Netspective Enterprise Frameworks Suite

Building Robust, Cost-Effective J2EE Applications Faster

White Paper

Palmer Business Park
4550 Forbes Blvd, Suite 320
Lanham, MD 20706

(301) 879-3321

http://www.netspective.com
info@netspective.com

Table of Contents

Java 2 Enterprise Edition (J2EE) 1

J2EE Challenges 1
- IDEs and App Servers are not enough 1
- Customer Satisfaction 1
- Business Functionality 1
- Design Issues 1
- Coding Issues 2
- Deployment Issues 2
- Security Issues 2
- Maintenance / Reusability Issues 2
- Maintaining Quality 2
- Process Artifacts / Documentation 2
- Team Productivity 2
- Development Process 2
- Project Management 2

The Solution – Application Frameworks 3
- Types of Frameworks 3

Netspective Enterprise Frameworks Suite (NEFS) 3

NEFS Benefits 3
- Declare More, Code Less 3
- Customization - Themes and Skins 3
- Business Functionality 4
- Flexible, Extensible Design 4
- Eliminate Tedious Java/Struts/JSP Code 4
- Flexible and Customizable Data Objects 4
- Reusable Database Schema 5
- Automatic Client- and Server-side Validation 5
- Rock-Solid Role- and Permission-based Security 5
- Error Handling 5
- Reports 5
- Multiple Runtime Environments 5
- Easier Deployment and Maintenance 5
- Automated Test Harnesses 5
- Automatic Generation of Process Artifacts / Documentation 5
- 3rd Party Frameworks Integration 5
- Team Productivity 5
- Development Process 6
- Project Management 6

NEFS Components 6
- Commons Core Library 6
- Axiom Relational Data Management Service (Data Management Layer) 7
- Sparx Application Platform (Presentation and Application Layers) 7
- Enterprise Console 7

How NEF Libraries Fit In Your Application . 7
- NEF Artifacts 7
- NEF Web Services 8

NEFS Features 9
- General 9
- Architecture 9
- Process and Methodology 9
- Presentation Layer (Sparx) 9
- Data Management Layer (Axiom) 10
- Security and Personalization Layer 10

Alternative Frameworks 11
- Custom Frameworks 11
- Application Server 11
- Embedded Frameworks 11
- Open Source Frameworks 11

NEF vs. Alternative Frameworks 12

Conclusion [TODO] 12

How to get further info on NEFS 12

References 12

Appendix 13
- NEFS Key Concepts 13
  - The NEF Project File 13
  - The XML Data Model (XDM) 13
  - Value Sources 14
  - Commands 14

Many enterprises use Sun Microsystems' Java 2 Platform, Enterprise Edition (J2EE) to enable the software developers to reduce development cost and time-to-market by simplifying problems relating to development, deployment and management of multi-tier mission- and safety-critical enterprise applications. Even though J2EE provides many benefits such as portability, scalability, and reliability, it poses development challenges such as increased complexity and reduced developer productivity (without appropriate tools).

Since most Java developers are not able to handle all the complexities of J2EE, it becomes very difficult for enterprises to develop and deploy robust J2EE applications without the help of *experts,* which are scarce and expensive. J2EE applications require complex and extensive middle layer coding using thousands of available APIs, which increases development time. Given the scarcity of expertise and lack of development time, proper software engineering practices such as prototyping, unit testing, and project documentation are sacrificed.

Application frameworks, with their implementation of design patterns, code libraries, code generators, and document templates present the only solution to these significant problems of complexity and time. Different types of frameworks such as template engines, event models, presentation frameworks, and architectural frameworks have been designed to cater to the varying needs of a development team. Many commercial and open-source implementations of these frameworks are available.

The primary shortcoming of the existing frameworks is that they take care of only a few aspects of the development lifecycle, leaving large portions of development responsibilities on the already burdened shoulders of developers and managers. To take full advantage of the framework ideology, companies are required to work with multiple frameworks. This introduces additional learning, implementation and integration efforts as well as new technology investments, which cancel out many of the benefits provided by these frameworks.

The *Netspective Enterprise Frameworks Suite* (NEFS) is a comprehensive application frameworks suite for building enterprise-caliber applications. It improves application quality and developer productivity as well as reduces time-to-market by eliminating repetitive coding through an extensive library of pre-built and well-tested e-business and database components. This results in increased customer satisfaction through implementation of required functionality in a fraction of the time.

NEFS requires less-qualified engineers to complete a project by using XML to code in a *declarative* style instead of an *imperative* style. Instead of hiring experienced Java architects and experts for a project, managers can instead create high-quality applications using average or inexperienced Java programmers. It also reduces the number of engineers on a project by producing and automatically maintaining process artifacts like unit tests and documentation. Design and prototyping, professional user interfaces, implementation, documentation, and production logs and metrics are just some of the development deliverables and phases that the NEFS helps accelerate and standardize.

Customers such as the Office of Management & Budget (OMB), Northrop Grumman, American Red Cross, DoD, Verizon Wireless, and Xerox have all been able to slash development time and increase quality of their mission- and safety-critical applications using NEFS.

To compete in today's economy, companies must be able to deliver software and services to customers with more value and at a faster pace than ever before. One study of this phenomenon reveals "*a desperate rush-to-market"* and "*a new and unique software market environment"* as its implications [1]. The same study also observes that the development speed, not the cost or quality, drive Internet-speed software development. In a marketplace with shorter release cycles, companies must deploy new features quickly or competitors will get out there earlier and snatch away customers.

