PowerBuilder to Java (PB2J)

Thin Client J2EE Migration

White Paper

Contents

Executive Information

This document describes MainTrend’s Solution for PowerBuilder to Java Conversion. Its purpose is to provide general information and understanding of the conversion process and JDataPanel Framework as a basis for this solution and a typical, general methodology that can be applied to application conversion projects utilizing JDataPanel Framework.

For many years PowerBuilder was one of the most serious and robust RAD tools for Client/Server and n-tier development. Software firms invested in PowerBuilder frameworks. Billions of PowerBuilder code lines have been written by thousands of trained developers. Reliable high-performance PowerBuilder applications are deployed worldwide. Enterprises invested in a PowerBuilder infrastructure assuming that they were investing in a platform that would remain vibrant for many years. However the forces of the market thought otherwise. PowerBuilder is now a niche player in the application development tool market. Even PowerBuilder's attempts to make inroads in the WEB development market were not very successful. The number of new installations is diminishing. PowerBuilder professional personnel are becoming scarce.

But there are many enterprises with deployed active PowerBuilder applications. These enterprises must maintain their current systems. Every system in production has to deal with dynamic changes and enhancements. Enterprises are at a point where they have to make a strategic choice: to continue building applications in PowerBuilder where there is no future or to move to another platform. Remaining with PowerBuilder causes no upheavals but insures the widening of the legacy gap. Moving to another platform is probably strategically correct but there are factors to be considered. One major consideration is an enterprise's investment in legacy applications and the significant cost of a new application development. Another consideration is how to ensure a rapid and smooth migration path from one infrastructure to another without disturbing critical systems, specifically how to integrate the new with the old.

Nowadays more and more organizations are urged to move their PowerBuilder applications to a modern thin-client environment. There are 3 choices, which can be considered: face lifting, manual rewriting from scratch and automated conversion.

Face lifting uses the existing PowerBuilder application as a back-end, and therefore does not really move it to a modern environment.

Rewriting PowerBuilder applications from scratch is a very costly, time consuming process and may lead to the long learning curve for the end-users. Another significant point is that legacy applications and their maintenance changes are not always well documented. Even when an original system analysis document exists, the risk of business knowledge loss is very high.

Automated conversion with manual correction of some script parts as a replacement methodology saves all the investments in business knowledge and opens the way to maintenance and further development in a new modern environment.

Migration Methodology

Moving to another platform requires a number of factors to be considered. One major consideration is an enterprise's investment in its legacy applications and significant cost of the new application development. Another consideration is how to ensure a rapid and smooth migration path from one infrastructure to another without disturbing critical systems. The more valuable the encapsulating an organization's business logic software is, the more complex is the process of its evolution. Maintaining legacy software business logic is the important part of an organization's knowledge storage. Smooth and reliable transformation requires preserving all of the needed business logic.

The time to start migration is determined by the moment the new requirements can not be implemented with the old technology. The main goal is not only to migrate into a new environment, but to have an opportunity to implement new technologies. The right migration strategy also means changing the software ideology to meet new requirements. Therefore, just emulating the old legacy environment with the new one is a wrong decision.

More and more organizations are looking to move their PowerBuilder applications away from the thick client, proprietary environment provided via PowerBuilder, into the n-tier thin client environment provided by Java. In order to accomplish this, some organizations attempted to manually rewrite their PowerBuilder applications to Java. Some other organizations used various conversion tools, currently available on the market.

Each approach has its pro’s and contra’s.

Rewriting from scratch as a replacement methodology can produce very effective resulting code, which exactly matches the target environment (if the development team is sufficiently qualified, if not, it is better not to touch the application at all). On the other hand, it is a very costly and time consuming process. It also may lead to the long learning curve for the end users and for the maintenance team. And the significant point is that legacy applications are not always well documented. Even when an organization has the original system analysis document, not all the maintenance changes may have been documented, so the risk of business knowledge loss is very high.

Automatic conversion as a replacement methodology is much faster, much cheaper, and also ensures that all the business knowledge is preserved. Also the resulting application is similar to the original one, and the learning curve for the end users and maintenance team is not as long as with a “brand new” application. On the other hand, the resulting code is generated automatically. It is good, but not necessarily optimal for the target environment and for the specific application.

