KUMARAN SYSTEMS

COBOL/CIC/VSAM TO J2EE/DB2 A LEGACY TRANSFORMATION

A White Paper

TABLE OF CONTENTS

KUMARAN SYSTEMS

COBOL/CICS/VSAM to J2EE/DB2

Abstract

Currently, most applications are being developed for or migrated to the web. With this scenario in mind, Kumaran Systems Inc. ventures into the research and development arena to migrate the COBOL application to the J2EE Framework. This paper describes the preliminary work done to achieve this objective.

Introduction

Java 2 Enterprise Edition (J2EE) has become the industry standard for web applications. Applications migrated to J2EE Framework are able to be deployed on any environment without making the typical tedious and time consuming changes. This allows users to utilize the application whenever and wherever it is needed. Not only does this offer a new level of convenience, but it also allows for simple and cost-effective administration. NxTran provides a cost- and time-effective solution for migrating legacy systems to the J2EE Framework.

The Requirement

The evolution of web technologies in recent years presents a compelling reason for companies to consider moving their legacy COBOL Customer Information Control System (CICS) application to the web.

What are some compelling reasons that companies should consider moving their legacy applications?

Maintenance Cost: There is a substantial hardware and software cost involved when running a mainframe.

Application Maintenance: Mainframe applications are built using legacy development environments such as CICS, and VSAM files. Hiring programmers with the knowledge to maintain these applications is a costly investment.

Kumaran Solution

Typical mainframe applications are comprised of COBOL/CICS programs, JCL, and VSAM cluster files. CICS defines the user interface. The business logic is embedded inside the COBOL application and business data is stored in the VSAM cluster files.

Migrating the mainframe application to J2EE involves two phases:

- **VSAM to DB2 or any other RDBMS data conversion**
- **COBOL/CICS to J2EE application conversion**

Kumaran's NxTran migrates CICS-based applications to J2EE-complaint frameworks such as EJB and Struts. The tool aims at minimizing any post-conversion tasks that may need to be carried out by the user after the process of migration, by incorporating a plethora of features into the product in terms of the new environment.

VSAM to DB2 Data Conversion

This section compares VSAM with DB2 and analyzes the need to migrate to DB2. The following illustrates the data conversion methodology that Kumaran utilizes to convert data from VSAM data sets to DB2 and how Kumaran's NxTran tool automates various stages in the process. Though the target database that is used in this white paper is IBM's DB2, this process can be applied to any other industry-standard RDBMS.

VSAM Structures Versus DB2 Structures

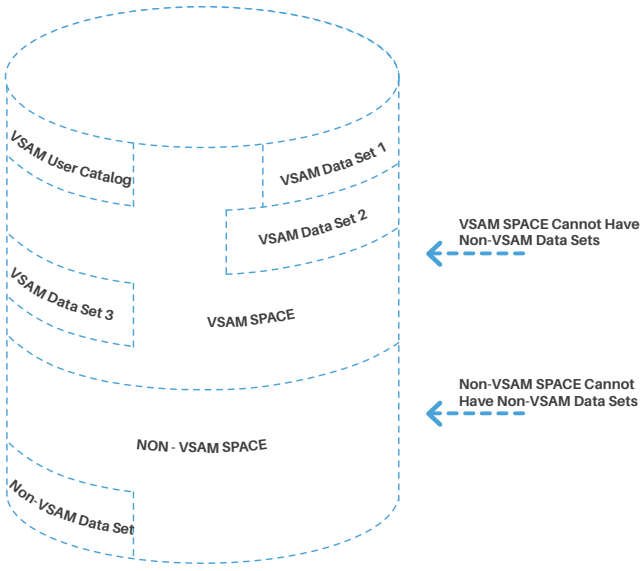
Data Structure for VSAM

VSAM and Non-VSAM data sets can be cataloged in the VSAM Master catalog. User data sets are cataloged in VSAM user catalogs. The user catalog is cataloged in a master catalog.

VSAM space can have two types of objects in it:

- VSAM Catalog
- VSAM data sets, including base clusters and alternate indexes

VSAM space on a disk can be defined by using Access Method Services. VSAM space can have one or more VSAM data sets in it.



KUMARAN SYSTEMS

Data Structure for DB2 UDB

These components comprise the database structure for DB2 UDB:

Containers

A container is a physical storage device. A directory name, device name, or file name can identify it. A container is assigned to a tablespace.

Tablespace

An IBM DB2 UDB database comprises one or more logical storage units called tablespaces.

Objects

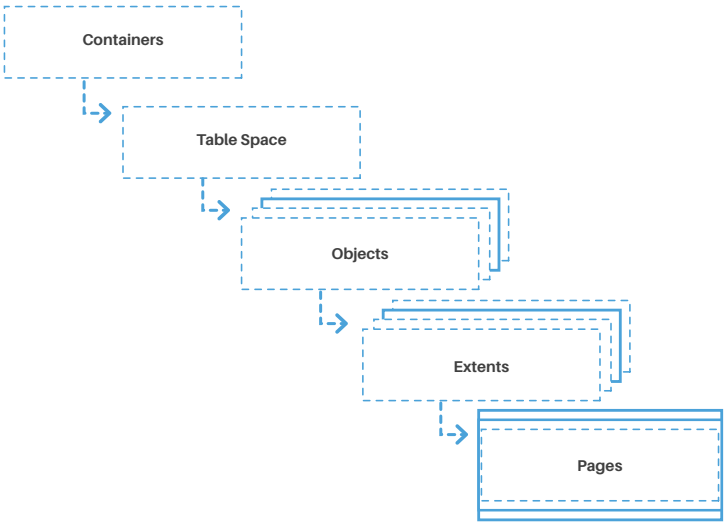
The level of logical database storage above an extent is called a segment. A segment is a set of extents that have IBM DB2 UDB stores data in data Blocks or pages. One data block or page is assigned a specific number of bytes of physical database space on disk.

Extents

An extent is a specific number of contiguous data blocks that are allocated for storing a specific type of information.

Pages

IBM DB2 UDB stores data in data Blocks or pages. One data block or page is assigned a specific number of bytes of physical database space on disk.



Why VSAM to DB2?

