Using XML and *Executable Specifications* in an Enterprise Application Framework to Generate Java, Increase Programmer Productivity, and Decrease Development Costs

---

An introduction to the Netspective Sparx Application Platform

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

(301) 879-3321

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

Copyright

Disclaimer and limitation of liability

Netspective and its suppliers assume no responsibility for any damage or loss resulting from the use of this developer's guide. Netspective and its suppliers assume no responsibility for any loss or claims by third parties that may arise through the use of this software or documentation.

Customer Support

Customer support is available through e-mail via support@netspective.com.

Contents

An Introduction to Sparx

Netspective's patent-pending Sparx enterprise application framework allows companies to build and deploy more thin-client, browser-based, data-driven, dynamic Java applications using fewer programmers, in less time, with higher-quality, and better documentation than conventional application servers and Servlet engines alone. Using Sparx, you can stop building applications from scratch each time and leverage your existing talent and components. As a pure Java library, Sparx can be integrated into existing systems at any development stage. Sparx will work with any Java2 JDK and Servlet engine and can work equally as well as a JSP library (with advanced, pre-defined custom tags) or a Servlet library (providing MVC-based page control). Some of our customers have been able to cut their development budgets in half (in some cases by up to 75%) by reducing the number or qualifications of their programmers. This is because Sparx allows junior and mid-level Java developers to be as or more effective than senior and more experienced developers that do not use Sparx.

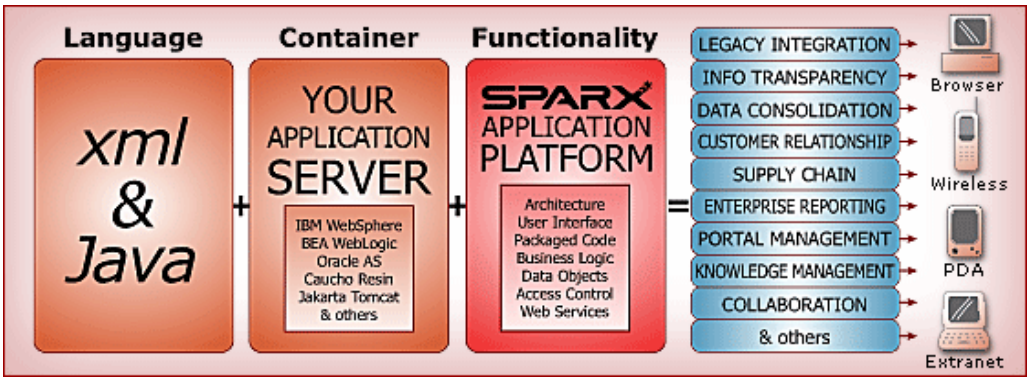


Figure 1: Overview of Sparx Functionality

Why Sparx?

Sparx is a complete framework which helps with the entire software development lifecycle. Design and prototyping, implementation, automatically generating unit tests, automatically creating implementation documentation, and providing production logs and metrics are just some of the development deliverables and phases that Sparx helps accelerate and standardize.

Sparx Benefits

- Application developers spend time on real features significant to end-users instead of infrastructure issues.
- Technical managers can better manage their application development projects by utilizing the built-in project management, application documentation, unit-testing, and artifact-generation tools.

- Most of the user interface and database logic is coded in a declarative style using XML instead of a programmatic style using Java. This significantly reduces the amount of code (as much as 50-75% of code can be eliminated), increases re-use, maintains consistency across multiple projects, and improves code quality.
- Analysts can use the declarative user interface features to create prototypes that can later be completed by programmers (no more throw-away prototypes).
- Applications are built by assembling declared UI (forms/dialogs) and database (SQL) components combined with application-specific business logic using single or multiple distributed application tiers.
- Sparx is not a templating system that simply generates HTML but a feature-rich framework that significantly reduces the time to produce high-quality data-intensive thin-client applications.
- Sparx 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.
- Implementation can be done using XML, Java, or both.
- Implements common design patterns like MVC and factories. Skins infrastructure allow identical business logic to be used across different user interfaces for a variety of browsers and platforms like handhelds.
- Sparx enhances and works equally well with all project development methodologies including waterfall, RAD, OOAD, and the agility of methodologies like eXtreme Programming (XP).

Sparx and its Relationship to other tools

Sparx is designed to run on all platforms and enhances the industry-leading application servers, tools, and database servers by increasing programmer productivity and reducing development costs. Sparx supplies almost 80% of all *application functionality* and *artifacts* leaving only *custom data management* and *business logic* to be implemented by junior or senior programmers. Based on its design goals, Sparx does not compete with many tools. Instead, it ties together most of the tools that an average programmer would use for a typical e-business project. Table 2 below shows a list of the various development tool categories and how Sparx fits with those tools.

