

UltraLightClient

by Canoo Engineering AG

Reviewed by
Peter Leitner

Remote Swing or server-side Swing – this is the most concise characterization of Canoo's UltraLight-Client library (ULC). ULC offers server-side peer classes for Swing. For each Swing widget, there's a peer ULC class with essentially the same API.

The value added by ULC is the built-in split between client and server: ULC splits each widget into a client part and a faceless server part, and synchronizes these so-called half widgets at runtime.

The result is a client that's rich but thin, an idea that sounds puzzling today, since we associate rich clients with fat clients, and thin clients with HTML-based peer clients.

Minimal Footprint

An important characteristic of ULC is its minimal footprint. Despite the fact that it's a client/server technology, it imposes neither a framework nor an application server onto its user.

All infrastructural tasks are delegated to standard J2EE. The client relies on native Swing, communication is configurable as HTTP(S) or RMI over IIOP, and the server half widgets may run within a servlet or in an EJB container.

Conceptually, ULC is just a smart widget set. Its impact on an application is limited to the presentation layer. Programmers can employ their technology of choice for business objects, per-

sistence, and other software layers. The only constraint ULC imposes is the thin client architecture.

Rich Thin Clients

ULC seeks to combine the benefits of HTML thin client applications and fat client productivity applications. Its starting point is the J2EE architecture with a server-side programming and execution model.

This makes ULC applications conceptually similar to HTML-based J2EE applications: all the business logic and the model of the presentation logic execute on the server. The client is a generic presentation engine, like a