VSAM is a file access method, whereas DB2 is a RDBMS, with all the inherent benefits of a DBMS. The main advantage of using an RDBMS is to impose a logical and structured organization on the data. An RDBMS delivers economy of scale for processing large amounts of data because it is optimized for such operations.

Additionally, using an RDBMS provides a central store of data that can be accessed by multiple users, from multiple locations. Data can be shared among multiple applications, instead of new iterations of the same data being propagated and stored in new files for every new application. Most flat-file approaches are designed to be accessed by a single user or a single process at a time.

Central storage and management of data within an RDBMS provides these benefits:

- Data abstraction and independence
- Data security
- Locking mechanism for concurrent access with these properties: atomicity, consistency, isolation, and durability (ACID)
- An efficient handler to balance the needs of multiple applications using the same data
- The ability to swiftly recover from crashes and errors
- Robust data integrity capabilities and simple access using a standard API
- Uniform administration procedures for data

A RDBMS offers the ability to provide many views of a single database schema. A view defines logical data independence, protection from changes to the logical structure of data migration scenarios, what data the user sees and in what manner the data is displayed. The RDBMS provides a level of abstraction between the conceptual schema, which defines the logical structure of the database, and the physical schema that describes the files, indexes, and other physical mechanisms used by the database. Users function at the conceptual level. For example, the user may query columns within rows of tables, instead of having to figure out how to access data by employing the different types of physical structures used by RDBMS to store data.

When a RDBMS is used, systems can be modified efficiently and effortlessly whenever business requirements change. New categories of data can be

added to the database without disruption to the existing system. With DB2, for example, adding a new field is as simple as issuing an ALTER statement to add the new column to the table. Performing a similar task in VSAM is much more difficult, especially if the file does not have any unused filler area at the end.

An RDBMS provides a layer of independence between the data and the applications that use the data. In other words, applications are insulated from how data is structured and stored. The RDBMS provides two types of data independence:

- Logical data independence – Protection from changes to the logical structure of data
- Physical data independence – Protection from changes to the physical structure of data

Migration Scenarios

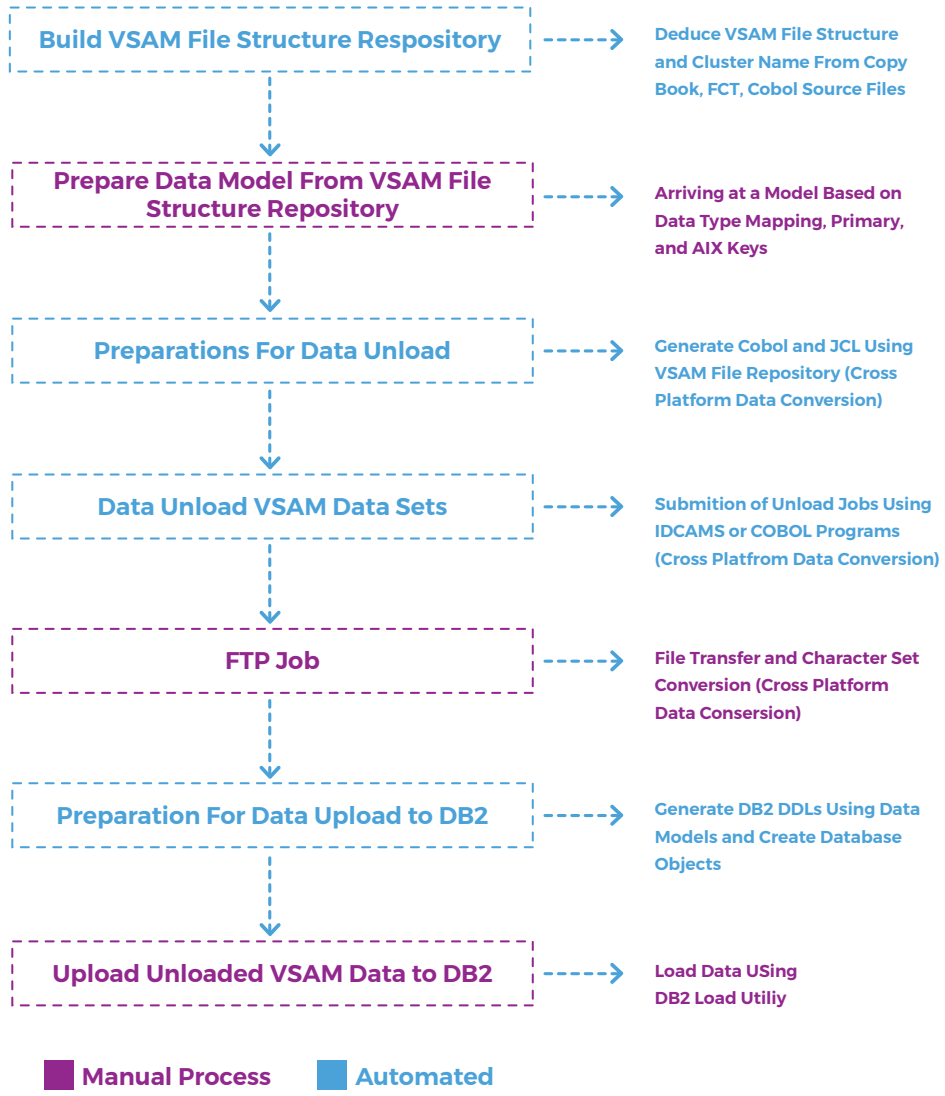
CASE 1 – COBOL programs accessing VSAM (Batch Processing)

In this case, VSAM file structures are either defined in COBOL programs in FILE SECTION of DATA DIVISION or in COPYBOOK that will be copied into the FILE SECTION alternatively during compile time.

CASE 2 – CICS/COBOL accessing VSAM (Online Applications)

In this case, VSAM file structures are not defined in the CICS/COBOL program. The structure needs to be deduced by referring to associated resources like FCT (File Control Table)

VSAM to DB2 Data Conversion



KUMARAN SYSTEMS

1. Build VSAM file structure repository
For every VSAM data set Kumaran's NxTran needs to find the associated record structure. A repository of such details is maintained by the tool in Extensible Markup Language (XML) and will be used as an input for subsequent steps in the conversion process.

The approach to find record structures and VSAM clusters vary from CASE 1 to CASE 2.