The "Sparx Fit" column indicates how Sparx works within a particular category. Each category can have one of the following options:

SPARX FIT	DESCRIPTION
Can Replace	Means that Sparx can optionally replace the tools in the category. For example, if you're using a tool in the category you can either continue to use it with Sparx or replace it completely with Sparx.
Enhances	Depending upon the number of stars in the "scope" column, means that Sparx enhances the tool in the category. None of the tools with this fit are competitors but using Sparx may obviate the need for the tools in certain cases. For example, Sparx can *almost* replace ERD tools depending upon your application's needs. Additionally, depending upon your requirements Sparx can *almost* replace specialized portals and content management systems (since you can easily create custom versions of those products in Sparx).
Utilizes	Means that Sparx does not have features of tools that are in this category but does work well with them.

SPARX FIT	DESCRIPTION
Includes	Means that Sparx requires one of the tools in this category and the underlined tool is included in the Sparx kit.

Table 1: Sparx Fit Column Legend

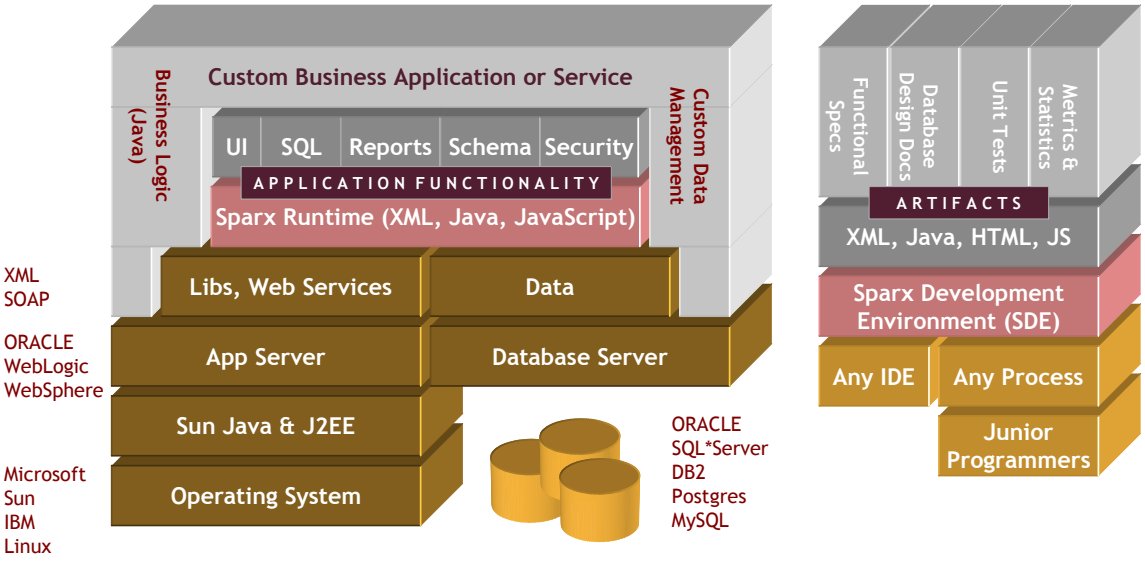
TOOL CATEGORY	REPRESENTATIVE TOOL VENDORS	SPARX FIT	SPARX SCOPE
Application Framework	Avalon, Struts, Turbine, Barracuda, Expresso	Can Replace	✪✪✪✪✪✪✪✪✪✪
O-R Mapping Framework	CocoBase	Can Replace	✪✪✪✪✪✪✪✪✪✪
Security Framework	Netegrity	Can Replace	✪✪✪✪✪✪✪✪✪✪
Database/ERD Design Tools	Embarcadero, ORACLE, Rational, TogetherSoft	Enhances	✪✪✪✪✪✪✪
Portal and CMS Servers	JetSpeed, Brock, IBM, Microsoft SharePoint, BroadVision	Enhances	✪✪✪✪✪✪✪
Analytics and Reporting Tools	Informatica, Crystal Reports, Actuate	Enhances	✪✪✪✪✪✪
Unit Test Tools	JUnit, HttpUnit	Enhances	✪✪✪✪✪✪
Template Language	Tag libraries, Cocoon, PHP, Velocity, ASP	Enhances	✪✪✪✪✪
CASE Tools	Rational Rose, UML, TogetherJ, Elixir	Enhances	✪✪✪✪
Functional and Load Test Tools	Mercury, Parasoft, Microsoft, Rational	Utilizes	
IDE	JBuilder, NetBeans, Eclipse, Visual Studio, Visual Café	Utilizes	
Transaction Server	MTS, DAO, TXSeries, Tuxedo	Utilizes	
Debuggers, Profilers	Microsoft, IBM, Sun, Sitraka JProbe	Utilizes	
Version-control Tools	ClearCase, CVS, Perforce	Utilizes	
Database Server	IBM, ORACLE, Microsoft, mySQL, Hypersonic SQL	Includes	
Application Server	Weblogic, Tomcat (Free), Websphere, Resin, JRun	Includes	
XML, XSLT Tools	Xalan and Xerces	Includes	
Build and Logging Tools	Ant and Log4J	Includes	
Web Server	Apache, IIS, or App Server	Includes	

Table 2: Sparx relationships to other tools

Sparx and its Relationship to your Application

Sparx is a pure Java library that resides *inside your application*, unlike other similar frameworks which are *containers for your application*. The main difference is interoperability of frameworks. It is designed as an enterprise framework that can stand-alone or enhance other COTS or in-house frameworks.

