



---

## **MicroDoc External Device Framework for Java**

Whitepaper, © MicroDoc® GmbH, Munich, 2000  
07.01.2000

Developing Self Service Terminals with Java

---

## Abstract

Software for self service terminals is different from the usual desktop application. From the experience of building self service systems for more than 5 years we found the following list of requirements:

***Very simple, easy to use user interface***  
***Connectivity to legacy backend systems, usually transactional host software***  
***Consistent modularization and abstraction of business processes***  
***Support for exotic hardware devices (PIN pads, gauges, special printers ..)***  
***Maximum reliability (24 by 7) and error tolerance***  
***Powerful mechanisms for remote service and administration***

Object orientations provides a great foundation for developing self service systems. Platform independence, short development cycles and reliable runtime behavior are arguments for using Java as an implementation language.

MicroDoc External Device Framework for Java (EDF/J) is a component based software toolkit to build self service systems like:

***Info terminals***  
***Ticket terminals***  
***Check-In terminal***  
***Reservation systeme***  
***Shop terminals***

EDF/J supports the integration of special hardware, offers connectivity to host systems of all kinds and provides an architecture to distribute user interface and application logic . EDF/J is designed to integrate with self service standards like CUSS and JavaXFS . The product is targeted to application developers who need to implement business logic for self service terminals and do not want to handle all the inherent complexity of a heterogeneous computing environment. EDF/J offers an interface to Wincor Nixdorf ProView™, a self service network maintenance and administration system.

---

## EDF/J Software Architecture

EDF/J is based on two autonomous components: device framework and service framework. The device framework is used to hide technical details of the connection to self service hardware and provides an easy to use programming model to access printers, PIN pads, gauges, scanners and the like. The service framework is an architecture for distributed applications and allows to employ synchronous and/or asynchronous communications between the client and server tiers.

### Device Framework

The following section outlines the modules of the device framework:

#### Transport Layer

The transport layer abstracts the underlying low level interfaces to physical devices of different types. Depending on the interface, the transport layers uses JNI to access platform specific libraries (DLLs) or IP sockets. EDF/J comes with a standard transport

layer for serial lines (RS 232) and TCP sockets. Other interfaces are available upon request.

### **Protocol Handler**

The protocol handler en/decodes data and control sequences for physical devices before they are passed to the transport layer. The protocol handler is also used to specify the communication mode between host and device. We found that it is possible to abstract protocol handler for specific classes of devices and for specific device vendors. Using the abstract handlers, it is very easy to add an entirely new device to the system. EDF/J comes with protocol handlers for the following devices: Omron Card Reader, IER ATB Printer, Dassault Gate Reader and Siemens Bag Tag Printer.

### **Driver**

The driver implements the logical API for the software device and hides the interaction between transport layer and protocol handler. The driver abstracts the physical interface and the protocol used for communication. Using two different drivers, you could connect the same software device (such as a printer device) via two different interfaces (such as serial or parallel communication lines).

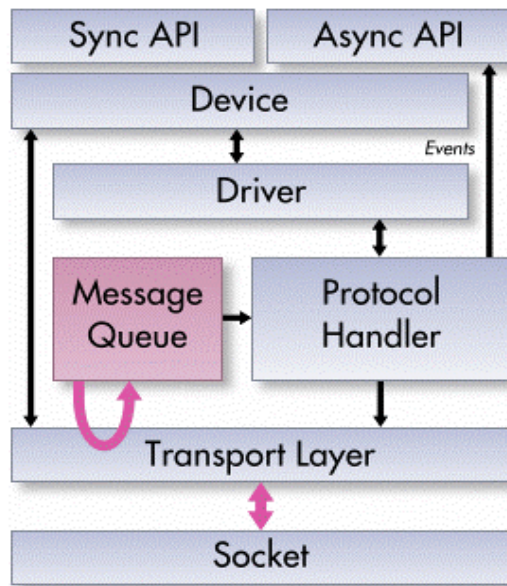
### **Device**

A software device is the software representation of a physical or logical device (i.e. printer or host session). The device implements specific protocols that reflect the external device's capabilities. Software devices accept commands and may send software events when the physical device's internal state changes. An application developer does not need to care about asynchronous notifications from a device, and can use a simple, single threaded programming model (even though the internal implementation of active devices employs multithreading).

### **Device Pool**

The device pool is a facility to administer devices for one or more applications. Devices can be leased from the device pool and must be returned after a specified amount of time. The device pools allows to implement kiosk systems that host a variety of concurrent applications that share the same hardware.