These factors create a pressure on the stakeholders and the development teams. Working in such an environment changes how software development is organized. Companies are now adopting software development methodologies such as Rational Unified Processes (RUP) and eXtreme Programming (XP) to cope up with challenges of faster delivery in an increasingly demanding market.

Java 2 Enterprise Edition (J2EE)

In addition to adopting new development methodologies, many companies are also making use of development technologies such as Java 2 Platform, Enterprise Edition (J2EE) that support faster development of scalable and portable multi-tier applications for enterprises. J2EE is targeted to simplify enterprise applications by basing them on standardized, modular components, by providing a complete set of services to those components, and by handling many details of application behavior automatically [3].

J2EE Challenges

Even though J2EE along with programming paradigms such as eXtreme Programming (XP) and Rational Unified Process (RUP) significantly reduces the development time and cost, there are many problems associated with its use. Each phase of application development cycle brings its own challenges.

IDEs and App Servers are not enough

Most of the challenges that are outlined below are not solvable by simply buying an integrated development environment (IDE) or application server. Proper software engineering, code reusability, code frameworks, and processes meet development challenges. IDEs and App Servers are just not enough to do the job.

Customer Satisfaction

The competitive market pressure results in rapidly changing requirements. When requirements are fuzzy and fast changing, intimate access to customers slashes time delays and ensures that high-priority features are correctly identified. This, however, poses the following challenges:

- Less time available for creating application mockups and prototypes
- Need to create full-fledged navigation, page flow, and forms mockup while working with the customers who are embedded into the development team
- Quickly creating impressive and creative web designs and GUI layouts.
- Need to incorporate customer ideas and requirements into apps "in a matter of hours"
- How to customize UIs and themes on role- and permission-basis

Business Functionality

Many existing applications face the following problems associated with business functionality:

- Missing business functionality in the final app because of time spent on infrastructure coding
- Implementing repetitive business functionality "from scratch"

Design Issues

Several design-related issues face architects when developing J2EE applications:

- Designing applications with solid object-oriented foundations that are implementation-technology (.NET, J2EE) independent
- How best to program to interfaces instead of classes
- How to create stable app design using standard design patterns
- How to avoid rigid, monolithic and brittle design
- Determining which J2EE services to use
- Taking decision on the appropriate use of complex J2EE components such as JNDI, JMS, JDBC, JNI, servlets and JSPs.
- Separating server-side/transport data formats from the GUI
- Providing sound mechanisms for event management, application flows, and widget control
- Dividing application's visual entities (panels, components, etc.) into panes with specific, consistent responsibilities and functions

- Separating GUI layout from GUI feature code (validation, events), application flow, navigation, server interaction, and business logic
- Avoiding tight-coupling and duplication of business logic
- Separating presentation components from service components
- How to avoid inconsistent data services across apps
- Defining and orchestrating the interaction between the presentation and service components
- Designing presentation layer to provide strong, orderly way to implement app logic and flow
- How to provide Wireless and PDA support
- How to write code that is easy to test
- Defining the proper mix between checked and unchecked exceptions
- How to choose between Servlets, JSP, standard tags, custom tags, beans (Java for JSP, EJB), etc. for the UI layer

Coding Issues

- How to master thousands of J2EE APIs
- How to reduce the time spent on writing repetitive DAOs, VOs and wrapper objects for persistence layer
- A different set of coding skills are needed for writing the EJBs, Servlets, and JSPs
- How to implement consistent data validation and error handling

Deployment Issues

- How to define and manage server configuration parameters and deployment descriptors
- Properly specifying JAR/WAR files to contain right data on EJB classes, bean type, etc.
- How to manage platform-specific data and files
- How to define and manage app configurations
- How to provide environments for development, debugging and production phases

Security Issues

- How to integrate security, privacy, and encryption on page, field, and business rule levels
- How to provide role-based security
- How to define and manage hierarchical permissions and roles management

Maintenance / Reusability Issues

Maximization of reusable components results in fast-paced development. This reuse is needed at business logic, interfaces, and backend infrastructure levels. The issues to be considered are:

- How to reduce the amount of code to be written for an app by repeated use of successful coding experience
- How to avoid additional coding efforts inherent in traditional reuse of components
- How to increase code reusability by removing problems such as SQL embedded inside Java code that are common in existing application infrastructures
- How to implement dependency inversion principles to ensure easy maintenance

Maintaining Quality

- How to increase customer satisfaction by maintaining app quality
- How to build test harnesses faster

Process Artifacts / Documentation

Part of every J2EE application development process is spent in technical writing. So even if your code is completed on time, this task takes up a lot of the total project resources. The associated challenges are:

- How to create software process and app implementation artifacts quickly
- How to keep the artifacts accurate and up-to-date
- How to find skilled technical writers

Team Productivity

- How to increase productivity of junior programmers
- How can the junior programmers use existing skill set to start building J2EE apps
- How to reduce the number of expert engineers needed for a project
- How to use the available expert resources effectively
- How to incorporate code collaboration to improve team productivity

Development Process

Programming paradigms like XP and RUP have exploded into the developer community due to the high demand for a more controlled engineering process. Although the benefits for such paradigms are tremendous, there are increased risks of project failures, the causes for which being:

- Higher learning curve
- Development time spent on creation of software process artifacts

Project Management

- How to obtain and analyze metrics (function points, SLOC, etc.) to monitor app performance