Sparx consists of a single JAR file containing the Sparx binary 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 Sparx. Therefore, you do not have to redesign or recode your application as you would to comply with the limitations of a framework container. Instead, you 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.



The Roles of XML and Web Services

XML

The eXtensible Markup Language (XML) plays a huge role in Sparx's ease of use, extensibility, and code generation. Sparx declarations are performed using XML -- 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 Sparx Java APIs. If you are not familiar with XML, please visit http://www.xml.com/ for some training materials. Sparx uses the JAXP, W3C Document Object Model (DOM), and SAX standards for parsing and processing XML files.

Sparx utilizes XML in a *declarative*, not *algorithmic* capacity. What this means is that XML is **not** used to define yet another programming or expression language. Instead, it is used to declare classes, rules, specifications, and other application requirements that are automatically parsed, read, understood, and executed by Sparx. The dynamic aspects of Sparx applications comes from Java through the use of a simple Value Source interface (an implementation of the Value design pattern), not a new programming language.

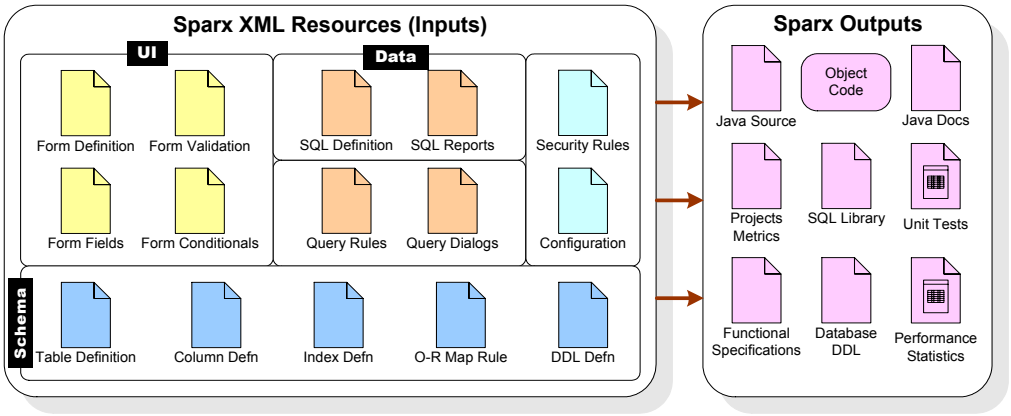


Figure 2: Sparx XML Inputs and Corresponding Outputs

XML Performance

When engineers first learn about XML and the amount of flexibility it affords in both application and data management, they jump at the chance to include XML features within their projects. However, soon they learn that dealing with XML sometimes means sacrificing performance. With Sparx, all XML data is cached and only read when it changes. Basically, all Sparx 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 Sparx XML performance impacts (if any) occur at the startup of a server-based application. Once the application starts all data is cached and XML-related performance issues are eliminated.

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 it's 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.

Sparx 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*). Sparx allows the web services to automatically become

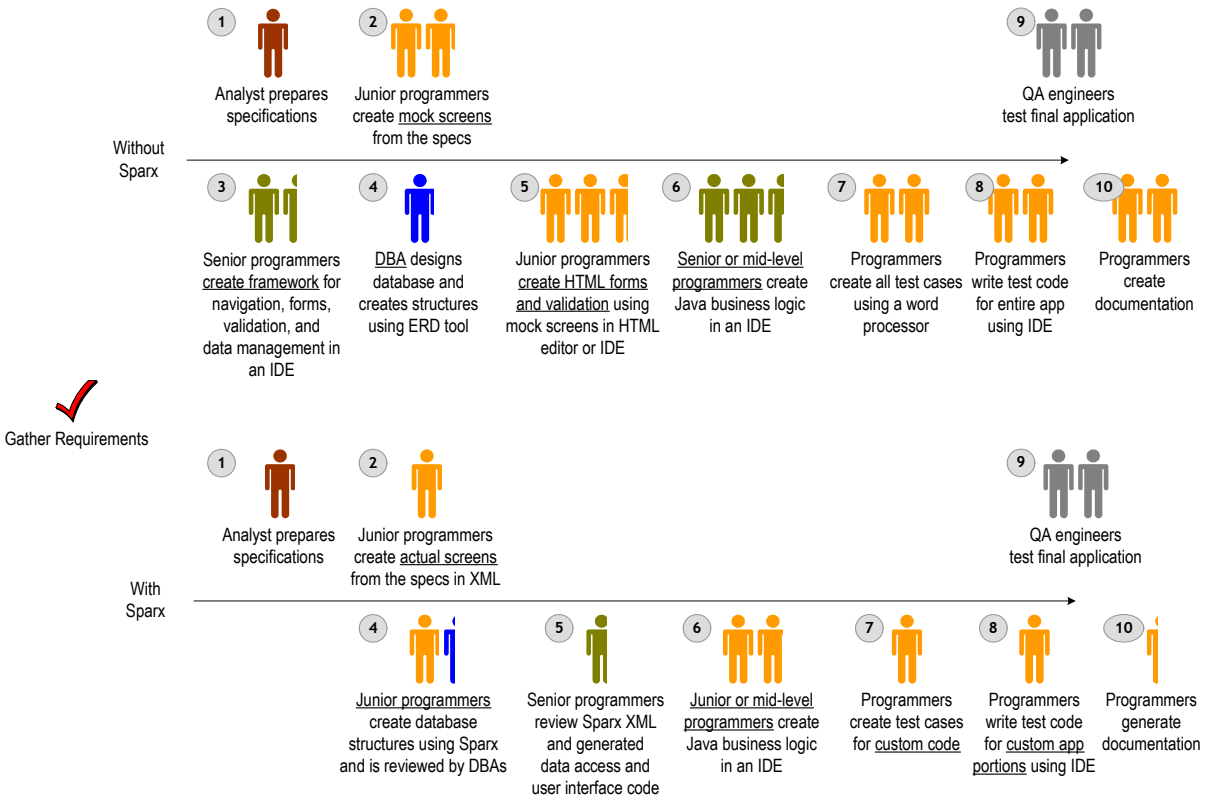
applications and applications to automatically become services with very little work on the part of analysts or programmers. For example, every Sparx form or dialog automatically provides the capability for becoming a web service. Additionally, any table or SQL query defined using Sparx automatically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