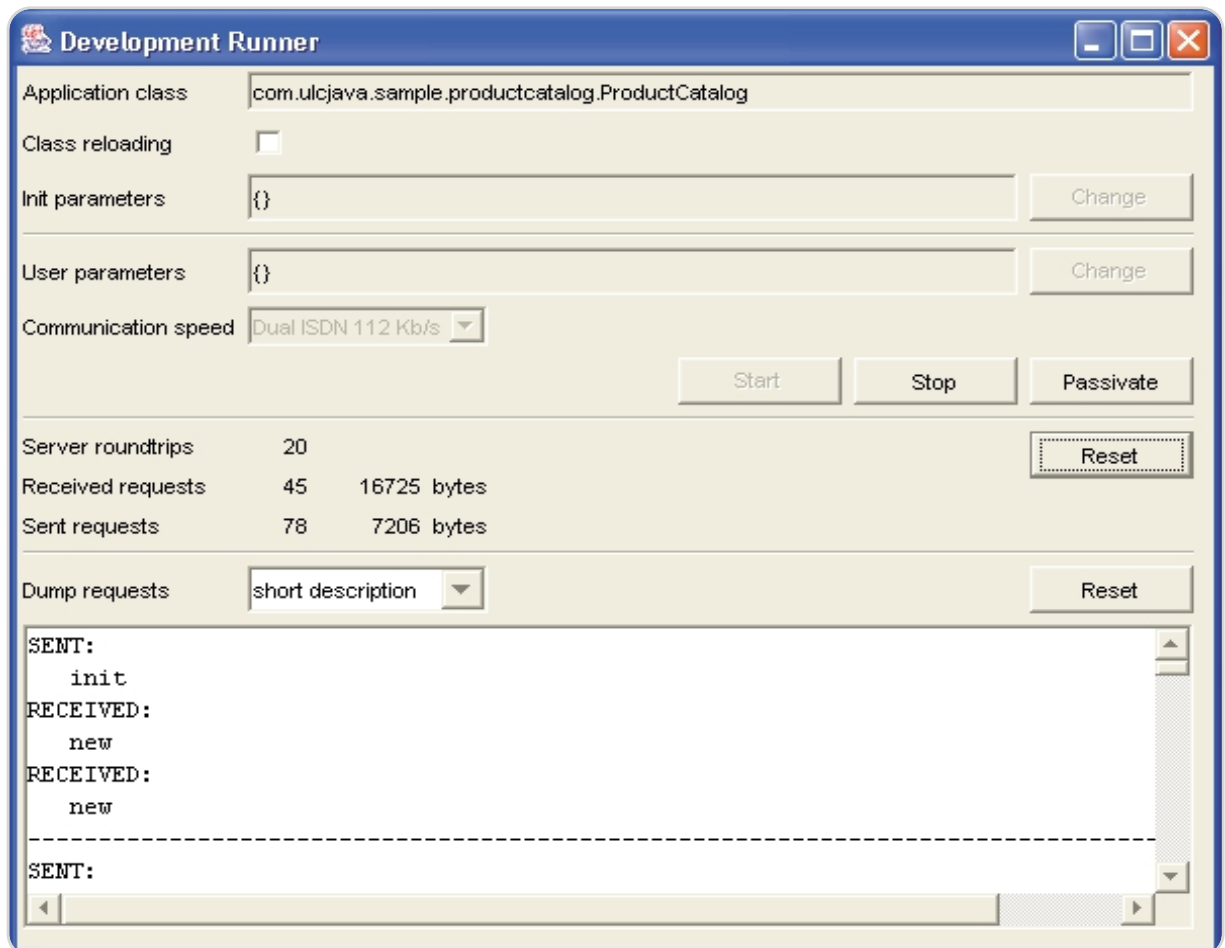


Figure 1 DevelopmentRunner GUI



Peter Leitner is a team leader and senior software engineer at Würth Phoenix. He has 10 years of experience developing object-oriented ERP systems and online/offline sales support systems.

peter.leitner@wuerth-phoenix.com

browser. The difference between a browser and the ULC engine is that the latter handles descriptions of rich graphical user interfaces instead of HTML.

While not new, the concept of rich thin clients has lost none of its appeal. It promises state-of-the-art usability combined with easy manageability and operation.

Let's see to what extent Canoo's ULC lives up to this promise.

Getting Started

Canoo's offering includes:

- The core ULC library, currently v. 5.1.3
- A visual editor for the WebSphere Studio Application Developer IDE, currently v. 1.0 (an Eclipse version is scheduled for Q2004)
- A load and performance testing tool, currently v. 2.2

Installation is simple: you can either extract the distribution archive or use the platform-specific installer for Windows, Linux, Mac OS X, or Solaris. The distribution contains the ULC libraries for the server container and the presentation engine, a 250-page developer guide, a client/server simulator called `DevelopmentRunner` that enables executing applications in a single virtual machine, and the source code for six sample applications.

The best way to start is to run the sample applications, some of which are available as online demos on the Web site. To execute them on your own machinery, either double-click their launch scripts or drop one of the WAR Files into your preferred application server and invoke their start page within your Web browser.

The ULC-Set example is a good point of departure. It provides an overview of all widgets available. Each widget sample is a click away from its source code.

Given the rich set of examples, the familiar Swing API and the comprehensive developer guide, a Java engineer will learn quickly. My experience is that a programmer who knows Swing and the basics of client/server computing will be fully productive after a few days.

IDE Integration

A nice feature of ULC is that it fits into your preferred IDE. The only additional tool you need is the `DevelopmentRunner` that comes with the library.

The `DevelopmentRunner` executes the client and the server in a single virtual machine, shortening the edit-compile-test cycle. Using this tool is simple: just

call it in the main method of your application class:

```
public static void main(String[] args) {
    DevelopmentRunner.setApplicationClass(<Your
    ULC Main Class>);
    DevelopmentRunner.main(args);
}
```

Since you can execute a ULC application entirely within your IDE, debugging and testing is easy. Your IDE's debugger and your favorite tools will all work.

For the purpose of monitoring interaction between client and server, the `DevelopmentRunner` offers a dialog window:

```
DevelopmentRunner.setUseGui(true);
```

This window displays the messages exchanged and allows simulating different bandwidths (see Figure 1). I like this latter option because it enables me to test the real-life behavior of my application within my IDE.

