specification fulfills this requirement.

The fourth requirement was that the decision support department should be supplied with tools that could meet a need to support collaborative model specification. This case taught KLM how to use a concurrent versioning system to synchronize changes on a central model repository.

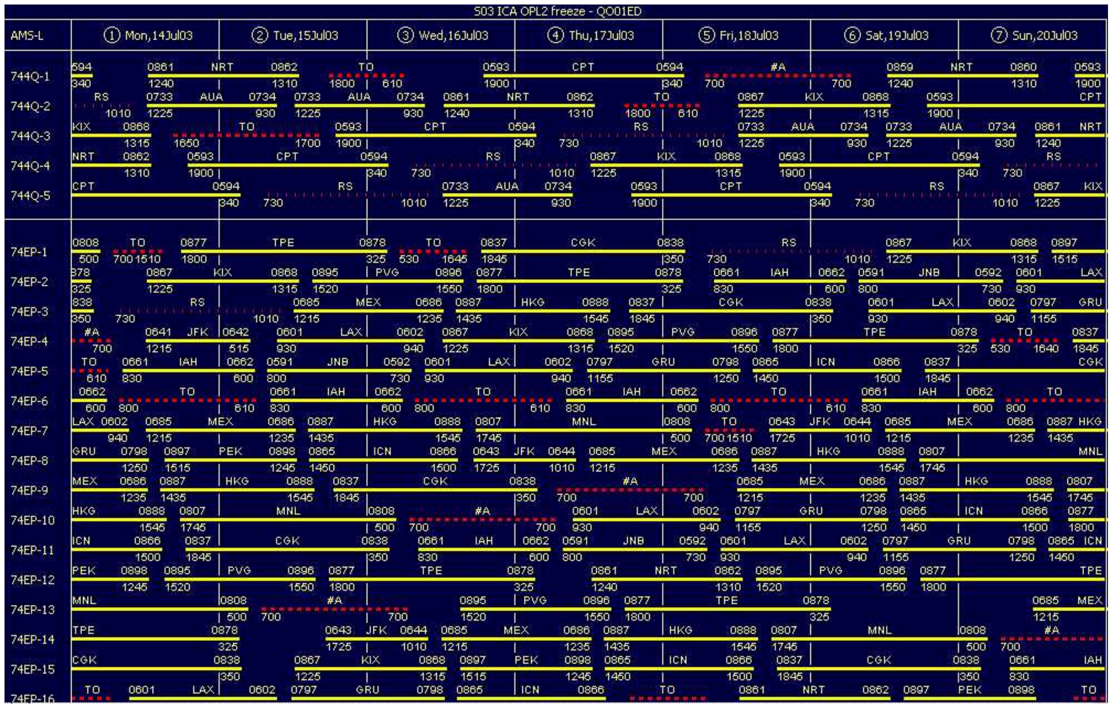


Figure 6: A Schedule as Viewed with Flash

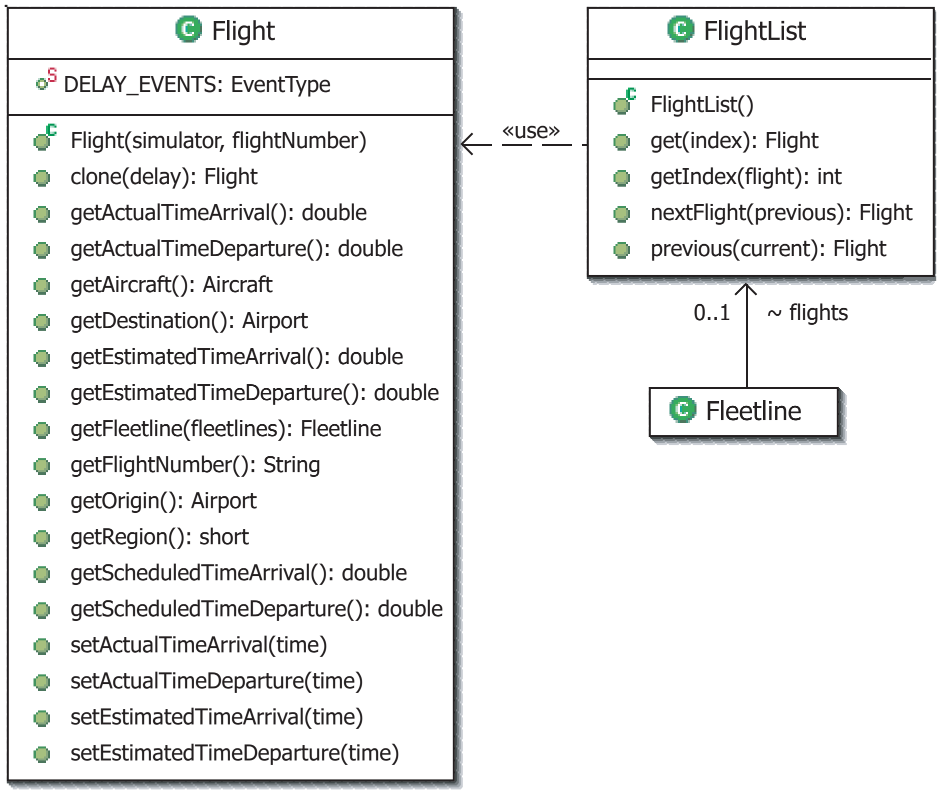
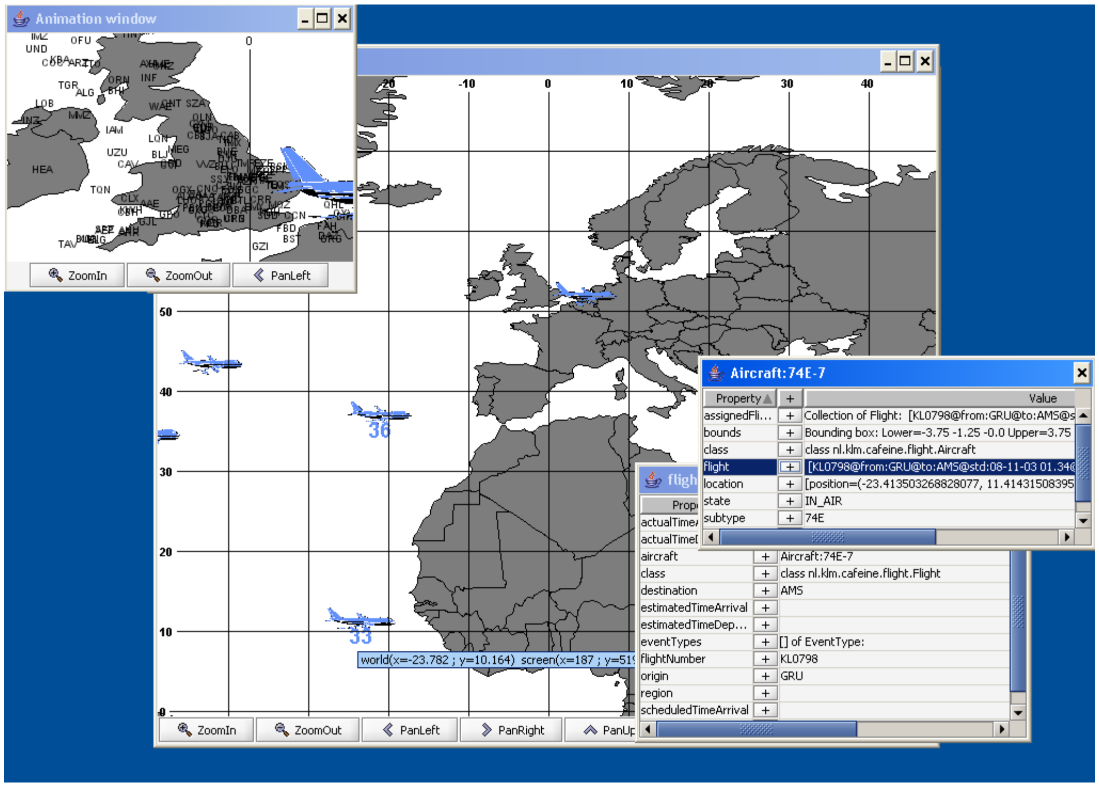

Flight
DELAY_EVENTS: EventType
Flight(simulator, flightNumber)
clone(delay): Flight
getActualTimeArrival(): double
getActualTimeDeparture(): double
getAircraft(): Aircraft
getDestination(): Airport
getEstimatedTimeArrival(): double
getEstimatedTimeDeparture(): double
getFleetline(fleetlines): Fleetline
getFlightNumber(): String
getOrigin(): Airport
getRegion(): short
getScheduledTimeArrival(): double
getScheduledTimeDeparture(): double
setActualTimeArrival(time)
setActualTimeDeparture(time)
setEstimatedTimeArrival(time)
setEstimatedTimeDeparture(time)
«use»
FlightList
FlightList()
get(index): Flight
getIndex(flight): int
nextFlight(previous): Flight
previous(current): Flight
0..1
~ flights
Fleetline