How Sparx Improves the Development Process

Based on the small number of inputs and the large number of outputs Sparx produces, it is applicable in a number of areas. The specific ways that Sparx improves the development process is shown in the following table and diagram.

WITHOUT SPARX	WITH SPARX
More code required	Less code required through auto generation
More staff required, in general	Fewer staff members required, in general
More senior staff required	Fewer senior and experienced staff required
Longer time to complete projects	Shorter time to complete projects
More tools to learn	Fewer tools to learn
Complex code that is usually not reusable	Simpler code that is reusable across projects
Documentation has to be written manually	Documentation is auto generated
Metrics and statistics need to be gathered manually	Metrics and statistics are generated automatically

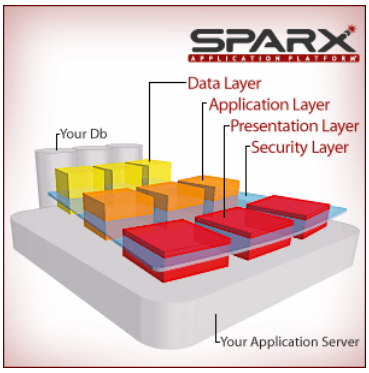
Table 3: How Sparx improves the development process



Key Sparx Concepts

Clean Separation of Presentation, Business, and Data Layers

Sparx encourages and can optionally enforce a very clean (almost pristine) separation of the three common layers existing in every useful application. The components of Sparx focus on specific layers, providing all or most of the functionality required for that layer and maintaining hooks and connections to the appropriate up-stream and down-stream layers. In Sparx it is both possible and suggested to never mix HTML, Java, JavaScript, and SQL in the various layers.



Executable Specifications

The majority of the Sparx features including dialogs (UI), SQL Statements, dynamic query rules and schema definitions, are implemented using what are called *Executable Specifications*. These executable specifications mean that most of the applications' resources double as both specifications (which can be extracted and automatically documented) and executable code. The exact same XML resources *simultaneously* serve as the declaration, functional specification, executable code, generated Java source code, and testable application functionality.

