

Elite.NET - An Architectural Framework

What is application architecture?

Architecture design is a complex process. It embodies best practice and acknowledged wisdom presenting a set of services, standards, design concepts, components and configurations that guide the development of applications and their architectures.

In addition to this application architecture must be suited to its specific business purpose and that purpose may change with time along with the resources available to you. The task of designing the reliable application architecture can be eased by the architectural framework tools and components.

What are the benefits of using an application framework?

Technology Infrastructure is the core of any enterprise. Making calculated investments in your infrastructure can create business opportunity and drive success; conversely, a poorly structured or managed infrastructure can impede your ability to remain competitive.

Elite.NET is an architectural framework that enables PIE Systems to deliver solutions in a multi-tier environment quickly and effectively. It also ensures that the majority of our time is spent on addressing your business problem(s) and minimizes the time required for building supporting components (aka plumbing). This naturally means that we can develop systems with less effort, lower cost, and faster.

Other benefits include:

- Clearly define the structure of the existing system
- Reduce the number and complexity of the interfaces between the components, improving the ease of:
 - Component upgrade
 - Component exchange
 - Component development and maintenance
 - Code re-use
- Set out the strategy for future developments
- Significantly reduce development time and hence improve time to market
- Improve quality of delivery through re-use and building of a common knowledge

What do we offer?

Elite.NET is an application architectural framework based on sound architectural design patterns.

Elite.NET - Architecture significantly reduces development time through re-use of existing components both bespoke and off the shelf and it allows the development team to focus in developing business requirements and not the "plumbing".

The key features of this architecture are:

- It is based on standard layered architectural design patterns
- XML is the primary data transformation type across boundaries
- All Business and Meta-Objects are XML based
- XML meta-objects are used to define schemas, transformation and processing rules between all layers: Data layer to/from Business Entities to/from the presentation layer
- Web services and pre-built components (med to large grain) are used to facilitate re-use

By selecting a core set of components that we consistently work with to deliver this framework in multiple client environments is where the true advantage is made.

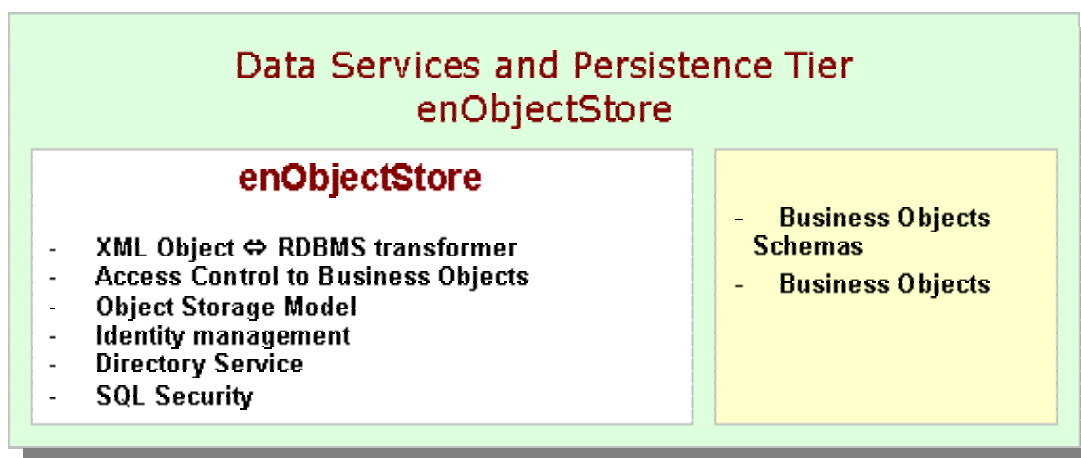
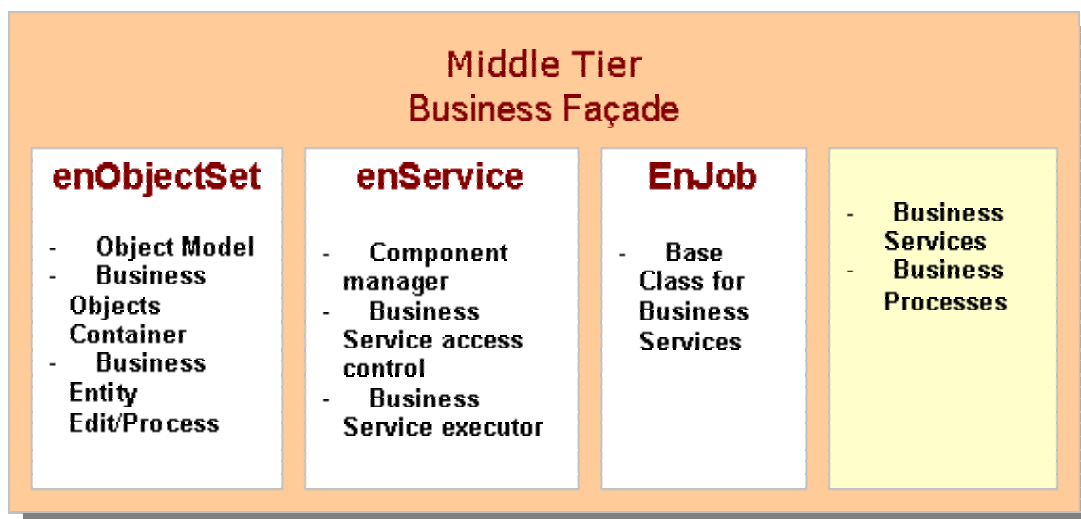
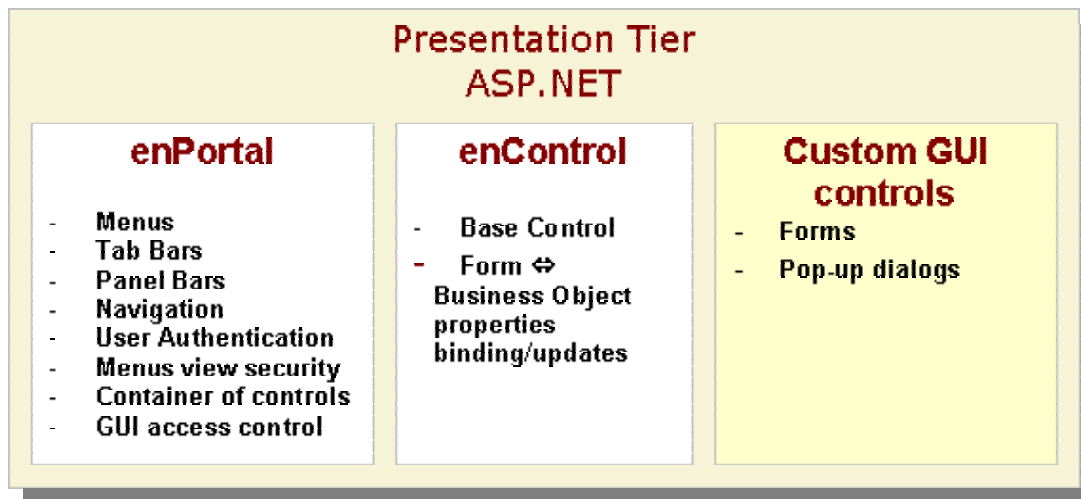
Elite.NET Application Architecture