## Device Framework component overview



© 2000 MicroDoc Computersysteme GmbH

## Service Framework

The following components are the building blocks of the service framework:

### Server

The server is a runtime environment for EDF/J services. The server hosts and administers the services and performs service life cycle control. The server also allocates and controls the system resources: device pool and thread pool.

### Services

A service is an implementation of a function request from one or more clients. A service may use one or more devices and is used to aggregate functions for clients on a higher level of abstraction. The service API may be used synchronous or asynchronous. Services can be used locally (within the callers process) or remote (via RMI or CORBA invocation).

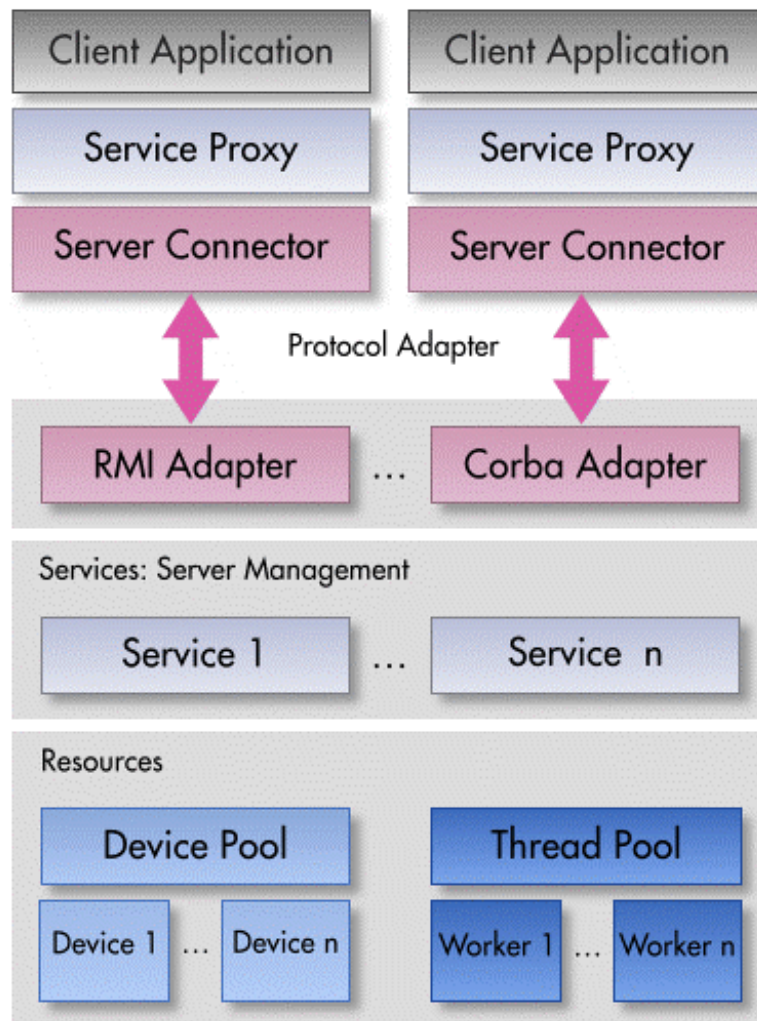
### Daemons

Daemons are special services that provide an API to register callbacks. Daemons run in their own threads and are used to create asynchronous event notifications.

### Server Connector

The server connection abstracts the interface to a distributed networking protocol. This abstraction allows to switch between RMI and CORBA without changing application code. Other distribution protocols can be integrated.