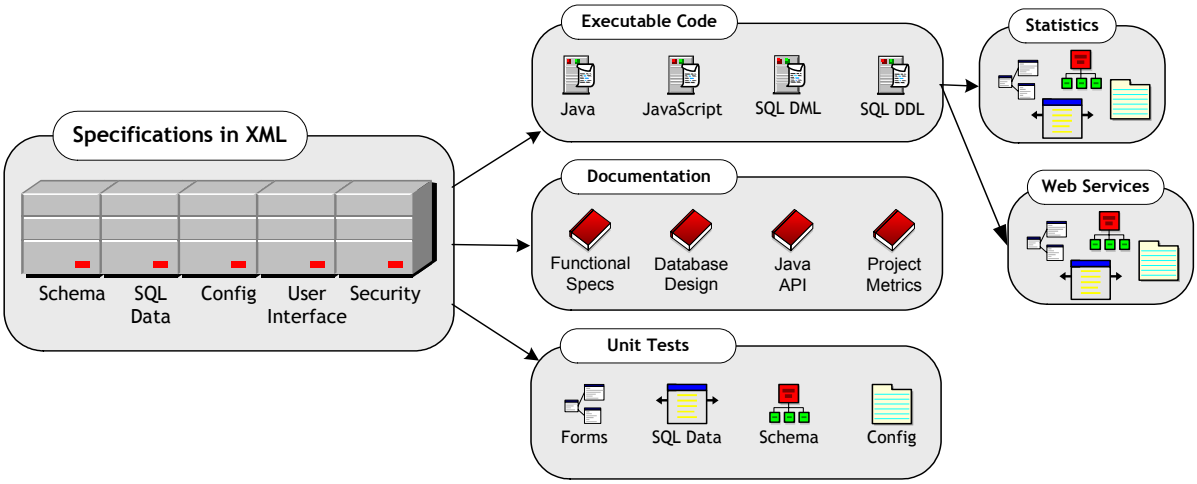


Figure 3: Diagrams describing how Executable Specifications Function

Java and XML Bindings

By design, much of Sparx is declared in executable specifications residing inside XML files; however, it's important to realize that not everything *must* be specified in XML and that the programmer has full control of Sparx resources within Java. To facilitate this control within Java, Sparx helps *bind* the XML to Java by generating facades, helper classes, and subclasses that allow programmers to use Java APIs for managing the XML.

SPARX BINDING	PURPOSE
Dialog Context Bean (DCB) or "Form Bean"	A Java class that represents the fields of a form so that programmers see fields as methods and dialogs/forms as classes without worrying about HTML or XML.
Identifiers classes	Simple classes that generate constants for XML nodes. Configuration items, SQL statement names, form names, and security roles and permissions are accessible through Java constants instead of basic strings.
Data Access Layer (DAL)	Java classes for every table, column and data-type defined in a schema.

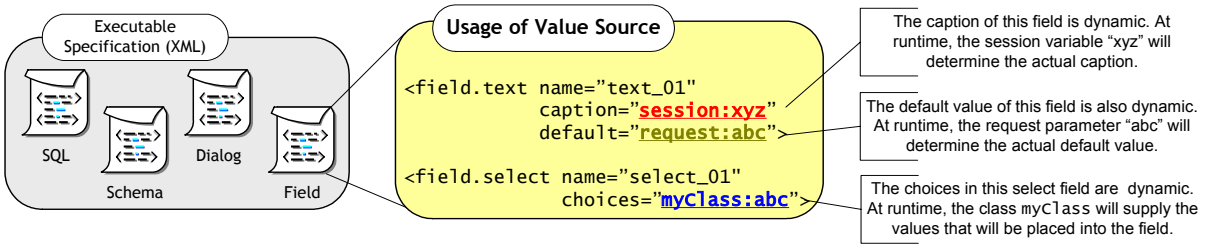
Table 4: Examples of Sparx Java Bindings to XML Resources

Value Sources

Value sources provide dynamic access to common business logic and may be considered a business rules library. Many of the classes in Sparx use value sources, which are simply Java classes that follow a specific interface, to provide values for captions, defaults, comparisons, conditionals, and many other types of variables. Value sources allow common business logic to be stored in reusable classes and then used either in XML or Java files where necessary.

As stated earlier, Sparx strives to keep XML files declarative in nature (not programmatic). As such, constructs like "for-loops" and "if-then" decisions are left to general-purpose languages like Java and JavaScript. Value sources allow XML files to declare usage of dynamic variables without creating yet another expression language that the programmer would need to learn.

Value sources can provide either single or multiple (list context) values and are used in dialogs, fields, SQL statements, and many other places where dynamic data is required. The format of a value source is similar to a URL (`name:params`).



TYPE	DESCRIPTION
Single Value Sources (SVSs)	A SingleValueSource (SVS) is an object that returns a single value from a particular source like a request parameter, text field, or session attribute. The concept is that a single instance with a particular URL-style parameter string will be provided and then whenever the value is needed, a `ValueContext` will be provided to allow either static content or dynamic content to be served. Many Single Value Sources can double as List Value Sources (depending upon the context).
List Value Source (LVSs)	A ListValueSource (LVS) is an object that returns a list of values from a particular source (like a select field or SQL query). The idea is that a single instance with a particular URL-style parameter string will be provided and then whenever the value is needed, a `ValueContext` will be provided to allow either static content or dynamic content to be served. Many List Value Sources can double as Single Value Sources (depending upon the context).
Custom Value Sources	The Value Sources infrastructure was designed for extensibility. There are already numerous sources defined by Sparx, but you are encouraged to create your own value sources that might wrap a SOAP call, an EJB call, calls to your own custom Java classes, and many other similar functions. By wrapping your own business functionality in a thin value source, all of your existing functionality and logic could be made available to Sparx.

Table 5 : Types of Value Sources

For a complete list of value sources available in Sparx, please visit http://developer.netspective.com/xaf/value-sources.html.

Centralized XML-based Configuration Files

Sparx favors XML storage of properties instead of using Java properties files. The `ConfigurationManager` class allows multiple properties to be defined in a single XML file, complete with variable replacements and the ability to create single-property or multiple property (list) items. Optionally, any property name could refer to value sources as part of the definition of a property so that the value of a property can become dynamic and be computed each time the property is used (in case the value of the property is based on a Servlet request or session variable or some other application-defined business rule).