CASE 1 Approach – File structures are parsed from either the COBOL source file or the COPYBOOK file or both. COBOL programs use logical file names for file IO operations and the logical name is assigned to the device name. The VSAM cluster or data set name is not referred to in the COBOL source directly. The association between the device and the VSAM cluster or data set is defined in JCL. To determine the cluster name for the file structure the tool will also need the JCL program to resolve the indirection made in the COBOL source and update the structure repository.

CASE 2 Approach – Data independence is the concept of a program being independent of the structure of the database or the data access methods. The CICS file control provides data independence to application programs so that the application programmer need not worry about such data-dependent COBOL parameters or JCL as any of these:

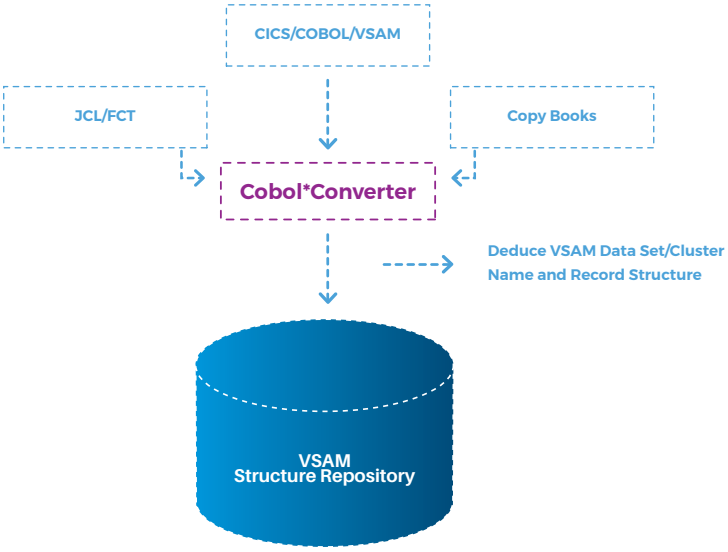
- INPUT-OUTPUT SECTION
- SELECT statement
- FD statement
- OPEN/CLOSE
- JCL

The following external structures will hold various metadata information pertaining to the VSAM data set that is used by the CICS program.

- File Control Table (FCT) holds the mapping information from VSAM cluster or data set to data source name.

- Symbolic Map File holds the WORKING-STORAGE variables that are used as input or output buffer for the screens variables or CICS.

The tool uses various external data structures like the FCT to deduce the cluster name and its associated structure using COBOL source files as inputs to drive the tool operations and update the VSAM file structure repository.



2. Prepare Data Model for VSAM Clusters
The VSAM File Structure repository built by Kumaran's NxTran is used to arrive at a data model that defines the relationships among entities and data constraints. The tool manages the deduction of VSAM data set record type definitions to the maximum extent. Sometimes, due to ambiguities arising out of redefinitions, manual intervention may be necessary. NxTran uses the data type mappings in Appendix A, to arrive at the column data types for the VSAM record types.

Typically, the data model is normalized to the Third Normal Form (3NF) where every VSAM file set becomes a db2 table with the base VSAM base cluster key becoming the primary key and the copybook fields into db2 columns.

Type Mappings
NxTran uses the record descriptions in the FD section of COBOL programs to deduce the record data definition. Kumaran's NxTran requires that the FD descriptions be provided in detail for every record. This is one of the primary requirements for a successful conversion. Kumaran's NxTran is capable of identifying the full FD description from several programs within the given application, provided that all the programs contain the total FD description. For example, consider these descriptions within two COBOL programs:

```
Program-1: ...
FD emp-file.
01 emp-rec.
        05 eno pic 9(5).
        05 ename PIC X(10).
        05 filler PIC X(20).
...
Program-2: ...
FD emp-file.
01 emp-rec.
        05 eno pic 9(5).
        05 filler PIC X(10).
        05 dno PIC 9(4).
```

NxTran can deduce the correct data definition that is written in SQL DDL for DB2.

CREATE TABLE emp(eno INTEGER, ename CHAR(10), dno INTEGER).

Some systems utilize directories to sort data. For example, when the personnel data is associated with different areas, (i.e., India, China, and Canada), it could be organized into the three data files:

/personnel/India.dat
/personnel/China.dat
/personnel/Canada.dat

For such systems, a facility has been provided to define an additional field. For example, AREA CODE CHAR(30) in the target record schema and merging the data into one table. Subsequently, the primary index of the new records will be altered to contain the additional field, and all future search statements on the record will be programmed regarding the new field.

Multi-Schema Definition

COBOL files with multiple record (01) entries will be broken down to a relevant number of record schemas. For example, consider this file:

```
FD order-record.
01 order-header.
        05 rec-type pic x.
        05 order-num pic 9(5).
        05 customer-id pic 9(6).
....
01 order-line.
        05 rec-type pic x.
        05 order-num pic 9(5).
        05 line-num pic 9(3).
```

The file will be broken down into two record schemas in the target database, namely order-header and order-line schemas. The indexes associated with the file will be lined up with the new record schemas.

Array (Table) Handling

COBOL allows the definition of fields as single or multi-dimensional arrays. All single and multi-dimensional fields will be split into two files. For example, consider this file:

```
Program-1: ...
FD personnel-file.
01 personnel.
        05 id pic 9(5).
        05 forename PIC X(10).
        05 surname PIC X(20).
        05 address PIC X(20) Occurs 3 times.
```

It will be converted to the following 2 record types:

```
Record: personnel,
Fields:
id Integer,
forename char (10),
surname char (20).

Record: personnel_mx
Fields:
id Integer,
indx integer,
address char (20).
```

On the program side, the first element of address field is still accessed as personel.address[1] - the database wrapper classes will access the variable as
SELECT personnel_mx WHERE id = personnel.id AND indx = 1;

Redefinitions

Redefinitions are often utilized to meet one of these two objectives:

- Reduction in memory utilization of the application
- Use of different data organization to simplify or enhance the programming code

Redefinitions of data sections are not supported in Java or within any ANSI SQL database. The redefinitions are handled by NxTran by reducing the fields to the lowest common denominators and replacing all subsequent entries to the lowest common denominator. For example in programs with such data:

```
01 personnel.
        05 id pic 9(5).
        05 name PIC X(30).
        05 detailed_names REDEFINES name.
                07 forename PIC X(10).
                07 surname PIC X(20).
```

In this case, all references to the name will be replaced by forename and surname. This technique applies to all data sections including the FILE and WORKING-STORAGE sections.
If a data entry redefines parts of its structures in different, non-compatible forms, that is, where no assumption can be made to the nature of the lowest common denominator, then there can be two options:

- The tool can be programmed to ignore the particular redefinition in instances when the purpose is reduction memory utilization.
- In other cases, manual intervention is required to clarify the final layout of the record structure.