## Service framework component overview



© 2000 MicroDoc Computersysteme GmbH

---

## Integration with Wincor Nixdorf ProView™

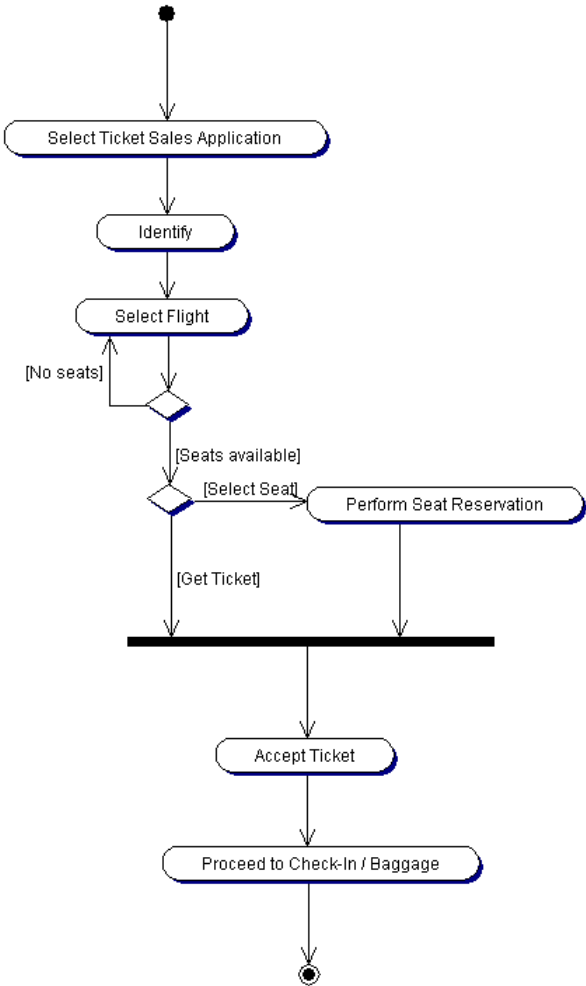
ProView™ is a client/server product for remote administration of self service devices and self service networks. Configuration management and maintenance functions are complemented by a Java agent concept that is used to integrate ProView™ with MicroDoc's EDF/J. ProView™ offers a ready to run solution for centralized and decentralized monitoring of large self service installations.

# Sample applications: MicroDoc Airlines/OOP 2000

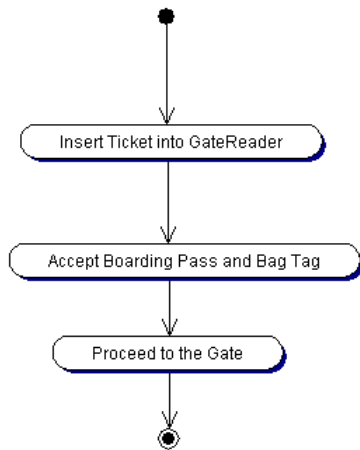
We have implemented a sample application for the OOP2000 conference and tradeshow: MicroDoc Airlines ticket sales and boarding. The software uses EDF/J to control a number of devices and shows how EDF integrates with ProView™.

## MicroDoc Airlines Ticket Sales and Boarding – Customer point of view

The user interface is a standard web browser. The customer has to identify himself and selects: departure airport, destination airport, date of flight and time of flight. In addition, seat reservation is possible. The ticket is printed at the point of sale on an ATB printer.

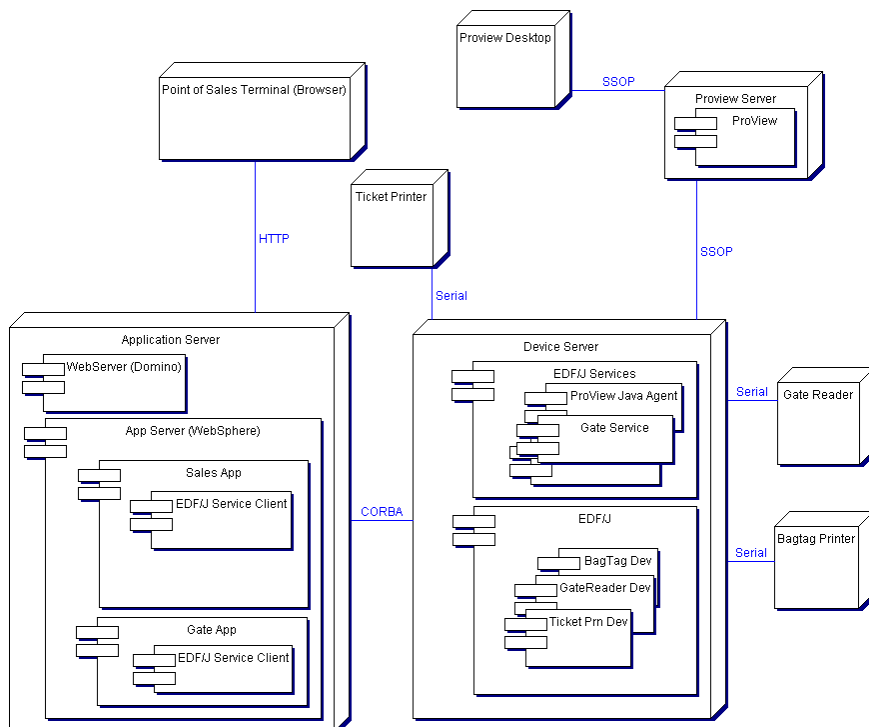


Taking the ticket, the customer proceeds to the self-boarding terminal. After inserting the ticket into the gate-reader (a device that would normally control access to the airplane) a bag tag is printed that can be used as a ticket to win a prize at our OOP booth.