XSLT

The W3C's eXtensible Stylesheet Language (XSLT) is used internally by most Sparx modules for generating code, functional specifications, metrics, and many other artifacts. Although it's not crucial for you to know XSLT to *use* Sparx, it is important to know XSLT if you'd like to *extend* Sparx to generate your own custom code or documentation. Please visit http://www.xslt.com if you are not already familiar with XSLT.

Sparx Modules and Components

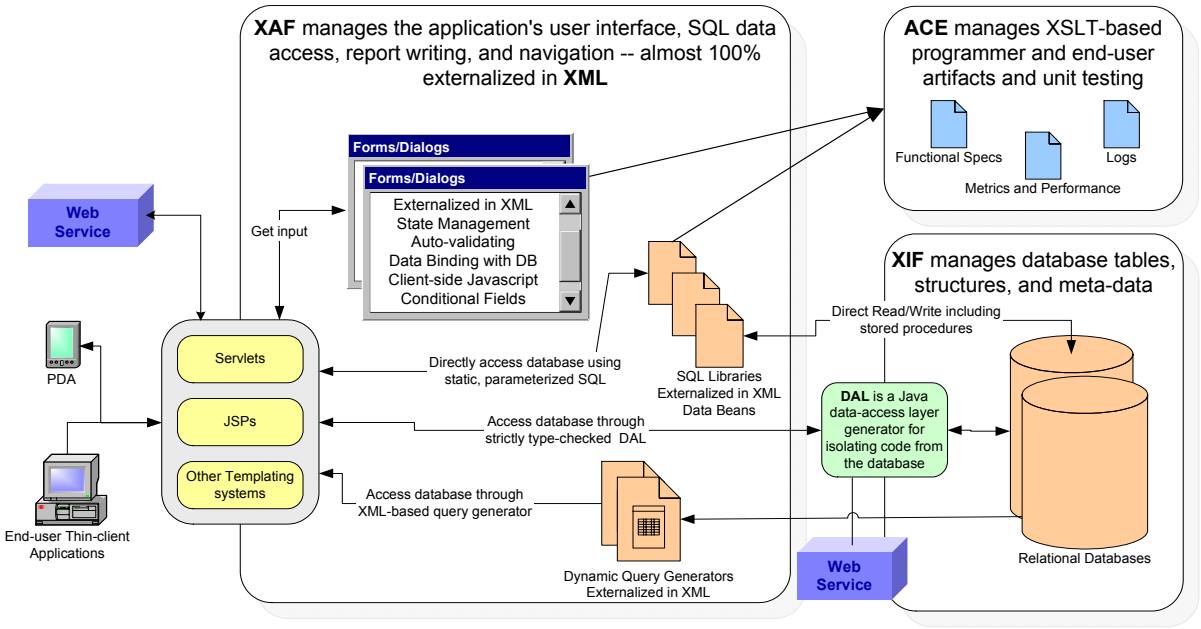


Figure 4: Sparx Modules Overview

MODULE		PURPOSE
ACE	Application Components Explorer	A browser-based administration console (written using Sparx) that provides a documentation generator, unit test generator, and code generators.
XAF	eXtensible Application Framework	Sparx framework responsible for application user interface, form beans, data beans, SQL data access, report writing, and navigation.
XIF	eXtensible Information Framework	Sparx framework responsible for application business rules and data management, database connectivity rules, and schema management.
DAL	Data Access Layer	Sparx framework responsible for creating tightly-bound Java classes to application schema (fully automated object-relational or O-R mapping).

Table 6 : Sparx Modules Overview

eXtensible Application Framework (XAF)

The eXtensible Application Framework (XAF) is Sparx's Java library consisting of over 250 re-usable classes that greatly simplify the development and deployment of small-, medium-, and large-scale thin client, browser-based, data-driven, dynamic e-business applications. Based on Enterprise Java standards like J2EE, XML, Servlets,

JSPs, JDBC, and JNDI, XAF uses proven re-use development methodologies that can be applied to produce robust Java server-side applications.