3. Preparations for Data Unload
For cross-platform data conversion, the tool generates COBOL and JCL source files by using the VSAM file repository that was built in the previous process. The generated COBOL source will READ the VSAM file sequentially and WRITE the Physical Sequential file.

4. Data Unload from VSAM data sets to Physical Sequential files (Flat Files)
For cross-platform data conversion, submitting the JCLs generated in Step 3 unloads data.
For the same platform conversion, the tool unloads data using IDCAMS (REPRO) with inputs from the VSAM structure repository.

5. Flat file transfer
(for cross-platform data conversion)
This step is done for character changes from EBCDIC and to get the file transferred to the target HOST where the uploading of data to DB2 will happen.

6. Preparations for Data Upload to DB2
DDLs are generated for DB2 application schema employing the data model by Kumaran's NxTran tool by using the following data type mappings. The generated DDLs are executed against the target DB2 db instance to create the necessary tables and other db structures.

7. Data Upload from Flat File to DB2
To upload data, the DB2 load utility is used to load data from the physical sequential or flat files to the tables created in Step 6. A load utility is capable of efficiently moving a large quantity of data into newly created tables, or into tables that already contain data.

The utility can handle all data types in Appendix A, including large objects (LOB) and user-defined types (UDT). The load utility does not fire triggers and does not perform referentially or table constraints checking, other than validating the uniqueness of the indexes. This provides for exponentially faster load times.
In the mainframe environment, the DB2 load utility loads data from a sequential dataset into one or more tables of a tablespace.

COBOL/CICS to J2EE Application Migration
CICS is a general-purpose data communication system that can support a network of many hundreds of terminals. It may be seen as a specialized operating system whose job is to provide a favorable environment for the execution of online application programs, including an interface to files and databases. CICS provides for the following broad services:

- Telecommunication - The functions required by application programs for communication with remote and local terminals and subsystems
- Multitasking - Control of concurrently running programs servicing many online users
- Data access and transaction control - Facilities for accessing database and files
- Intersystem communication - The ability to communicate with other CICS systems and database systems, both in the same computer and in connected computer systems

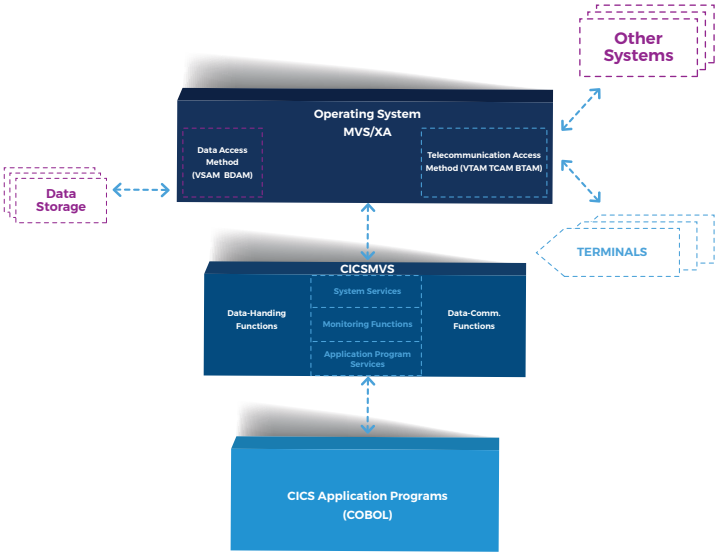
J2EE is a platform that offers a multi-tiered distributed

application model, the ability to reuse components, integrated Extensible Markup Language (XML)-based data interchange, a unified security model, and flexible transaction control. J2EE component-based solutions are not tied to the products and application programming interfaces (API) of any one vendor. Vendors and customers enjoy the freedom to choose the products and components that best meet their business and technological requirements.

CICS/COBOL vs. J2EE Environment:

In CICS, a transaction is a collection of logically related programs in an application. A transaction could be completed through several tasks that can be seen as a single thread of execution. A task can receive data from and send data to the terminal that started it, read and write files, start other tasks, and carry out many other actions. In CICS, a task can contain several Logical Units of Work (LUWs), quite similar to atomic transactions in RDBMS. Thus an application is a collection of programs, which are logically grouped to form several transactions. A task is created when each transaction gets executed. Within each task, we could have multiple LUWs. Tasks are managed by the CICS task control program; the management of multiple tasks is called multitasking.

Like any online applications interacting with users at the terminal, in CICS transactions are conversational. To the user, a series of non-conversational transactions gives the appearance of a single conversational transaction. This means that every time a transaction is run, or logically resumed, a new task is created. CICS calls it pseudo-conversational transactions.



CIC/COBOL Environment

The J2EE platform uses a multi-tiered distributed application model. Application logic is divided into components according to function, and the various application components that make up a J2EE application are installed on different machines depending on the tier in the multi-tiered J2EE environment to which the application component belongs:

- Client-tier components run on the client machine or browsers.
- Web-tier components run on the J2EE server.
- Business-tier components run on the J2EE server.
- Enterprise information system (EIS), (for example, RDBMS)-tier software runs on the EIS server.

J2EE applications are made up of components. A J2EE component is a self-contained functional software unit that is assembled into a J2EE application with its related classes and files. It communicates with other components. The J2EE specification defines the following J2EE components:

- Application clients and applets are components that run on the client.

- Web components such as Java Servlet and JavaServer Pages™ (JSP™) technology components that run on the server enterprise JavaBeans™ (EJB™) components (enterprise beans) are business components that run on the server.

J2EE components are written in Java and are compiled in the same way as any program in the language.