- How to reduce development costs by reducing the number of expert developers, web designers, and technical writers
- How to monitor team productivity

The Solution – Application Frameworks

"A framework is a partially complete software system that is intended to be instantiated. It defines the architecture for a family of systems and provides the basic building blocks to create them. It also defines the places where adaptations for specific functionality should be made." (Buschmann 1996)

Table: Anatomy of a Framework

FEATURE	DESCRIPTION
Knowledge base	Product reviews, configuration information, lessons learned
Document Templates	Requirements, Use Cases, Design Specs, Test Plans, etc.
Design Patterns / Standards	Coding and naming standards, proven design strategies, etc.
Code Libraries	Base classes, format utilities, tag libraries, etc.
Code Templates and Generators	Automated generation of starting code, based on templates

To be useful, a framework should be reusable, extensible and well documented. It should contain functionality useful across a wide variety of application domains, and it should be easy to learn.

Types of Frameworks

Different types of frameworks have been designed with corresponding open source and commercial applications. The following table lists some of the important ones:

FRAMEWORK TYPE	EXAMPLES
Template Engines	FreeMarker, Velocity, WebMacro
MVC 2 based Presentation Frameworks	Struts, Maverick, Webwork
MVC2 based Architectural Frameworks	Expresso, Tapestry, Turbine
Vertical Frameworks	Oracle Business Framework

Netspective Enterprise Frameworks Suite (NEFS)

The Netspective Enterprise Frameworks Suite (NEFS) is a standards-compliant frameworks suite which helps build secure data-driven web applications by providing out-of-the-box general-purpose e-business functionality and making it accessible to junior programmers through XML. Design and prototyping, implementation, unit tests, implementation documentation, and production logs and metrics are just some of the development deliverables and phases that the NEF helps accelerate and standardize. Unlike existing frameworks, NEFS takes care of *all* phases of a project's lifecycle.

NEFS Benefits

Declare More, Code Less

Through its ideology of "declare more, code less", NEFS leverages XML to significantly reduce the amount of code, increases re-use, maintains consistency across multiple projects, and improves code quality.

The input fields, page navigation, page flow and validation criteria are all declared in simple XML files. This results in quick creation of mockup screens and prototypes. It also reduces complexity and modification time associated with incorporating customer feedback thus resulting in increased customer satisfaction.

The declarative programming style allows analysts to create prototypes that can later be completed by programmers.

Complete application navigation and page flow can be created declaratively *within hours*. These navigations and page flows are easily customizable based on user roles and access rights.

Customization - Themes and Skins

NEFS separates form/report presentation from form/report design and logic by automatically creating all HTML and DHTML in user-defined *skin* objects. It provides several built-in skins to be used to build professional and creative web design and GUI layout for your applications.

You can define your own skin objects and use them to customize your application's look-and-feel quickly and without the help of expert designers.

Data-driven applications involve handling a lot of data which usually equates to a lot of user screens. Once your custom skin is developed, Sparx generates the app look-and-feel and provides rich functionality reducing the total development time.

The skins infrastructure also allows identical business logic to be used across different user interfaces for a variety of browsers and platforms like hand-helds.

These themes and skins can also be customized for various user access levels and roles using the appropriate XML tags/attributes. This significantly reduces the time and effort required to incorporate customized look-and-feel for individual users.

Business Functionality

NEFS provides a high-level breakdown of your system (Sparx, Axiom, Commons) and defines a low-level breakdown of types and how these types communicate to solve your business problems (dialogs, fields, pages, menus, queries, etc.). This alleviates the burden of having to define the plumbing for your system and also allows you to focus primarily on our business logic.

By making efficient use of reusable framework components as well as custom-built components, NEFS supports use of common business functionality across applications.

It supports writing all your business logic using EJBs, rules processing systems, or plain Java objects and connecting them to NEF components like dialogs, fields, and schemas easily.

It allows re-use of common application functionality and business values across modules and projects through *commands* and *value sources*.

Flexible, Extensible Design

NEFS is built on object-oriented foundations and can be used effectively with any implementation technology.

It takes benefits from implementing best practices as well as design and implementation patterns such as MVC, Command, and Lazy Init. It also complies with standards such as Java/J2EE, Servlets/JSP, XML, JDBC, EJB, HTML, and CSS.

It uses automatically-generated web services that change as the rest of your application changes. As you add features, tables, forms, and functionality the web services remain in sync.

It keeps the GUI separate from the server-side data formats. The data access objects are automatically generated by the framework.

It favors no specific templating system but comes bundled with support for both JSP and FreeMarker.

It does not favor Servlets over JSPs or JSPs over Servlets and can work in one, the other, or both environments simultaneously with no loss of functionality in either environment.

It allows the web services to *automatically* become applications and applications to automatically become services with very little work on the part of analysts or programmers. For example, every Sparx form/dialog automatically provides the capability for becoming a web service. Additionally, any table or SQL query defined using Sparx automatically has the capability to run in both "application" and "service" modes.

Eliminate Tedious Java/Struts/JSP Code

NEFS allows you to declare your user interface forms, fields, validation rules, conditional logic, and security policies using simple XML tags and get fully functional theme- and skin-based HTML output automatically. All the Java code you write is related to application functionality, not user interface (HTML) generation or database access.

It lets you declare your database structures and relationships using simple XML tags and get fully functional SQL DDL, SQL DML, HTML documentation and XML import/export capabilities *automatically*.