Figure 7: Class Diagram of `Fleetline`

Java is a general purpose programming language designed for a networked, distributed environment. The DSOL specification of OPiuM makes use of libraries to connect to external databases, in-memory databases, directory services

```
<model>
    <model-class>nl.klm.cafeine.model.Model</model-class>
    <class-path>
       <jar-file>http://www.simulation.tudelft.nl/airfields.jar</jar-file>
       <jar-file>ftp://anonymous:klm@ftp.klm.com/pub/data.jar</jar-file>
       <jar-file>file:/C:/development/cafeine/world.jar</jar-file>
    </class-path>
</model> <properties>
    <property key="DATABASE_PROPERTIES" value="/database.properties"/>
</properties>
```

Figure 8: Experimentation in the DSOL Specification of OPiuM



Property	+	Value
assignedFli...	+	Collection of Flight: [KL0798@from:GRU@to:AMS@s
bounds	+	Bounding box: Lower=-3.75 -1.25 -0.0 Upper=3.75
class	+	class nl.klm.cafeine.flight.Aircraft
flight	+	[KL0798@from:GRU@to:AMS@std:08-11-03 01.34
location	+	[position=(-23.413503268828077, 11.41431508395
state	+	IN_AIR
subtype	+	74E

Prop		
actualTime		
actualTime		
aircraft	+	Aircraft:74E-7
class	+	class nl.klm.cafeine.flight.Flight
destination	+	AMS
estimatedTimeArrival	+	
estimatedTimeDep...	+	
eventTypes	+	[] of EventType:
flightNumber	+	KL0798
origin	+	GRU
region	+	
scheduledTimeArrival	+	

Figure 9: Animation in the DSOL Specification of OPiuM

and remote files over any network protocol, e.g. ftp or http. A diagram showing the fulfilment of the requirement to link to external data sources is presented in Figure 8.

A general over reaching conclusion is that, from our perspective, the DSOL implementation has yet to reach its limitations with respect to scalability and performance. The possibilities to *not* animate, and to distribute execution over multiple processors, support our strong conviction that the DSOL specification is well equipped to serve more operational, daily processes, however more research is required to support this.

6 OBTAINING THE SOFTWARE

DSOL is published under the General Public License. More information on the license can be found at `<http://www.gnu.org/copyleft/gpl.html>`. The DSOL project description can be found at `<http://www.simulation.tudelft.nl>` and the software can be downloaded from `<http://sourceforge.net/projects/dsol/>`

REFERENCES

Jacobs, J., and P. Jacobs. 2004. Gisbeans: a Java library for geographical information systems.

Retreived October 21, 2004 from <http://gisbeans.sourceforge.net>.

Jacobs, P., N. Lang, and A. Verbraeck. 2002. A distributed Java based discrete event simulation architecture. In *Proceedings of the 2002 Winter Simulation Conference*, ed. E. Yucesan, C.-H. Chen, J. Snowdon, and J. Charnes, 793–800. San Diego: CA, USA: IEEE: ACM Press. Retreived October 21, 2004 from <http://www.informs-cs.org/wsc02papers/102.pdf>.

KLM 2004. KLM, annual report 2003/2004. Retreived October 21, 2004 from <http://www.klm.com/corporate_en>.

AUTHOR BIOGRAPHIES

PETER H.M. JACOBS is a Ph.D. student at Delft University of Technology. His research focuses on the design of simulation and decision support services for the web-enabled era. His working experience within the iForce Ready Center, Sun Microsystems (Menlo Park, CA), and engineering education at Delft University of Technology founded his interest for this research. His e-mail address is <p.h.m.jacobs@tbm.tudelft.nl>.

JEROEN B.P. MULDER is a consultant at the department Decision Support of KLM's Corporate Information Office. As a Decision Support consultant he designs, develops and applies (new) Decision Technology systems in a scientific econometric, and information technology perspective and judges the integration of these systems in the KLM's IT infrastructure and organization. His email address is <jeroen.mulder@klm.com>.

ALEXANDER VERBRAECK is chair of the Systems Engineering Group of the Faculty of Technology, Policy and Management of Delft University of Technology, and a part-time full professor in supply chain management at the R.H. Smith School of Business of the University of Maryland. He is a specialist in discrete event simulation for real-time control of complex transportation systems and for modeling business systems. His current research focus is on development of open and generic libraries of object oriented simulation building blocks in Java. Contact information: <a.verbraeck@tbm.tudelft.nl>.