PowerBuilder as a migration source is the ideal case for **the mixed approach**. A central element in PowerBuilder environment is DataWindow, a very powerful object, concentrating almost all the data processing of a PowerBuilder application and almost all the visual representation of the application’s data. The PowerBuilder DataWindow object is one of the main reasons for the success that PowerBuilder has achieved as a software development tool. Usually in a PowerBuilder application DataWindow objects consume about 60% of the application building efforts. So we decided to include DataWindow objects in the 100% automatic part of the conversion, and to make all the other PowerBuilder objects’ conversion as flexible as possible, automatically generating the target Java classes and allowing manual adjustments when needed.

Such migration concept unites advantages of both approaches:

- The original application is automatically reversed engineered.
- All the DataWindow definitions are automatically converted to JDataPanel XML definitions and corresponding Java classes (JDataPanel framework is a MainTrend's product for J2EE environment; it is described more in details in Appendix A).
- All the other PowerBuilder objects are parsed and converted to corresponding Java classes.
- All the Java classes are generated in exact match with the original inheritance tree.
- All the attributes and methods of the original classes are automatically converted.
- The bodies of all the methods in the generated Java classes, in addition to the converted code, as an option can contain the original PowerScript code, included as a comment. This allows manual adjustment or even rewriting for such scripts and getting the effective resulting code, which exactly matches the target environment. Also such approach preserves the business knowledge from being dropped or omitted and allows to use re-trained original maintenance team.

PowerBuilder to Java (PB2J) is a software engine toolset and set of methodologies that take an application coded in PowerBuilder and generate Java / Xml application source code with reasonable need for a programmer intervention. Java as a target environment allows maximum platform flexibility. Java has proven itself to be a superior technology as a modern business application tool for highly reliable applications. Moving to Java is a strategic decision for an enterprise, a decision to make another step into the future. An enterprise that decides to migrate to this modern development environment will use PowerBuilder to Java (PB2J) as its conversion tool.

Target Environment

The conversion target is a full thin client environment based on the modern J2EE and AJAX methodologies.

The new architecture is a pure thin client (“browser”) solution with a J2EE container and web server on the server side; the application is packed as a J2EE servlet running within a J2EE container; database access and some business logic are provided via J2EE components. All the data access layer (DAL) and presentation layer (GUI) object definition XML files may also be stored on the central web server.

The server side can be any J2EE container / Java Application Server, running on any Java supporting operating system (IBM WebSphere, BEA WebLogic, Oracle Application Server, Tomcat, JBoss, Sybase EAServer, etc.).

The resulting application requires no client installation, except of a browser, with no operating system limitations. Currently supported browsers are Internet Explorer 8.0 and higher, and Firefox 30.0 and higher. It is recommended that the same browser software version will be installed on all the client workstations and all the browser instances on the client workstations will have the same settings (security, JavaScript etc.).

Application changes, which are made to the XML definition files – database access definitions and GUI definitions, are available immediately (if “always reload” option used). Business logic changes, encapsulated within application server components, are available immediately after installation. Business logic and GUI framework changes, encapsulated within the J2EE Servlet, are available immediately after the Servlet container restart. The database is accessed from the application server components.

This architecture provides centralized application management and does not require any additional software to be installed on the end-user workstations.

Migration Process and Solution Architecture

Generally the migration process has the following steps:

1. Detailed analysis, choosing of the target server-side platform
2. Verification and fixing of the source application
3. Export of all the PowerBuilder objects of the migrated application
4. Reverse engineering of the original application
5. Code generation
6. Manual rewriting of the application classes' method's code
7. Preparing test environment
8. Database migration (if a new database platform is chosen)
9. Unit testing, including required database connection for the data access layer objects
10. External links migration (if any)
11. Code integration and thin platform adaptation. Application restructuring according to the thin client conversion model.
12. Database integration and required changes according to the chosen database platform
13. General graphical design, fine tuning of the resulting GUI in line with customer standards
14. Application integration testing and required fixing
15. User acceptance
16. Building the production environment and putting the new application to production
17. Trainings for the customer's developers