If your favorite IDE is Eclipse or WebSphere Application Developer, use the drag-and-drop visual editor that comes as an add-on product (see Figure 2). This editor corresponds almost exactly to the Swing-based editors available for these platforms. It generates Java and reflects code changes back to the user interface.

The somewhat privileged positioning of Eclipse and WebSphere is also documented by the fact that the `DevelopmentRunner`'s user interface is integrated in their workbench.

Working with ULC

We've been working with ULC for two years now and have developed two applications. One of them is WÜRTHPHOENIX CIS (www.wuerth-phoenix.com), a company information system that is shown in Figure 3. WÜRTHPHOENIX CIS supports profit centers in their financial reporting, planning, and projection. It was created by three developers within 18 months, and has been rolled out as a standalone application and in a hosting datacenter where it's serving over 280 companies worldwide.

The second application we developed is WÜRTHPHOENIX ERP-Basic, an enterprise resource planning (ERP) application for small and medium-sized companies. WÜRTHPHOENIX ERP-Basic supports purchasing, planning, order management, and logistics, including inventory accounting as well as reporting and statistics. Ten developers completed it within 14 months.

For development, we use PCs with

Canoo Engineering AG

Kirschgartenstrasse 7
CH-4051 Basel, Switzerland
Phone: +41 61-228-9444
Web: www.canoo.com

Specifications

Platform: Any platform that supports the Java Runtime Environment 1.2.2 or later

JVM: Java Runtime Environment 1.2.2 or later
Java Application Server: Any J2EE compliant servlet and EJB container

Pricing:

- **ULC Library:** \$1,495 per developer. Includes an unlimited runtime license and all configuration options (deployment in servlet, EJB container, or standalone, with HTTP(S) or RMI/IIOP).
- **Visual Editor:** \$499
- **ULC Load:** \$3,000

Test Platform

Sun JRE 1.2.2/1.3.1/1.4.1/1.4.2.
Windows 2000/XP, Red Hat Linux, and Solaris.
BEA WebLogic, IBM WebSphere Application Server, JBoss, and Tomcat

Windows 2000, running Eclipse 2.2.1, JDK 1.3.1, and ULC's `DevelopmentRunner`. An integration server and testing server round off our setup. They're running Windows 2000 and Linux Red Hat AS 2.1, respectively, with Jakarta Tomcat 4.1.29.

Our experience is that developing with ULC is indeed similar to working with Swing. The benefit as compared to Swing is that you essentially get a J2EE-compliant client/server application for free. The only client/server-related issue you need to remember is that your application runs in a multiuser and multithreaded environment:

1. Avoid using static variables, because they're not thread-safe. Instead, manage objects in your server session, like in HTML/servlet applications.
2. There is the option to communicate between sessions. You can, for example, realize a notification mechanism that propagates data changes to all sessions.
3. When opening a modal dialog window, the thread won't wait as in Swing. As a consequence, set up a listener that reacts to the action closing the dialog.

A positive surprise for us was the scalability of ULC's thin client solution: the built-in minimization of client/server communication and the economic use of memory on the server work well for us.