It allows you to declare your database SQL and parameters using simple XML tags and get fully functional reports and synchronized and validated input capabilities automatically.

It provides a series of scripts that, among other functions, allow you to compile your application, generate code, and upgrade NEF libraries and components.

Flexible and Customizable Data Objects

NEF provides a flexible approach to accessing persistent data and managing your user sessions through object-relational map, static and dynamic queries, and stored procedures.

It allows decoupling of user interface from the data model.

It also provides the hooks to plug into your persistent data and manage it in your domain tier.

Through NEF, defining and registering your own special types is very easy. Once registered with the framework, the custom types can be used anywhere.

Reusable Database Schema

Through re-use of Schemas across applications and different database vendors, NEF enables database programmers to spend time on essential tables and schema elements significant to a specific application instead of rewriting common schema elements for each application.

Automatically generated database documentation is kept up-to-date.

Since the DDL can be generated using the built-in script, experienced DBAs are no longer required to create consistent, high-quality SQL DDL during the design and construction phases.

Automatic Client- and Server-side Validation

Using NEF's client- and server-side Javascript validations, features like required fields and conditionally displayed fields become trivial to implement and test.

Programmers can also insert (extend) or replace JavaScript feature with their own custom functionality.

Rock-Solid Role- and Permission-based Security

You can develop secure applications using NEF's built-in support for user authorization rules and declarative control of access to application components based on authorization rules, such as roles and permissions, without any programmer intervention.

Error Handling

NEF error handling provides a way to handle errors propagating from the database as well as from your own code. Using this approach, you can create your error handling pages to display custom messages to the users.

Reports

You can declaratively define presentations for your queries, without the need to write your own markup, and save a lot of development time.

Multiple Runtime Environments

You can use the *development*, *testing*, and *production* environments to make appropriate decisions about data sources and other environment-specific settings. For example, throwing exceptions in a development environment but e-mailing errors in testing or production.

Easier Deployment and Maintenance

NEFS manages the objects' life-cycles saving you from the difficulties of providing efficient object reuse.

It automates the app builds using the integrated Ant build engine and supplied build files.

It allows you to combine multiple "sub-applications" into a main application context -- share common themes, skins, navigation, security, and sessions across multiple applications.

Maintenance headaches are reduced through the use of automatic deployment descriptors, packaging, logging, performance statistics, and application metrics.

Automated Test Harnesses

NEFS simplifies your testing process by automatically generating test objects for all the forms/dialogs as well as SQL queries.

Automatic Generation of Process Artifacts / Documentation

NEFS generates for you and automatically maintains XP/Agile, RUP, CMMI, and Waterfall process methodology artifacts like application metrics, unit tests, functional specs, and traceability documentation that take weeks of back and forth development time otherwise.

All the artifacts are automatically updated as the application components are changed. This ensures the availability of *accurate* and *up-to-date* documentation and process artifacts without requiring help from expert technical writers.

3rd Party Frameworks Integration

NEFS provides the ability to integrate 3rd party frameworks such as Struts and OpenSymphony and leverage existing investments.

Team Productivity

NEF takes an experienced J2EE programmer and improves their productivity and, in many cases,

eliminates the need for specialized resources for architecture and design, prototyping, documentation, logging, deployment and maintenance.

NEF improves developer productivity and reduces time-to-market by eliminating repetitive coding through an extensive library of pre-built and well-tested e-business functionality and database components.

It allows the junior programmers to use simple XML tags and build complex J2EE applications without any extra training.

It works with any IDE, editor, or studio apps thus leveraging the existing skill set of the programmer.

It uses open source tools such as Ant, JUnit, HttpUnit, Log4J, Jakarta Commons, which are already familiar to most of the Java programmers.

Through its light-weight browser based Enterprise Console, NEFS allows the teams working in the same room or spread over different geographical areas to collaborate with their customers, management and other team members.

The XML based project files are easily version controlled using any of the available version control tools such as CVS.

Development Process

NEFS delivers benefits to your programming process. The declarative approach in conjunction with code generation provides for rapid development, clean separation of the things that change from the things that stay the same, and very quick turnaround time when you need to modify/fix your system.

The pre-packaged architecture, functionality, code, documentation, and guidelines significantly enhance your ability to improve the resource utilization of your team.

It also allows you to use expensive architects and senior developers for true architectural and business analysis tasks and quality improvement.

NEF delivers powerful prototyping features which allow users to see what they're getting before coding begins.

It helps you keep software soft. That is, UI, database and other components are kept in resource files instead of code.

Since there's less code to deal with, handling changing requirements is relatively easy.

NEF also reduces the learning curve for your team by enabling the use of already familiar tools and techniques.

Project Management

NEF requires less-qualified engineers to complete a project by using XML to code in a declarative style instead of a programmatic style.

It reduces the number of engineers on a project by producing and automatically maintaining process artifacts. This reduces the overall project cost.