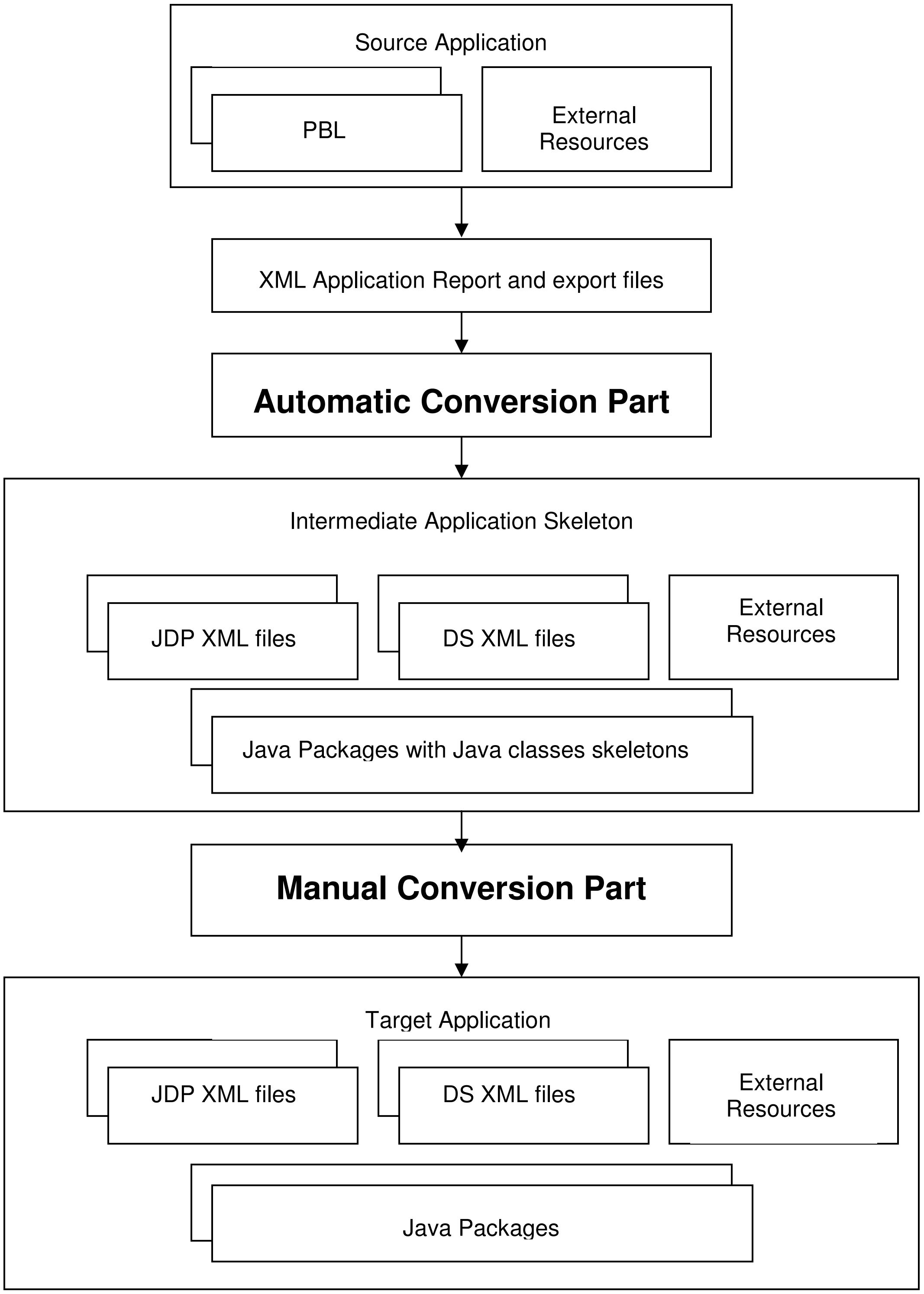
The conversion itself is covered by steps 3 through 6 and uses PowerBuilder Conversion Suite. PowerBuilder Conversion Suite consists of these parts:

- J2EE-based JDataPanel framework that allows future flexible and reliable development in the new Java-based environment, and which also includes an independent product for rich Form/Report design – MainTrend's JDataPanel Designer. The framework supplies all the needed functionality that PowerBuilder DataWindow objects have, including database access, forms and reports. JDataPanel can be also used to create new Java applications, not just to convert existing PowerBuilder applications.
- PowerBuilder Converter itself, which translates each PowerBuilder application into a set of Java and XML modules.
- Conversion and development methodologies, including best practices for the most effective manual completion and enhancement of the resulting applications.