## MicroDoc Airlines Ticket Sales and Boarding – Operator point of view

We integrated ProView™ to monitor and administer our demo sales and boarding application. The following graphic shows the system components:



All devices have been attached to a single device server (for convenience). The device server hosts EDF/J services and communicates via RMI or CORBA. The sales application is implemented as Java servlets that run on an application server. The boarding application is an EDF/J service without a GUI.

The ProView™ server is used to start and stop the applications and logs events and statistics. It communicates with the device server via the SSOP protocol. The ProView™ Desktop is used to visualize the system state and to interactively control the whole system.

## **MicroDoc Airlines Ticket Sales and Boarding – Developer point of view**

The developer of an application faces a number of tasks: Implementing devices, implementing services, implementing application flow control and a user interface. EDF/J allows to separate the tasks to be performed by a team of developers with different skills.

### **Development of devices**

The ticket sales and boarding samples used the standard EDF/J devices. No extra programming is required.

### **Implementation of the services**

Services for ticket sales and boarding have been implemented according to the EDF/J programming model. The sales services is very simple and is used to print the customers personal information and the flight selection on an ATB ticket. The boarding services uses two devices (gate-reader and bag tag printer) and automatically prints the bag-tag when a ticket is inserted in the gate reader.

### **Application flow control and backend systems**

The sales applications features an HTML UI. The pages are created by Java servlets that run within an application server (IBM WebSphere). Flight and ticket data come from a relation database (IBM UDB) and are accessed via Enterprise Java Beans that run in the application server (IBM WebSphere) and use MicroDoc's Java persistence Framework MPF/J.

### **Integration with ProView™**

ProView™ was integrated using the ProView™ Java agent technology. We used existing device templates from ProView™ so no programming was required for the ProView™ server and for the ProView™ desktop. The Java agent was integrated on a service level and can start and stop devices and the whole application. A ProView™ interface will be part of future releases of EDF/J.

---

## **Summary**

EDF/J is a framework for the development of modern, distributable Java applications that make use of external devices (self service terminals, kiosk applications, shopping terminals etc.). The high level of abstraction implemented in the EDF/J APIs isolates the application developer from the underlying technical complexity. The service programming model is a secure base for distributed applications. Java technology and the open structure of EDF/J will help you to create cost efficient applications that are reliable and open for future technologies.

---

# Appendix

## Used software for the sample application

Operating system for the device server: Windows NT 4.0 Workstation, SP 5  
Operating system for the ProView™ server: Windows NT 4.0 Workstation, SP3  
Operating system for the Application/Web Server: Windows NT 4.0 Workstation, SP5  
Operating system for the ProView™ Desktop: Windows NT 4.0 Workstation, SP5  
Operating system for the Sales Terminal: Windows NT 4.0 Workstation, SP5

Internet Browser: Microsoft Internet Explorer 5.0

Application Server: Lotus Domino Server R 5.02a, IBM WebSphere 3.0

Development tools:

IBM VisualAge for Java Version 3.0  
IBM VisualAge for Java Servlet Builder  
IBM VisualAge for Java Domino Access Builder  
Microsoft Frontpage

MicroDoc External Device Framework for Java, Version 1.2.2  
MicroDoc Persistence Framework MPF/J

## MicroDoc Computersysteme GmbH

MicroDoc specializes in the object technology since 1991. Industry customers from banking, finance and travel rely on MicroDoc designs and implementations for mission-critical applications. Exceptional high standards of quality and the employment of cutting edge technology made MicroDoc an internationally acclaimed technology provider. MicroDoc is a licensee for IBM Software Solutions Labs in Raleigh North Carolina, USA and supplies part of the VisualAge for Smalltalk product (Domino Connection). MicroDoc also participates in the development of the IBM VisualAge for Java product as a contractor for IBM Canada

## Copyright Notices

MicroDoc is a registered trademark of MicroDoc Computersysteme GmbH  
Windows, Windows NT, Internet Explorer, Frontpage are registered trademarks or trademarks of Microsoft Corporation  
Lotus, Domino are registered trademarks or trademarks of Lotus Development Corporation  
IBM, WebSphere, DB2, UDB are registered trademarks or trademarks of International Business Machines Corporation  
All other trademarks are property of the respective owners