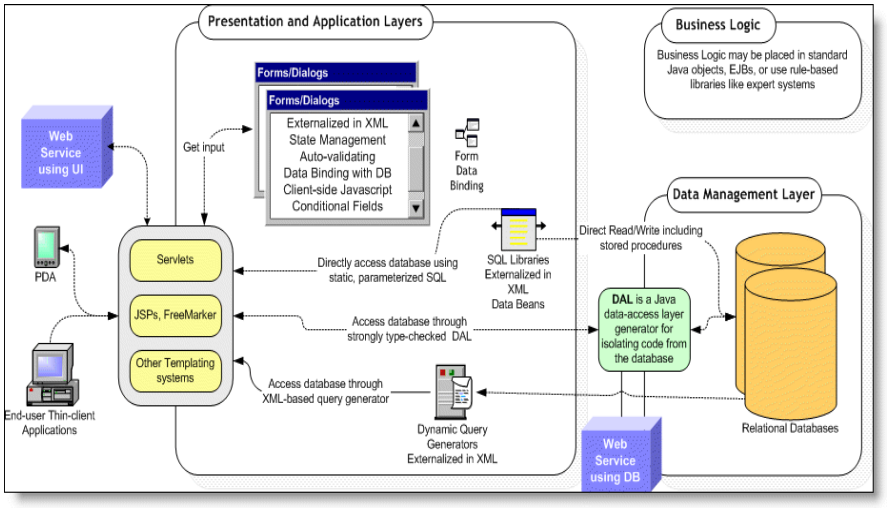
It provides metrics for project comparison, project complexity, feature and function points, and source lines of code.

Managers can easily monitor team progress by using the NEF Enterprise Console.

NEFS Components

The NEF suite comprises multiple libraries and frameworks, all of which are designed to work together.

Figure 1: NEF Components



Commons Core Library

The *Commons* library contains general purpose classes for access control, configuration management, value sources, data validation, and XML parsing. The

Commons library may be used in both web-based and non-web-based applications.

Axiom Relational Data Management Service (Data Management Layer)

The *Axiom* service provides static query, dynamic query (query definitions), and schema management functionality. Axiom is a sophisticated relational database management framework that provides the capability to completely manage schemas, SQL DDL, SQL DML, XML import/export, and vendor-independent persistence. The Axiom framework depends upon the Commons library and may be used in both web-based and non-web-based applications.

Sparx Application Platform (Presentation and Application Layers)

The *Sparx* presentation framework helps manage navigation, dialogs (forms), reports, and other complex user interface functionality. Sparx helps create sophisticated yet easy-to-use user interfaces on all common browsers including Microsoft Internet Explorer, Mozilla, and Opera. The Sparx framework is built using Commons and Axiom and is designed exclusively for creating web-based thin-client applications.

Enterprise Console

The NEF and all application components built with it can be viewed with the *Netspective Enterprise Console* servlet. Each application has a private instance of the Console. It serves as a central repository for the project containing all the implementation and design documentation. It provides facilities for collaborative unit/integrated testing of presentation and data management layer. It also supports central management of connection pools to identify open and leaked connections. In addition to that, console can be used for monitoring project metrics as well as database and security policies.

The Enterprise Console is an optional component for every application built with NEF and it is turned on by default. The programmers may decide to turn it off completely for their applications or secure it differently.

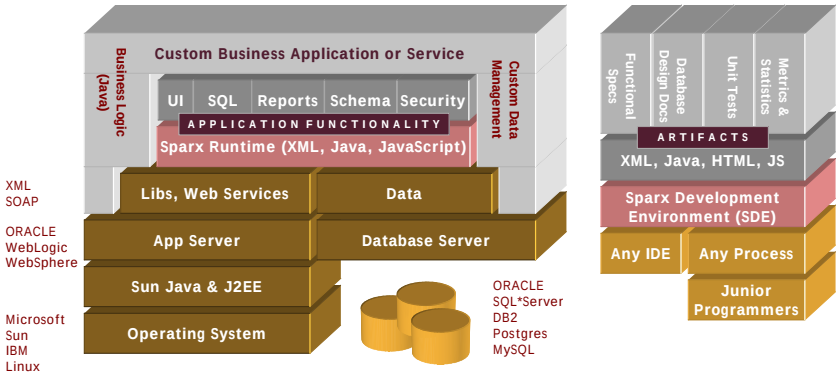
The console is an excellent example of one of NEF's features called *multiple sub-applications*. It means that a primary application can host and co-exist with other shared sub-applications that may be related. The sub-application scheme uses the standard servlet context mapping scheme of the URI paths used by servlet containers. Each sub-application is merely another servlet and can share (if it needs to) the main-application's themes, skins, and descriptors or may use its own.

How NEF Libraries Fit In Your Application

NEF components are pure Java libraries that reside *inside your application*, unlike other similar frameworks which are *containers for your application*. The main difference is interoperability of frameworks. Netspective Frameworks are designed as enterprise frameworks that can stand-alone or enhance other COTS or in-house frameworks.

NEF consists of three JAR files containing the Commons, Axiom, and Sparx binaries and a set of HTML and XML resources like XSLT style sheets, icons and an extensive JavaScript library. This simple structure affords developers a great deal of flexibility in how they want to use any of the Netspective Frameworks. Therefore, they do not have to redesign or recode their applications as they would to comply with the limitations of a framework container. Instead, they can start out by using a few Sparx features here and there and adopting more of the Sparx ease and speed of development as the need arises. And, because the frameworks all work like normal Java libraries, version control, debugging, and code management is not affected.