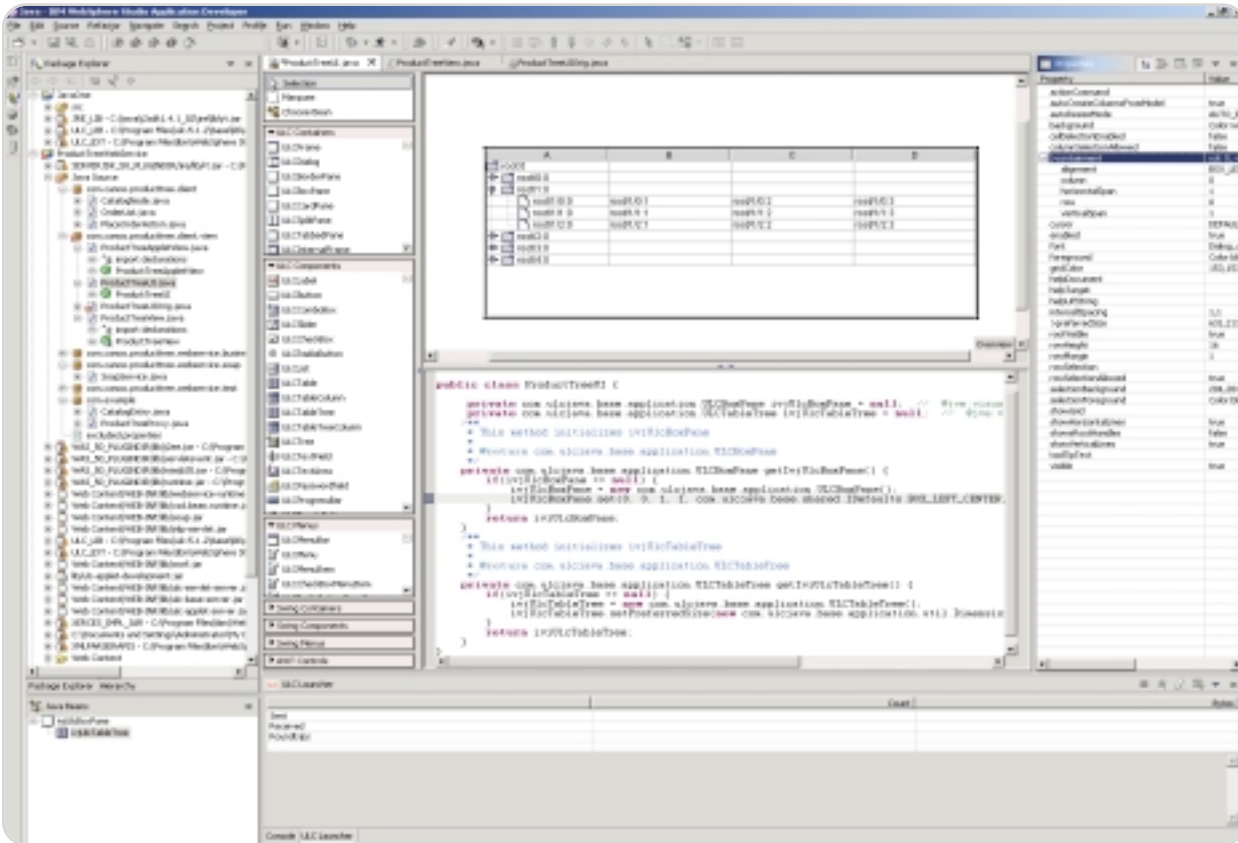


Figure 2 ULC Visual Editor

They enable our users to work over a worldwide VPN or over the Internet, sometimes connecting with slow modem lines, and to use a cost-effective server infrastructure.

A further important point for us is manageability of releases and production. Our applications run on a variety of platforms, for different customers. On the client side, the 350K presentation engine runs under Windows NT, 2000, and XP as an applet and a standalone Java application distributed with JNLP. On the server side, the applications run on Windows 2000, NT, XP, and Linux AS 2.1. ULC's J2EE compliance and the fact that it runs on any JDK from 1.2 upward enable us to support this wide variety with a single code base.

Extending ULC

It's possible to extend ULC to support new kinds of widgets and data objects. Example extensions are special value formatters for text fields, specific enabling strategies that are needed on the client, or widgets supporting graphic functions. There's an extension API that we've used to develop a special table widget. Unfortunately, the code samples of the distribution contain only one such extension. I wish that Canoo included more of them, or, better, offered a community service for the exchange of additions and code snippets.