Containers are the interface between a component and the low-level platform-specific functionality that supports the component. Before the web, enterprise bean or application client components can be executed. It must be assembled into a J2EE application and deployed into its container. Here are some of the services provided by the container:

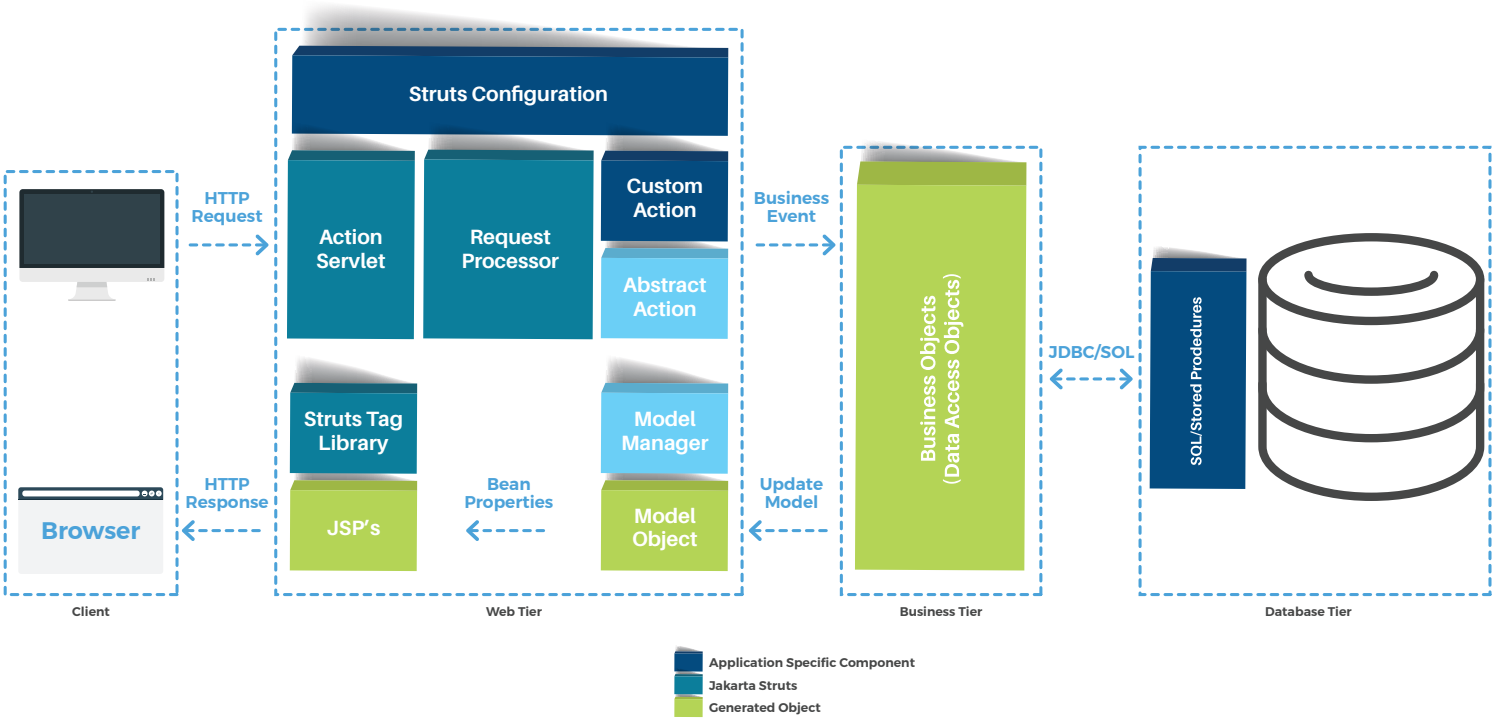
- The J2EE security model lets you configure a web component or enterprise bean so that only authorized users access system resources.
- The J2EE transaction model lets you specify relationships among methods that make up a single transaction so that all methods in one transaction are treated as a single unit.
- JNDI lookup services provide a unified interface to multiple naming and directory services in the

enterprise, so that application components can access naming and directory services.

- The J2EE remote connectivity model manages low-level communications between clients and enterprise beans. After an enterprise bean is created, a client invokes methods on it as if it were in the same virtual machine.

context as J2EE transactions by way of invoking session beans.

In an enterprise bean with container-managed transactions, the EJB container sets the boundaries of the transactions. Typically, the container begins a transaction immediately before an enterprise bean method starts. It commits the transaction just before the method exits. Each method can be associated with a single transaction.



J2EE Environment

Comparing the above two environments, the CICS application can be mapped to one or more logically related J2EE components that are deployed in the J2EE container, which is identified uniquely with Universal Resource Identifiers (URIs). The container handles the client request by locating an instance of the J2EE component, such as Java Servlet, from its servlet pool or creates one if one doesn't exist already. It then assigns a thread from its thread pool to service the client. This is achieved by invoking the instance's service method. The execution of the service method is analogous to a task in the CICS environment. The service method may also support multiple LUWs, which are known in the J2EE

Session Bean

A session bean is a type of enterprise bean that serves as an extension of the client application. It performs tasks on behalf of a client and maintains a state related to that client. This state is called a conversational state because it represents a continuing conversation between the stateful session bean and the client. Methods invoked on a stateful session bean can write and read data to and from this conversational state. This state is shared among all methods in the bean. Stateful session beans tend to be specific to one scenario. They represent logic that might have been captured in the client application of a two-tier system.

Migrating CICS User Interface to JSP in a Struts Framework:

In CICS, screens are defined with Basic Mapping Support (BMS) macros that are a form of assembler language. When the map is defined JCL is used to assemble it into a symbolic map and a physical map. The physical map contains the executable to carry out do these actions:

- Build the screen, with all titles and labels in their proper places with all the attributes for the various fields.
- Merge the data from the program in the proper places on the screen when the screen is sent to the terminal.
- Extract the data for the processing program when the screen is received.