We use PowerBuilder "library export" functionality to obtain all the sources of PowerBuilder objects for the specific PowerBuilder library set. After the reverse engineering stage, the code generation stage comes. All the DataWindow definitions are automatically converted to JDataPanel XML definitions and corresponding Java classes. And all the other PowerBuilder objects are parsed and converted to corresponding Java classes according to the original inheritance tree. The manual modernization part includes significant legacy and thick-client features (e.g. OLE automation, "goto" etc.), GUI tuning according to the customer's requirements and database access tuning.

PB2J toolset uses sophisticated concept of multi-level embedded parsers and template-based code generators. This concept allows use of different parsers for different parts of script and objects' types, and improves the conversion performance.

Visually the conversion process can be depicted in this manner:



The generated objects are grouped into the following packages:

- "dal" – XML-definitions of the data access layer objects (DataStores, the database-related part of DataWindow objects)
- "app" – all the Java classes
- "presentation" – all the XML-definitions for the GUI objects (GUI-related part of DataWindow objects)

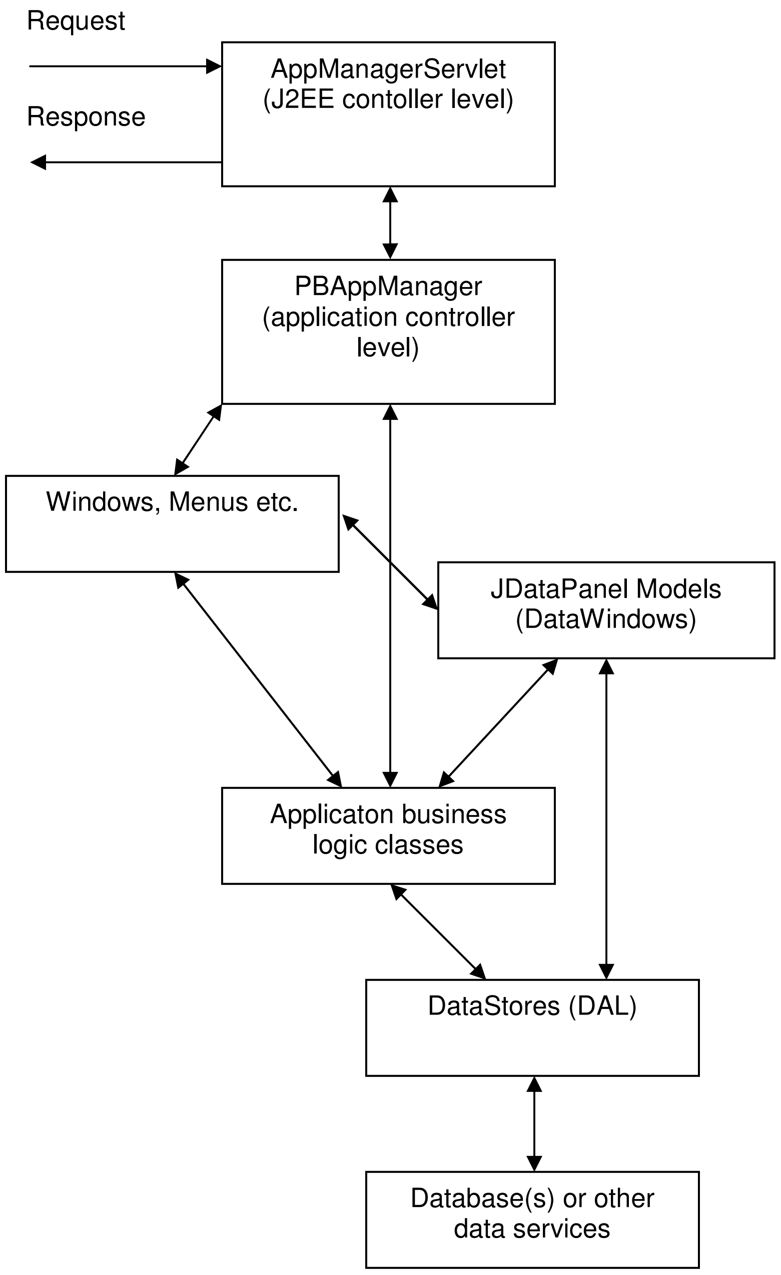
PowerBuilder's DataWindow object unites in one object two layers – presentation layer and data access layer. Sometimes even business logic is embedded into a datawindow (e.g. a computed field referencing a global function). It was tolerable for rapid application development concept, but does not meet modern ideology of layers separation. Therefore during the conversion each DataWindow object is converted into three JDP framework objects: a DataStore XML definition (data access layer object), a JDataPanel XML definition (presentation layer object), and a Java class for embedded logic. Thus the more close correspondence to n-tier requirements is achieved.