Performance

ULC employs a number of techniques to minimize round-trips and communication, including local validation, lazy loading, caching, and compression of messages. Yet, the performance of an application never depends on its presentation layer alone. What counts is end-to-end performance, and this has to be measured individually for each application.

Canoo offers a tool for end-to-end testing, called ULC Load. With this tool you can record user interactions and replay scenarios in parallel to simulate any number of users. You can then measure response time and bandwidth. Furthermore, you can export the results for analysis. We found this tool to be simple and useful. It allowed us to test efficiently, and helped in the sizing of the production environment. Regarding performance, in our experience ULC was never the bottleneck in an end-to-end test run.

Summary

ULC is a lean but effective technology: you develop as if Swing widgets were running on the server. You have a homogeneous, server-side programming model, and get a scalable client/server application you can deploy in any J2EE environment.

This is doubtlessly one of the most efficient ways to realize rich client busi-

ness applications, in particular if you can share the J2EE infrastructure with HTML-based applications, thereby eliminating the need for developing a separate solution for functions like security, logging, load balancing, or monitoring.

There are, of course, limitations. If an application needs more than the standard widgets offered by Swing, there is no out-of-the-box solution. For such cases ULC needs to be extended, typically by integrating third-party widgets or components. ☺

Snapshot

Target Audience: Software developers

Level: Intermediate to advanced

Pros:

- Simplifies rich client development for client/server apps
- Small footprint, optimized client/server communication
- Deploys easily in many configurations
- Standards-based (J2EE, J2SE)
- Integrates with existing Web infrastructure
- Attractive pricing; no runtime license fees

Cons:

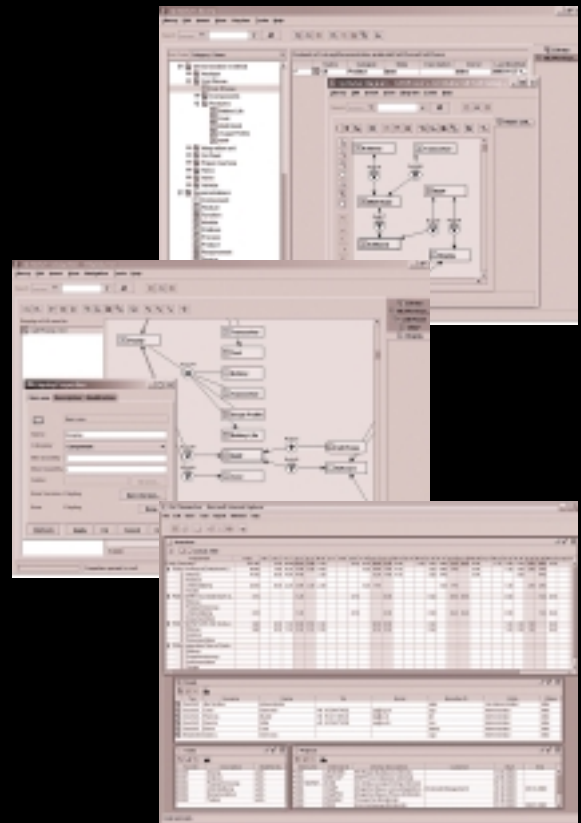
- Few out-of-the-box widgets beyond those of Swing
- Use of third-party graphics libraries possible but requires integration

ULC

Rich Thin Clients for J2EE

UltraLightClient offers
a server-side API to Swing,
providing rich GUIs
for J2EE applications.

- **Server-side programming model:**
develop scalable web applications for thousands of users
as simply as stand-alone Swing applications.
- **Superior security:**
no application code is executed on the client, nothing
is stored in a browser cache.
- **Application deployment on server:**
a lean Java presentation engine on the client serves
any number of applications.
- **Pure Java library:**
use your favorite IDE and get add-on tools for visual editing,
client/server simulation, and load/performance testing.



canoo.com

Canoo Engineering AG

<http://www.canoo.com/ulc/>

Download your free trial today!