Figure 2: NEF Position in J2EE Stack

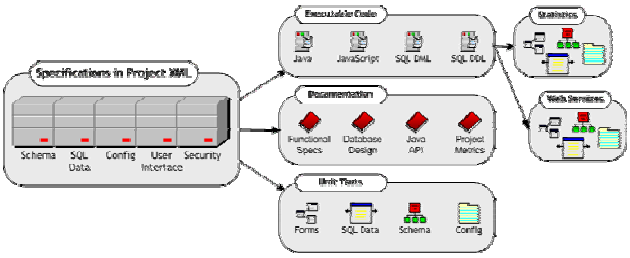


NEF Artifacts

Using simple XML declarations for schema, SQL, UI, etc., NEF produces a variety of objects and artifacts. Many of the components that NEF produces are not generated but instantiated while

others are both generated first then instantiated. The `WEB-INF/classes/auto` directory contains all classes that are generated. So, if a class is found in directory other than `WEB-INF/classes/auto` (the auto-generation directory) then it's a custom or user class that was created or prepared outside of NEF. NEF loads most components using XML which is translated at the startup of the application into objects that are cached, shared, and executed. Unlike most code generators, which create code that must be separately compiled, many of the NEF components are automatically executed in memory with no compile/link/debug cycle.

Figure 3: NEF Artifacts



NEF Web Services

The general topic of web services refers to the ability of applications and systems to speak to each other over Internet protocols.

The "normal" case of web applications has a customer accessing a catalog site and making a purchase over a secure website. This interaction is quite common but sometimes its preferable to have a computer system automatically place an order with other computer systems. For example, suppliers could provide web services to large corporations so that corporations could automatically, without human intervention, place orders to the supplier when their inventory runs low. NEF supports both web applications (where a human being is interacting with an application or computer system) and web services (where a service is being created for use by other computers).

NEFS Features

The NEF has hundreds of features that support the development of sophisticated applications with exceptional ease.

General

NEF General Features	
Declarative web application development	OS and App-server independence
Works with any IDE, editor, studio apps	Supports total lifecycle of development
Junior-level programmer competency Required	Uses normal Java inheritance and delegation for extensiblity
Works but does not require EJB	Native J2EE API always available
Improves debugging by using the integrated Jakarta Commons Logging engine.	Automate your builds using the integrated Ant build engine and supplied build files.
Encapsulate app functionality into commands and increase reuse across projects.	Encapsulate business values logic into objects that be reused across modules and projects.
Works well with other frameworks like Struts	Capability of migrating to .NET
Collaborate with other developers across the room or across the world using only a browser.	Ability to version-control components and apps

Architecture

NEF Architectural Features	
Executable specifications - rules declared in XML automatically produce app logic	Plug-in architecture to replace any framework classes with custom classes
All object construction of UI, DB, Security performed in external XML files	Fully integrated security, UI, persistence, and business logic frameworks
Automatic code generation	Business rules and values repository reusable across applications
Support for web services	Logging and Performance Stats
Large library of J2EE design/architecture patterns	Large library of application integration and functionality patterns
Centralized configuration parameter through XML-based properties files with support for variables	Allows production, testing, development modes and allows apps with conditional logic capability

Process and Methodology

NEF Process and Methodology Features	
Web-based administration console allows visualization and management of all framework objects	Automatic unit test harnesses of major application components
Basic end-user integration testing with no programmer intervention	Unit Testing of Custom Components (using Junit)
Automatic generation of functional specifications and implementation documents	Automatic generation of function points, SLOC, and other application metrics

Presentation Layer (Sparx)

NEF Presentation Layer Features	
Control complex application behavior with integrated security and navigation personalized to individual users or roles.	Develop sophisticated multi-module single-sign-on apps using one Servlet context.

Leverage built-in report writers and XSLT transformers or connect to your favorite reporting engine.	Create auto-validated forms with dozens of built-in field types that easily handle and recover from all user errors.
Use your favorite template engine for content - JSP, FreeMarker, Velocity, Tea, etc.	Develop apps that have a consistent UI that can switch looks and feel without code changes.
100% declarative navigation, workflow, forms and controls with no JSP or custom tag needed	Automatically run SQL to bind data in forms/fields/pages
Includes professionally designed navigation, forms, reports and styles which reduce the need for graphics artists	Automatic declarative server- and client-side validation, controls, keypress filters and data formatting
Complete HTML generation of forms (without JSP or templates required)	Wireless, PDA, Browser Support

Data Management Layer (Axiom)

NEF Data Management Layer Features	
Create vendor-independent relational tables (in XML and Java-based meta data) and generate SQL DDL, DAOs, and XML import / exporters.	Define SQL in XML and automatically generate Lightweight Data Access Objects (DAOs) with bind variables and statement pooling.
Connect to multiple data sources within a single page	Connection Pooling (both built-in and access to JNDI)
Manage pooled connections centrally to visualize open and leaked connections.	Reusable library of static and dynamic SQL queries kept in XML files
Define SQL rules in XML and let your users enter search criteria that auto-generates SQL with bind variables and joins.	Isolate database-specific functionality such as auto-inc into delegated objects.
Automatic declarative inserts/updates/deletes of data	Automatic creation of forms to add/edit/delete data from tables
Automatic SQL performance statistics tracked for all static and dynamic SQL	Automatic generation of HTML Design Documentation for Database Schema

Security and Personalization Layer

NEF Security Layer Features	
Harden your applications with auto-managed users, roles, and permissions.	Declarative permission and role assignment to forms, fields, tables, etc.
Automatic support for multiple attempts lockout	Auto-generated login form
Multiple Access Control Lists for cross-app, sub-site, etc.	Tracking of all user logins, logout, audit trail
Ability to integrate with other commercial or custom single-sign-on solutions	Authentication and authorization may be extended using JAAS/custom security classes

Alternative Frameworks

There are several alternative frameworks available but they offer only *partial* solutions to the problems at hand.