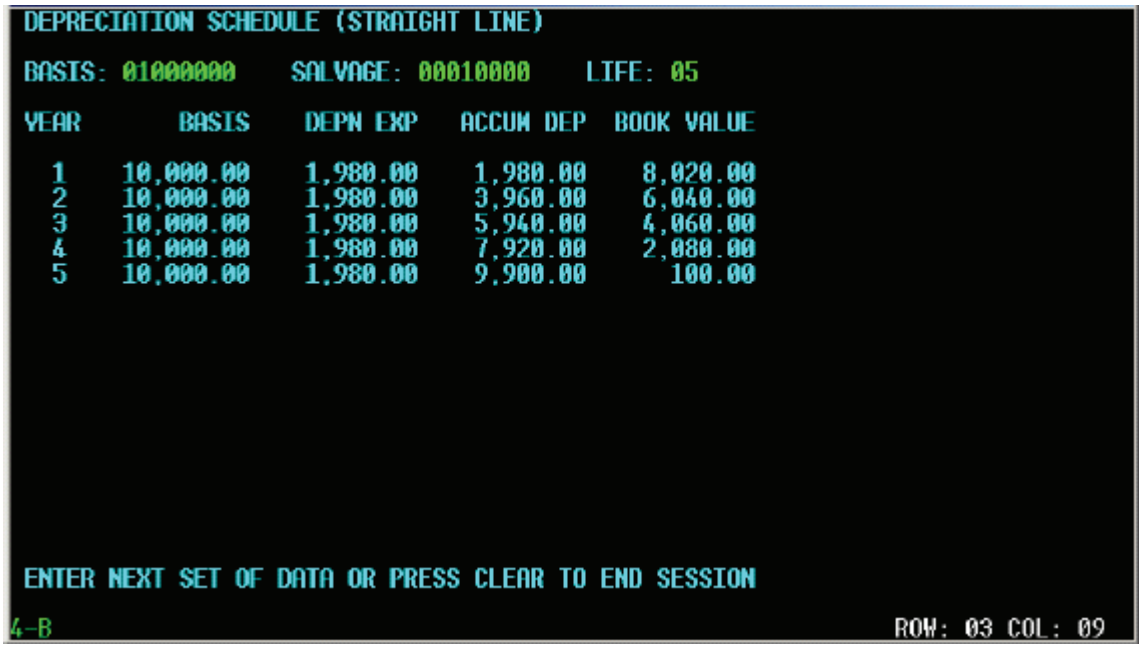
The symbolic map contains a COBOL structure that defines all the fields that are copied into the processing programs during compilation, so that field variables can be manipulated during runtime.

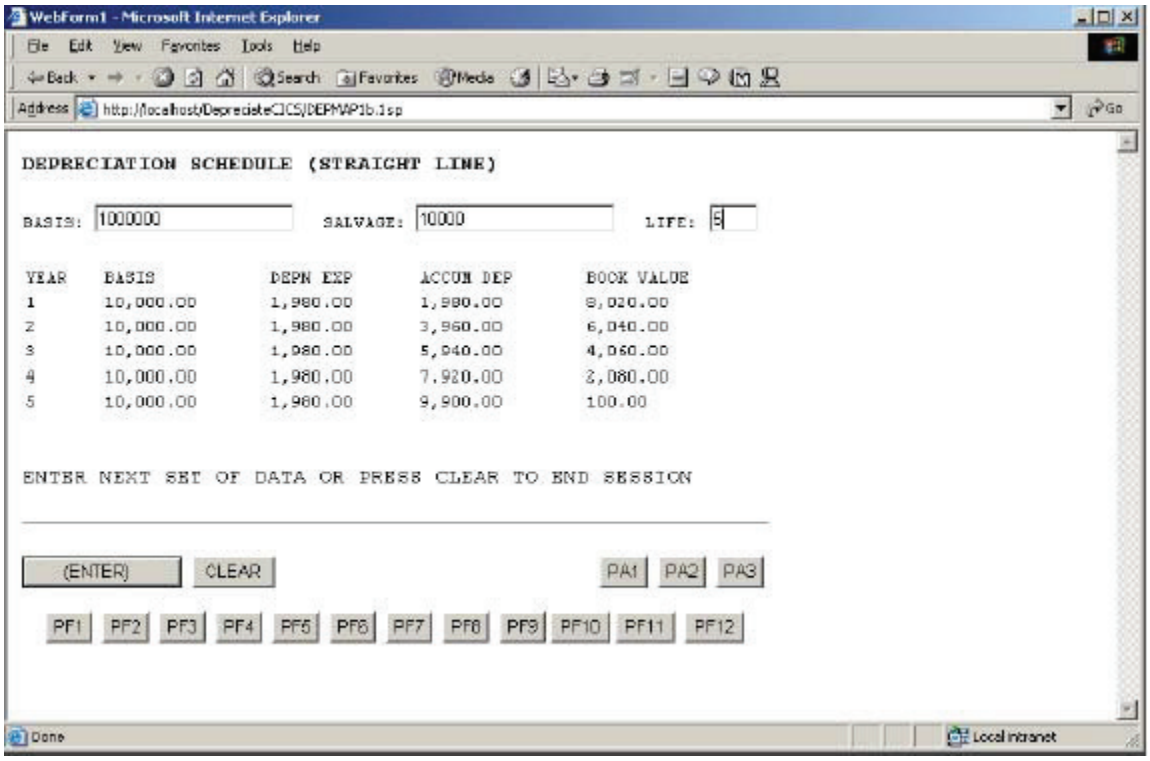
The field definition macros, DFHMDF are used by the Kumaran JSP generator tool to generate the equivalent JSP pages by using Kumaran custom tags and struts tag-libs. The resulting JSP page will generate an HTML page and a Cascading Style Sheet (CSS), which is generated to handle the positioning of HTML form elements similar to the CICS screen layout with these mapping rules:

A DFHMDF Macro is typical as follows:
Fieldname DFHMDF POS=(line, column),
LENGTH=number,
INITIAL='text', OCCURS=number,
ATTRB=(attr1, attr2...)