The presentation layer object definitions and data access layer object definitions (XML files) can be stored with the resulting Java application, or can be deployed on a separate web server.

The generated application is deployed with the "thin" framework as follows:

```
[webapps]
  [... the application's directory
    [css]
    [html]
    [images]
    [js]
    [jsp]
    [META-INF
      MANIFEST.MF
    ]
    [WEB-INF
      [classes
        [The application’s Java classes / XML files by package name:
          [dal]
          [app]
          [presentation]
        ]
        [net]
          [maintrend - MainTrend's engine Java classes]
        ]
      [lib
        [all the needed libraries, including Oracle JDBC driver]
      ]
      web.xml
    ]
    index.jsp
    login.jsp
    Configuration.xml
  ]
]
```

Logically the converted application works within a J2EE container as a dispatcher servlet, which provides all the needed for the thin client environment functionality:


Request
Response
AppManagerServlet (J2EE contoller level)
PBAppManager (application controller level)
Windows, Menus etc.
JDataPanel Models (DataWindows)
Applicaton business logic classes
DataStores (DAL)
Database(s) or other data services

The conversion services, pricing model and estimation

We offer an automated conversion service that can lift PowerBuilder applications to the Java/XML technologies while preserving business logic base and user interface investments.

Most of the work can be done remotely, and manual part can be divided between MainTrend and a customer or a third party conversion partner, to be as much as possible convenient to all the parties.

There can be different models of the conversion projects - from full conversion to different "reverse engineering" stages, automatically generated script can be used by customer's developers as a tool for business logic capturing, etc. Business model can also vary - from fixed price projects to hourly based consulting. All the mixed type models are also available.

Summary

This White Paper describes the MainTrend's path for successful PowerBuilder migration into new advanced J2EE environment.

This migration methodology provides:

- Rapid, accurate and valid way for organizations to migrate PowerBuilder applications to Java
- Low assimilation costs for the new application
- The same productivity and more, in a new modern environment.
- No limitations for the target server-side platform: any Java-supporting platform will fit
- Much lower cost and faster solution for the legacy applications' migration path.
- Flexible open-platform development environment. The applications can be further developed by our GUI designing tools and framework alongside and integrated with the most cutting-edge commercial development tools
- Code reuse enabled environment and reduced maintenance costs.
- Seamless integration with modern technologies and new approaches.

Reducing the conversion process significantly, PB2J frees organizations up from the translation process so they can continue to do what they do best — remain dedicated to running their core business functions instead of dedicating themselves to one usually lengthy migration project.

By working with the MainTrend's team of specialists, organizations receive the added benefit of applying our experience and advanced expertise to their unique and often complex requirements.

Appendix A:
JDataPanel (JDP) Framework

JDataPanel Framework is a central element of MainTrend's programming environment. It is both a base for all the conversions (PowerBuilder-to-Java, PowerBuilder etc.), and also the independent framework for building J2EE applications. When developed, the purpose of the framework was not to write a PowerBuilder engine using Java as a programming language, rather to build a general framework that included PowerBuilder functionality and PowerBuilder's DataWindow as a special case. The purpose was to give Java developers a powerful and flexible Java and XML based environment for building Data Access Layer and Presentation Layer components.