Custom Frameworks

Due to the lack of robust commercial offerings, today most enterprises choose to build their own frameworks and platforms. These in-house and home-grown solutions tend to have a great deal of momentum behind them because they may be favored internally by the experienced and senior engineers that construct them. However, home-grown solutions exhibit the following shortcomings:

- The custom frameworks often have limited features and are usually poorly documented.
- They are notoriously difficult to extend because they are usually designed for special-purpose jobs.
- They are usually expensive to create and maintain and require engineering attention which is taken away from business requirements. Programmers often end up spending time on infrastructure issues instead of issues that directly affect end-users.

With its unique architecture, NEF can supplement in-house frameworks or replace them completely. It is able to work alongside home-grown solutions and is extensible enough to allow enterprises to make a slow switch over to NEF.

Even if all of your engineers are very experienced and your architecture is very sound, part of every J2EE application development process is spent in prototyping, technical writing, testing, and maintenance. Even if the code is completed on time, these other tasks take up to another 30 to 40% of the total project resources. NEF can take an experienced J2EE programmer and improve their productivity and in many cases eliminate the need for specialized resources for architecture and design, prototyping, documentation, logging, deployment, and maintenance.

Application Server

Most J2EE application servers contain a great deal of common infrastructure code that helps *contain*, *manage*, *deploy*, and *execute* applications. They also increase scalability, reliability, and availability of applications by providing features like clustering and failover. What they lack, because it's not part of their product requirements, is the ability to help build the applications they contain. Just as an operating system is a container for executing applications written in any language but does not assist in the development of the applications, a J2EE application server is a container for executing Java and J2EE applications but does not assist in the development of the Java/J2EE applications. Sparx, as a J2EE framework, helps create the applications that you can then deploy onto any J2EE container or application server.

Embedded Frameworks

Many commercial embedded frameworks are provided by application server vendors like WebLogic or WebSphere. The main issues associated with these frameworks are:

- They are usually vendor-specific and tie you to a particular application server, operating system, or database server
- The frameworks are not designed for complete coverage of sophisticated applications and engineers end up having to do a great deal of infrastructure work on their own.
- Most embedded application server frameworks are very light on functionality.

NEF, on the otherhand, can supplement or replace your application server vendor's frameworks with a good mix of easy-to-use components for faster development and the sophistication that senior architects need to create robust applications. Also, all NEF-based applications are application-server, database-server, and operating-system independent.

Open Source Frameworks

Freely-available open source frameworks are quite good for certain types of applications. There are a number of good tools in this category like Struts, Enhydra, Expresso, and others; however, they all share the following issues:

- They have limited features or functionality
- Most of the open source tools cater to very talented Java engineers since they require a serious understanding of low-level computer science concepts, J2EE APIs, and Java programming.
- Most open source tools are written by experienced programmers for experienced programmers and as such depend on mutual understanding instead of heavy documentation.
- While they save money in the beginning (since they are free) they end up costing much more because they often require more experienced engineers, usually require a higher number of engineers, and don't have the quality management tools built-in.

NEF can complement or replace most frameworks like Struts or Expresso.

NEF vs. Alternative Frameworks

There are two primary competitive advantages that Netspective's frameworks have over other frameworks.

Simplification	NEF focuses on pure reduction of time and effort for the engineer and the use of the technology by less-experienced staff members. Also, the expert programmers enjoy working on end-user and application domain tasks instead of grunt coding in the infrastructure.
Holistic approach to engineering	NEF concentrates not just on the coding aspects but the engineering aspects such as testing (quality), metrics (measurement), monitoring performance, and documentation.

Conclusion [TODO]

How to get further info on NEFS

Although a great deal of information has been presented in this document, it's only a small portion of the information available online and in other documents. The documents and evaluation applications available at http://www.netspective.com will show you:

- How to evaluate NEFS online and on your own system.
- How to create a sample application.
- How to use the Sparx build scripts and NEFS Enterprise Console to help develop your application.

You may also contact our support team at support@netspective.com or call us at +1 (301) 879-3321

References

1. R. Baskerville R. Ramesh, et al, "Is Internet-Speed Software Development Different?," *IEEE Software*, Nov./Dec. 2003, pp. 70–77.

2. J. Bosch, P. Molin, et al, "Object-Oriented Frameworks - Problems & Experiences," Object-Oriented Application Frameworks, eds. M. Fayad, D. Schmidt, R. Johnson, John Wiley, 1999.

3. Java 2 Platform, Enterprise Edition (J2EE) Overview http://java.sun.com/j2ee/overview.html

Appendix

NEFS Key Concepts

The NEF Project File

All of the NEF (Sparx, Axiom, and Commons) components are declared in an *input source* file known as the *Project File*. The Project File may be either a single file with all components or may be broken up into multiple files and pulled into the main Project File by using `<xdm:include>` tags.

Example: Basic NEF Project File

```
<?xml version="1.0"?>
<project
xmlns:xdm="http://www.netspective.org/Framework/
Commons/XMLDataModel">
   <xdm:include
resource="com/netspective/commons/conf/commons.
xml"/>
   <xdm:include
resource="com/netspective/axiom/conf/axiom.xml"/>
   <xdm:include
resource="com/netspective/sparx/conf/sparx.xml"/>
   <xdm:include
resource="com/netspective/sparx/conf/console.xml"/>

   <!-- Your application tags go here. -->

   <xdm:include file="your/own/file.xml"/>

   <!-- Your other application tags go here. -->
</project>
```

The *com.netspective.sparx.Project* class

All of the tags that are processed in the NEF Project file are managed by Java classes. The class that manages the XML root <project> tag is called com.netspective.sparx.ProjectComponent. The ProjectComponent class owns an instance of the com.netspective.sparx.Project class. The Project class then has ownership of all of the tags contained within the <project> tag.