ATTRB1	HTML
ASKIP – The field cannot be keyed into, because the cursor will skip over it if the user fills the preceding field.	**The field can be mapped as normal texts or <INPUT type=text...> based on the type (Label/Text field) Auto skip can be handled in java script for TEXT type**
PROT – The field cannot be keyed into, but the cursor will not skip over it if the user fills the preceding field.	**PROT can be mapped as a label or Text Field with READONLY option enabled in JSP**
UNPROT – The field can be keyed into.	**HTML Text absolute positioned with <INPUT> tag. Cursor will stop in this field for user input.**
ATTRB2	**HTML**
NORM – Normal Display Intensity	**Normal HTML Text**
BRT – Bright (Highlighted) Intensity	**BOLD HTML Text**
DRK – Dark (not displayed)	**<INPUT type="password" name="..." ...>, only for ATTRB=(DRK, UNPROT)**
ATTRB3	**HTML**
FSET – Turns on the modified data tag. Field is sent from terminal to memory only if the field changes.	**FSET is handled using JavaScript variables.**
FRSET - Field is sent irrespective of changes.	**FRSET is handled using JavaScript variables.**
ATTRB4	**HTML**
NUM – The field can be keyed only numbers, decimal points, and minus signs.	**HTML Text Control with JavaScript to accept numbers only.**

This figure shows the CICS/COBOL User Interface and the migrated user interface using JSP/STRUTS in J2EE:





KUMARAN SYSTEMS

These additional features are incorporated in the converted application at the user interface level.

- Two state fields to Check box item
- Key invocation to Pushbuttons
- Implementations of Toolbars for shortcuts to Program Function keys (PF keys)
- Implementations of Visual Attribute
- User/error messages to Alert
- Multi-state items to List/Radio group items

The CICS field names become struts from bean properties with the accessors get<fieldname> and set<fieldname>. Struts DynaActionForm is used to harvest the form variables from the client request.

A Java bean instance is used to maintain the attribute states of every field in an HTML form. Dynamic changes to the attribute during runtime are set to the bean properties and the bean is used during page generation by the custom tag libraries.

NxTran can read a range of CICS statements in the application program and convert them to equivalent Java code. Similarly, the tool can read BMS macro screen definitions and translate them to JSP and Form/Java Beans.

This is an illustration of how BMS macros with the map set's is converted to its equivalent JSP/Java code:

```
        DFHMSD TYPE=MAP,
        LANG=Cobol,
        MAPATTS=(COLOR,HILIGHT),
        EXTATT=YES,
        MODE=INOUT,
        CTRL=FREEKB,
        STORAGE=AUTO,
        TIOAPFX=YES
        DFHMDI SIZE=(3,80),LINE=1
        DFHMDF POS=(8,20),
                LENGTH=8,
        ATTRB=PROT,
        INITIAL='VALUE B:'
Eno     DFHMDF POS=(8,29),
                LENGTH=3,
                HILIGHT=REVERSE,
                  ATTRB=(UNPROT,NUM)
        DFHMDI SIZE=(3,80),LINE=4
        DFHMSD TYPE=FINAL
        END
```

Equivalent JSP code

```
 <%@ page language="java" %>
<%@ taglib uri="/web-inf/struts-bean.tld" prefix="bean" %>
<%@ taglib uri="/web-inf/struts-html.tld" prefix="html" %>
<html:html>


<body>
        <html:form action="/sample">
        <div name="div1" class="page1">
        <span class="text1">Value B:</span>
        <span class="input1">
        <html:text property="eno" styleClass="inputstyle"
                size=3/>
        </span>
        </div>
        </html:form>
</body>
</html:html>
```

Equivalent Form Bean Class

```
import org.apache.struts.action.Action;
public class sampleForm extends ActionForm{
        private String eno="";
        public String getEno(){
                return eno;
        }
        public void setEno(String eno){
        this.eno=eno;
        }
}
```

Request-Response Processing

These CICS commands are used to send and receive the screen to the terminal:

```
EXEC CICS SEND
        MAP(mapname)
        MAPSET(setname)
        Options...
END-EXEC.

EXEC CICS RECEIVE MAP(mapname) MAPSET(setname)
END-EXEC.
```

The RECEIVE MAP functionality is typically implemented in a client browser as HTTP GET or POST method. The J2EE Container receives the client request along with the form variables and the request itself is handled by the STRUTS framework components like form field harvesting. This is accomplished before passing the bean to the application

code implemented as part of the framework functionality.

Similarly, EXEC SEND functionality is implemented as ACTION FORWARDS to send a selected view/JSP to the client browser with JSP handling the dynamic generation of the HTML form with fields populated with values from the business tier, the form bean, or both.

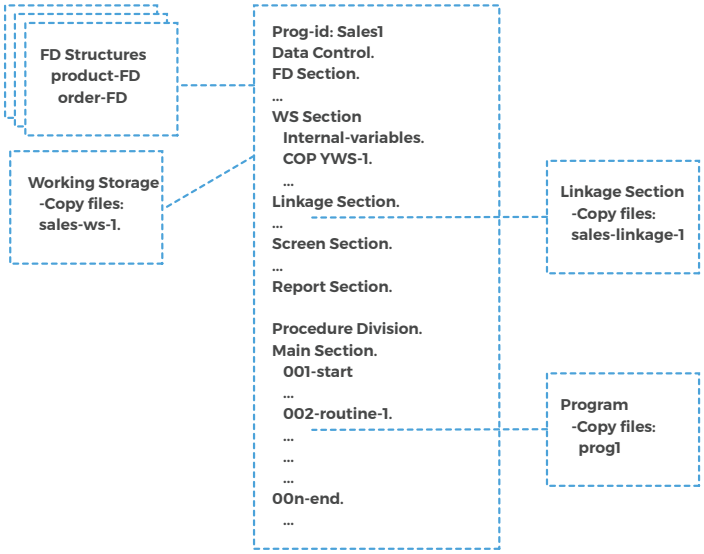
The Kumaran tool automates most of the CICS/COBOL UI (BMS) generation into a STRUTS framework application with JSP as the views remove the code plumbing required to migrate the UI manually.

CICS/COBOL Application Migration To J2EE

A COBOL program typically utilizes the main module and several copy files. The copy files provide means of sharing file descriptions, working storage, or internal variable structures and other definitions. The copy files can also be used in conjunction with the main program module, similar to a library file.

The program module often has references to screens and reports and needs to define the data files that it needs to access.

This diagram illustrates the overall structure of a typical COBOL module:



Decoupling the Presentation layer from the business layer.