JDataPanel Framework consists of the following parts:

- JDataPanel components (Presentation layer)
- Data manager components (Data access layer)
- Application manager components (Control layer)
- Graphical designers (JDataPanel Builder and DataStore Builder)

JDataPanel is a universal XML-based user interface description:
- It has both "wide" and "deep" architectures: correlates with intuitive concept of object with attributes ("wide") and sub-objects ("deep").
- It has built-in integration with the Business Logic and Data Access Layers together with readiness for layers separation.
- It is a powerful form / report framework.

User Interface Description Structure

```
<?xml version="1.0" encoding="utf-8"?>
<group name="Main" ... (other attributes) >
  <textgroup name="t1" fontfamily="Courier New" x="0" ...>
    <text name="t1_1" container="t1" ...>Hello!</text>
  </textgroup>
  <group name="Header" datasource="DS1" ...>
    <text .../>
    <field name="date" datafield="indate" datatype="Date" ...>
      <editcontrol name="JDateField".../>
    </field>
    <group ...>
      <field .../>
      <text .../>
    </group>
  </group>
</group>
```

JDP framework supports an MVC (Model – View – Controller) design pattern even in thin client environment (Renderer + browser as "View", JDP Model as "Model" and JDP commands as "Controller").

DataStore part of the framework is used as a “Data Model” part of the “Model” in MVC pattern. DataStore framework is also a universal data access layer solution, and can be used independently from JDataPanel part of the JDP framework.

DataStore objects have their xml representation, which looks as follows:

```
<?xml version="1.0" encoding="utf-8"?>

<datastore name="EXAMPLE_DSD" clearOnRetrieve="true" comment=""
deleteAllowed="true" insertAllowed="true" retrieveLimit="0"
trimEndOnRetrieve="false" updateAllowed="true" updateFields="CODE,
DESCRIPTION" updateKey="CODE" updateSourceName="EXAMPLE_TBL"
definition="select EXAMPLE_TBL.CODE, EXAMPLE_TBL.DESCRIPTION from
EXAMPLE_TBL EXAMPLE _TBL order by EXAMPLE_TBL.CODE asc" adapter="1">

  <field name="CODE" dataType="String" sourceAlias="EXAMPLE_TBL"
sourceFieldName="CODE" sourceName="EXAMPLE_TBL" updateable="true"/>

  <field name="DESCRIPTION" dataType="String"
sourceAlias="EXAMPLE_TBL" sourceFieldName="DESCRIPTION"
sourceName="EXAMPLE_TBL" updateable="true"/>

</datastore>
```

Flexibility of the DataStore framework is based on the encapsulation of the data access functionality with the “adapters” concept. Each type of the data storage can be accessed with the appropriate adapter:

- SQLAdapter -> JDBC -> Database
- XMLAdapter -> XML files
- InMemoryAdapter

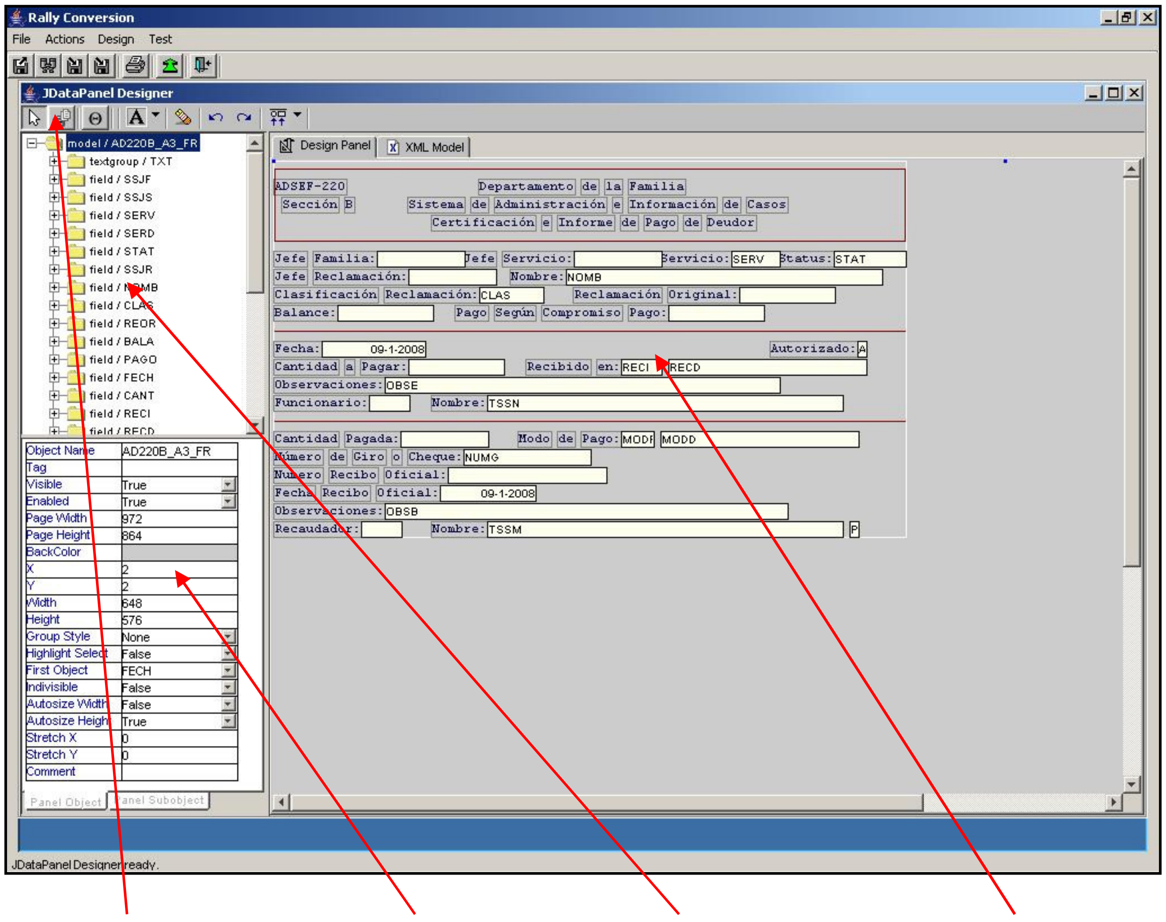
The adapters’ concept is open and any needed additional adapter can be easily developed.

Application manager components are the base for the building thin client solutions for converted from different sources applications, or for the applications which are built from scratch with JDP framework.

These components can be easily inherited and extended to suite all the needs of the chosen target platform.

To increase the development productivity and to help the developers in working with JDP framework, the Graphical designers (JDataPanel Builder and DataStore Builder) can be used:

Here is the example of the JDataPanel (form), opened within JDataPanel Builder:



model / AD220B_A3_FR
- textgroup / TXT
- field / SSJF
- field / SSJS
- field / SERV
- field / SERD
- field / STAT
- field / SSJR
- field / NOMB
- field / CLAS
- field / REOR
- field / BALA
- field / PAGO
- field / FECH
- field / CANT
- field / RECI
- field / RECD

Object Name	AD220B_A3_FR
Tag	
Visible	True
Enabled	True
Page Width	972
Page Height	864
BackColor	
X	2
Y	2
Width	648
Height	576
Group Style	None
Highlight Select	False
First Object	FECH
Indivisible	False
Autosize Width	False
Autosize Height	True
Stretch X	0
Stretch Y	0
Comment	

ADSEF-220 Departamento de la Familia
Sección B Sistema de Administración e Información de Casos
Certificación e Informe de Pago de Deudor

Jefe Familia: Jefe Servicio: Servicio: SERV Status: STAT
Jefe Reclamación: Nombre: NOMB
Clasificación Reclamación: CLAS Reclamación Original:
Balance: Pago Según Compromiso Pago:

Fecha: 09-1-2008 Autorizado: A
Cantidad a Pagar: Recibido en: RECI RECD
Observaciones: OBSE
Funcionario: Nombre: TSSN

Cantidad Pagada: Modo de Pago: MODI MODD
Número de Giro o Cheque: NUMG
Numero Recibo Oficial:
Fecha Recibo Oficial: 09-1-2008
Observaciones: OBSB
Recaudador: Nombre: TSSM P