XAF Features

FEATURE	DESCRIPTION
Advanced Forms/Dialogs	The XAF refers to HTML forms as "Dialogs" because it handles the two-way interaction between browsers and users completely; this includes data persistence, data validation, a sophisticated client-side JavaScript library and user interface skins. Dialogs can be defined completely in XML, completely in Java, or a combination of the two. Even in XML, the entire Dialog including labels, captions, validation logic, conditional displays, and other advanced UI features can be easily defined. By keeping the entire definition in XML, non-programmers or junior engineers can create forms and more experienced developers can attach business logic as appropriate.
Data sources and Database Connectivity	The XAF provides powerful database connection and aggregation services. Starting with a simple interface to one or more database connection and pooling engines and including such features as dynamic data source definitions and selection, the database connectivity support sets the stage for both static and dynamic SQL libraries and pooled/cached result sets.
SQL DML Generation	JSP custom tags and java classes are provided to automatically create SQL insert/update/remove DML (Data Manipulation Language) commands. By providing simple name/value pairs of data, XAF can automatically generate complex DML statements.
SQL Statement Libraries	To encourage reusability and encapsulation and reduce the amount of time spent creating "data beans", XAF allows all SQL statements and dynamic parameters used in a project to be specified in one or more SQL files. Once defined, a single or multiple SQL statements may be used in reports, dialogs (forms), Servlets, or JSP-pages. In many cases, SQL statement pooling completely replaces simple data-serving beans since data objects are automatically created for all SQL statements. Data can be easily aggregated from multiple data sources because each SQL statement in the statement pool can be specified (either in XML or JSP) to come from a variety of pre-defined or dynamic data sources.
Dynamic Queries	A powerful XML-based tool called Query Definitions allows developers to define tables, columns, joins, sort orders, and other important data through the use of Meta Information about data relationships. Once a developer creates a query definition, XAF allows end-users to use simple HTML-based forms to automatically generate accurate SQL and performance-tuned statements to create paged reports or export data to external sources.
Reports	XAF reports are defined completely in XML. This includes headings, banners, column types, calculations, grouping, sort order, etc. By keeping the entire definition in XML, non-programmers or junior engineers can create report definitions and more experienced developers can attach business logic.
Skins and Device Independence	XAF separates form/report presentation from form/report design and logic by automatically creating all HTML and DHTML in user-defined "skin" objects. The skins perform all drawing operations while the report/form objects manage all of the fields and validation. One immediate benefit of skins is the ability to design and describe a dialog once and execute it on mobile, small form- factors (handhelds), notebooks, and desktops or to different formats like PDF, comma-separated, and tab-delimited files.

FEATURE	DESCRIPTION
Security and Personalization	XML-based centralized Access Control Lists (ACL's) allow developers to restrict access to forms, reports, pages, and other resources based on user names, types, location, roles, capabilities, or other permissions. With security and other features, all XAF applications support personalization features that allow applications to respond differently to different users based on location, user type, or user names.

Table 7: Sparx XAF Module Features

Application Component Explorer (ACE)

The Application Components Explorer (ACE) is a Servlet that provides a browser-based administrative interface to the myriad of Sparx dynamic components and objects. ACE is automatically available to all Sparx-based applications during development and can be optionally available in production.

ACE Features

FEATURE	DESCRIPTION
Automatic Implementation Documentation	Instead of having to create functional specifications and other implementation documentation manually, ACE automatically documents (using the XML definitions and XSLT style sheets) all the forms (web dialogs), SQL statements, schema objects, and other programming artifacts in a centralized browser-based interface. Developers concentrate on application creation while Sparx automatically documents their work. Managers can use this documentation in a real-time basis to track programmer work and productivity.
Application Unit Testing	While developers are working on forms and SQL statements, ACE automatically provides browser-based testing of the forms and statements. No servlets or JSPs, need to be written for basic testing of forms, validations, and SQL statements. Once initial testing is completed and requirements are solidified, then the forms and statements can be aggregated to create interactive applications. End users can use the interactive testing tools to see code as it is being developed (supporting eXtreme Programming concepts).
Project Metrics	As developers create forms, SQL statements, query definitions, JSP, servlets, and other code, ACE automatically maintains basic application metrics. Metrics are an important part of every sophisticated software development process and Sparx can not only capture the metrics but store them in XML files so that they can be analyzed over time.
Application Performance Tracking and Logging	All mission-critical and sophisticated applications need to be tuned for both database and application performance. XAF provides log output that ACE tracks for maintaining data about execution statistics for SQL statements, servlet and JSP pages, dialogs/forms, and security.
Centralized Project Documentation	ACE provides a centralized location for all project documentation for any application. Instead of storing application code and programmer documentation separately, ACE brings tag documentation, Javadocs, MS Office documents, and other project documents into a single easily accessible place. Managers will no longer need to hunt for documents.

Table 8: Sparx ACE Module Features

eXtensible Information Framework (XIF)

The eXtensible Information Framework (XIF) is Sparx's Java and XML library consisting of dozens of reusable tables, columns, and indexes that are useful for almost any e-business application.