In CICS programs there is a clear differentiation of the presentation layer from the business layer. Basic Mapping Support (BMS) makes application programs device-independent. These are the primary functions:

- To remove constant presentation layer information from an application program by placing the default constants in BMS screens
- To provide an access to data fields using symbolic field names

This allows the repositioning of fields in the presentation layer without modifying the application programs or business layer.

In J2EE, STRUTS uses the design pattern Model View Controller (MVC) – Model 2, to separate the view logic from the business logic.

Kumaran Tool migrates all presentation layer logic to JSP Views and the CICS application program logic to Struts Action classes invoking business delegates or service adapters to process the user information that has been requested or submitted in the business tier. Session facades have been implemented by using Session beans.

Conversation State Retention in Client Transactions

In a CICS/COBOL application, whenever a task ends, all working storage variables associated with it get destroyed. However, the DFHCOMMAREA is retained until the end of a transaction.

```
LINKAGE SECTION.
01 DFHCOMMAREA
   02 ...
```

Pseudo-conversational transactions state information is passed from one task to another with the help of the DFHCOMMAREA. In J2EE, the state of the conversation is maintained with the help of session objects. Similar to the DFHCOMMAREA, HttpSession objects live on the J2EE Container; they're just automatically associated with the client by a behind-the-scenes mechanism like cookies or URL-rewriting. These session objects have a built-in data structure that lets you store any number of keys and associated values.

Migrating Data Access Logic

Access to data varies depending on the source of the data. Access to persistent storage, such as a database, varies greatly depending on the type of storage such as relational databases, object-oriented databases, flat files, and vendor implementation. The migrated CICS/COBOL applications use the JDBC API to access data residing in a DB2 RDBMS. The JDBC API enables standard access and manipulation of data in persistent storage, such as a relational database. The JDBC API enables J2EE applications to use SQL statements, which are the standard means for accessing RDBMS tables.

Application logic using JDBC APIs to access the data source, in this case, db2, can potentially create a direct dependency between application code and data access code. Data access logic including the connectivity and data access code within the application logic introduces tight coupling between the application logic and the data source implementation and makes it difficult and tedious to migrate the application from one type of data source to another. When the data source changes, the components need to be changed to handle the new type of data source.

Kumaran's solution is to use a Data Access Object (DAO) to abstract and encapsulate all access to the data source. The DAO manages the connection with the data source to obtain and store data. The DAO implements the access mechanism that is required to work with the data source. The data source could be a persistent store like a DB2 RDBMS, or in this case, an external service like a B2B exchange, a repository like an LDAP database, or a business service accessed via CORBA Internet Inter-ORB Protocol (IIOP) or low-level sockets.

The migrated application logic, which relies on the DA, uses the simpler interface exposed by the DAO for its clients. The DAO completely hides the data source implementation details from its clients. Because the interface exposed by the DAO to clients does not change when the underlying data source implementation changes, this pattern allows the DAO to adapt to different storage schemes without affecting its clients or application logic. Essentially, the DAO acts as an adapter between the application logic and the data source.

NxTran automatically introspects the database and generates the necessary DAOs to access the database tables and transforms the EXEC CICS READ/WRITE/REWRITE/UPDATE statements into application code creating and accessing appropriate DAOs in the migrated application.

Read Statement

```
EXEC CICS READ
        DATASET(<data set name>)
        INTO (data area)
        RLDFLD(data-area)
        LENGTH(data value)
END-EXEC
```

Corresponding Java Code

```
String qry="select cols from <dataset name> where
<colname>=<value>";
PreparedStatement pstmt =
conn.prepareStatement(qry);
ResultSet rs = pstmt.executeQuery();
while(rs.next()){
}
```

Update Statement

```
EXEC CICS READ
        DATASET(<data set name>)
        INTO (data area)
RLDFLD(data-area)
LENGTH(data value)
UPDATE
END-EXEC
EXEC CICS REWRITE
DATASET(<data set name>)
FROM(data area)
LENGTH(data value)
END-EXEC
```

Corresponding Java Code

```
String qry="update <datasetname> set col1=<val1>
,col2=<val2> where <colname>=<value>";
PreparedStatement pstmt =
conn.prepareStatement(qry);
int count= pstmt.executeUpdate();
```

Write Statement

```
EXEC CICS WRITE
        DATASET(<data set name>)
        FROM(data area)
        LENGTH(data value)
RLDFLD(data-area)
END-EXEC
```

Corresponding Java Code

```
String qry="insert into <data set name>(columns)
value(<values>)";
PreparedStatement pstmt =
conn.prepareStatement(qry);
int count= pstmt.executeUpdate();
```

APPENDIX - A

SQL Column Type	COBOL Data Type	SQL Column Type Description
SMALLINT	01 name PIC S9(4) COMP-5.	16-bit signed integer
INTEGER	01 name PIC S9(9) COMP-5.	32-bit signed integer
BIGINT	01 name PIC S9(18) COMP-5.	64-bit signed integer
DECIMAL (p, s)	01 name PIC S9(m)V9(n) COMP-3.	Packed decimal
REAL	01 name USAGE IS COMP-1.	Single-precision floating point
DOUBLE	01 name USAGE IS COMP-2.	Double-precision floating point
CHAR (n)	01 name PIC X(n).	Fixed-length character string
VARCHAR (n)	01 name. 49 length PIC S9(4) COMP-5. 49 name PIC X(n). 1<=n<=32 672	Variable-length character string
LONG VARCHAR	01 name. 49 length PIC S9(4) COMP-5. 49 data PIC X(n). 32 673<=n<=32 700	Long variable-length character string
DATE	01 identifier PIC X(10).	10-byte character string
TIME	01 identifier PIC X(8).	8-byte character string
TIMESTAMP	01 identifier PIC X(26).	26-byte character string
GRAPHIC (n)	01 name PIC G(n) DISPLAY-1.	Fixed-length double-byte character string
VARGRAPHIC (n)	01 name. 49 length PIC S9(4) COMP-5. 49 name PIC G(n) DISPLAY-1. 1<=n<=16 336	Variable length double-byte character string with 2-byte string length indicator
LONG GRAPHIC (n)	01 name. 49 length PIC S9(4) COMP-5. 49 name PIC G(n) DISPLAY-1. 16 337<=n<=16 350	Variable length double-byte character string with 2-byte string length indicator