The XML Data Model (XDM)

The eXtensible Markup Language (XML) plays an important role in NEF's ease of use, extensibility, and code generation. All dialogs, fields, validation rules, some conditional processing, all SQL statements, dynamic queries, configuration files, database schemas, and many other resources are stored in XML files that are re-usable across applications. Although XML is the preferred method for creating resource files, almost anything that can be specified in XML can also be specified using the NEF Java APIs. NEF uses the JAXP and SAX standards for parsing and processing XML files.

XML is not used as yet another imperative programming or expression language like Java, C/C++, or Pascal. Instead, it is simply used to declare classes, rules, specifications, and other application requirements that are automatically parsed, read, cached, and executed by one or more NEF components. The dynamic aspects of NEF applications comes from Java through the use of `Value Source` and `Command` interfaces, not a new programming language.

Overview

XML Data Model (XDM) is designed to help Java programmers construct and configure Java objects using XML files without worrying about parsing and error checking. The `NEF project` file uses XDM to declare all project components. The primary objective is to declare your object's data and structure in XML and have XDM automatically construct the objects and assign the data provided in an XML file with almost no performance penalties. XDM does not introduce a new language nor does it replace Java classes, methods, or business logic. The main use for XDM is to allow you to use resource files (XML) to declare the hierarchy and initial data model (state) of your objects. The reason there is almost no performance penalty is that the XDM simply reads the XML and invokes the exact same constructors and methods in your classes exactly as if you had done so by manually writing Java code (including inheritance and composition).

XDM Features

XDM automatically translates XML tags and attributes to your object hierarchy and data model with no manual mapping or configuration.

XDM uses simple JavaBeans naming standards and the Java Reflection API to instantiate and initialize the objects declared in the XML resource files.

XML elements (tags) are instantiated into runtime objects exactly as if you called the object constructor in Java code.

XML attributes invoke Java class mutator (setter) methods exactly as if you called the set method in Java code.

XML element text (PCDATA) may be automatically appended using configurable method calls.

XML child elements that have element content can be configured to create child objects or call mutator (setter) methods. This allows you to use the element or attribute style of XML resource files.

Automatic type conversion validates and translates text from XML attributes into the proper object's mutator (setter) method parameter types: boolean, int, float, String, etc. Automatic error checking will ensure that when your class setter method parameter type changes, the XDM will automatically validate the new parameter type.

The translation process is configurable so that you can create element or attribute aliases (so that you can create XML tag and attribute names that may be mapped to different method names in Java than what the JavaBean standards would imply).

XDM templates provide object composition and inheritance behavior exactly as Java code.

XDM include files provide the capability to break out large object construction tasks into multiple resource files.

XDM transformers provide the capability to filter XML files through XSLT or other transformations before object construction takes place.

Using the factory design pattern, the XML traversal is separated from the object instantiation and method invocation.

Custom handlers allow portions of XML to be handled manually.

XDM Performance

With NEF, all XML data is cached and only read when it changes. Basically, all NEF XML files are read using lazy-read approach; meaning, they are read only when needed and even then only once. So, the majority of all NEF XML performance impacts (if any) occur at the startup of a server-based application. Once the application starts all data is cached and shared across users and XML-related performance issues are eliminated. Also, NEF relies on the SAX programming interface for XML parsing and processing thus requiring far less memory and performance overhead than frameworks that rely on DOM interface.

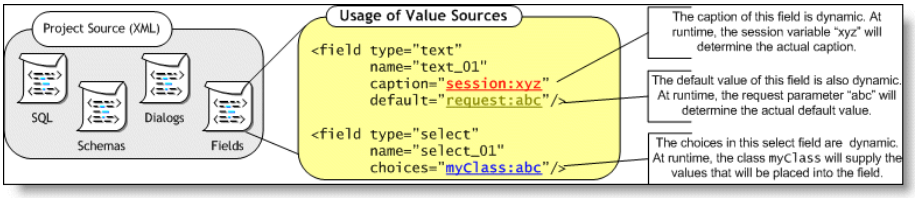
XDM Tags versus JSP Tags

JSP tag libraries are usually used for emitting HTML. They are commonly used in the view component of the MVC paradigm. NEF XML tags are used to describe and declare all aspects of your application (UI, database, security, etc). NEF can use existing JSP tag libraries but they are seldom needed because NEF's Sparx framework handles most tasks within its own, much simpler, and much more powerful XML tags.

Value Sources

Value sources allow dynamic data to be included in XML without creating a programming language inside XML. There are many locations in the Project where value sources are used: configuration variables, forms, SQL statements and SQL bind parameters. Value sources allow common business logic and *business values* to be stored in shareable instance and then used either in XML or Java files where necessary. Value sources can be either single or multiple (list context) and are used anywhere dynamic data is required.

Figure 2.0: Value Sources Overview



Commands

Commands are Java classes that execute arbitrary tasks defined either by you or the framework. They are used to encapsulate common logic and reuse that logic across pages and dialogs/forms.