XIF Features

FEATURE	DESCRIPTION
Reusable Schemas	Allows for re-use of Schemas across applications and produces and maintains Schema documentation. XIF encourages the creation and re-use of a set of data-types and table-types that define standard behaviors for columns and tables. Data-types and table-types comprise the SchemaDoc database dictionary and can easily be inherited and extended.
SchemaDoc fully describes a schema using XML	Almost all schema objects like tables, columns, data types, etc. are managed in a database-independent XML SchemaDoc. The entire schema is managed in XML as an XML document (a SchemaDoc) and SQL is generated through XSLT style sheets (the templates). The same SchemaDoc can be used to generate database-specific SQL DDL allowing a single XML source schema to work in a variety of SQL relational databases (like Oracle, SQL Server, MySQL, etc.).
Multi-database DDL Generator	Database-specific SQL DDL is created by applying XSLT style sheets to a SchemaDoc. Experienced DBAs are not required to create consistent, high-quality SQL DDL during the design and construction phases of an application. Database-dependent objects like triggers and stored procedures are not managed by the XIF and are created using existing means.
Object-relational Mapping	Database-independent Java Object-relational classes are created by applying XSLT style sheets to a SchemaDoc. This is a called the Application DAL (Data Access Layer). XIF can automatically generate a Java Object-relational DAL (Data Access Layer) for an entire schema, automating the majority of SQL calls by providing strongly-typed Java wrappers for all tables and columns.
Application-Centric	Database Programmers spend time on essential tables and schema elements significant to a specific application instead of rewriting common schema elements for each application.

Table 9: Sparx XIF Module Features

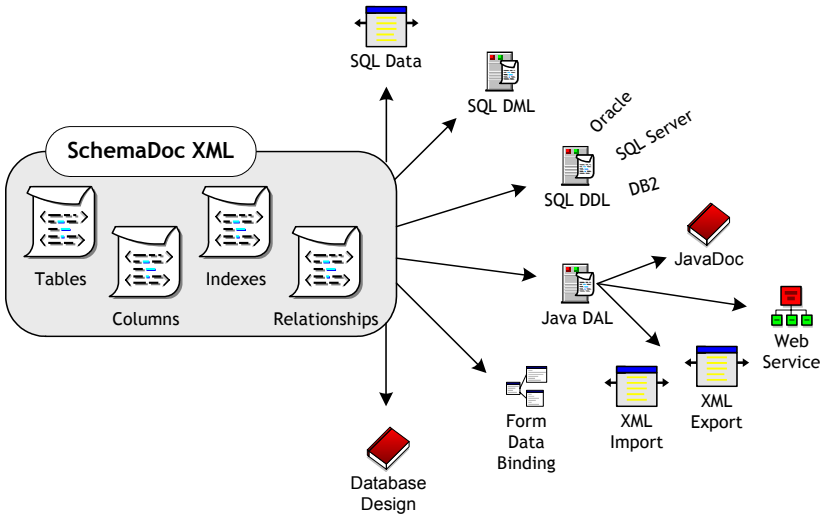


Figure 5: The same XML-based SchemaDoc creates multiple outputs

The diagram above demonstrates how a single XML SchemaDoc (the schema's *executable specification*) allows for multiple outputs like the SQL DDL (for creating tables and columsn in the database), the Java API (for HTML and Java API documentation (for describing the structure to programmers and database administrators), and forms/dialogs data binding.

Sparx Data Access Layer (DAL)

Using XSLT style sheets, the XIF can generate a complete Java O-R map to every table in the schema. This Java O-R map is called the Sparx Data Access Layer, or DAL. Once you have a valid XML SchemaDoc, you can generate the DAL using either a command-line based build scripts (recommended) or through ACE.

DAL Benefits

- The DAL allows strongly-typed Java classes to be generated for each data-type, table-type, and table in the schema.
- The entire schema becomes fully documented through the generation of JavaDoc documentation (the DAL generators generate JavaDoc comments automatically for all classes, members, and methods).
- Each data-type becomes a `Column` Java Class (which an actual table's column becomes an instance of).
- Each table-type becomes a `Table Type` Java interface that each appropriate Table class implements.
- Each table generates specific classes. Assuming a table called `Person`,
- A `Person` Interface is created
- A `PersonRow` class that implements the `Person` domain is created
- A `PersonRows` class that can hold a list of `PersonRow` objects is created
- A `PersonTable` class that contains the column definitions (names, validation rules, foreign key constraints, etc) and SQL generation methods is created.

DAL Features

FEATURE	DESCRIPTION
DataAccessLayer Class	This is the primary class that programmers will use to access their schema through Java. It contains all of the table definitions for the Schema.
Column Classes	Each data-type becomes a Java class that extends the `com.xaf.db.schema.AbstractColumn` class and implements the `com.xaf.db.schema.Column` interface. For example, the "text" datatype becomes a `schema.column.TextColumn` class; the "integer" datatype becomes the `schema.column.IntegerColumn` class; etc.
Domain, Row and Rows Classes	Each table generates three different Java files which represent the data stored in the table.
Table Classes	Each table generates its own `Table` class. For example, assuming a Person table in the database, a table.`PersonTable` class is generated.
API Documentation	The entire DAL API that is generated is fully documented.

Table 10: Sparx XIF Module Data Access Layer (DAL) Features

What's Next

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 kits available at http://www.netspective.com and http://developer.netspective.com will show you:

- How to evaluate Sparx online and on your own system.
- What the major modules of Sparx are and how to use them.
- The Sparx directory structure and the purpose of each entry.
- How to create a Hello World sample application.
- How to use the Sparx build scripts and ACE to help develop your application.
- How Sparx manages the user interface (forms and dialogs) interaction.