Elite.Net application model as N-tier distributed component application.

The following is an example of the framework as it would be implemented in a Microsoft .NET environment. The white boxes are areas in which we have existing components, re-usable code, or take advantage of off the shelf software. As you can see, by using this approach, the primary development activity is spent developing business logic (blue-green) that directly benefits your organization.

Elite.Net framework provides

Application provides



Databases, Message Queues, and File Systems

Framework Entities:

Business Entity, Business Object, Meta Object, Object Set, Object Schema, Component, Service, UI Portal, Control, Control Profile, Business Service, Job, Object Store.

Meta-Objects have four categories: configuration, profile, schema, rules.

- Profiles: Control Profile (<name>.enControl), Component Profile (<name>.enService), Job Profile (<name>.enJobControl)
- Schemas: Object schema (<name>.enSchema), ObjectStore schema (Schema.enObjectStore)
- Rules: Business Entity rules (<name>.enObjectSet), Business Service rules (<name>.enRuleSet)
- Configurations: ASP.NET (Web.config), Elite.Net (enWeb.config), ObjectStore configuration (Schema.enObjectStore)

Application Component:

There are two categories of application components: Interactive (UI) and non-Interactive. Interactive components are Web Controls (ascx) or/and Windows Controls (ocx). There are two GUI containers to host these UI controls: enPortal and enIntegrator. enPortal.aspx is Web Page container of ascx controls. enIntegrator is Windows container of Windows OCX Controls. enIntegrator can run IE Web browser control to communicate with enPortal based Web ascx controls.

Non-interactive components are Business Services, which can have many interfaces, but primary interface is “enJob” interface.

To provide “enJob” interface non-interactive component has to inherit base class “enJob”. To run as Elite.Net Job a component has to have enJobControl Profile and to be invoked via Business Façade (enJobExec component) for asynchronous execution by enJob Initiator via enJob Spool or for synchronous execution.

Each UI & non-UI component and their every public function is defined by <component.function>.enService meta-object (Component Profile).

When a component or any of its public function does not have enService meta-object, then this component/function is not available.

Access control security for components/functions is defined by access grants of roles/users to corresponding enService meta-objects.

When role/user does not have the grant to read <component.function>.enService object, then this role/user can not execute this component/function.

All categories of components use enObjectSet container to create/add/update/read/analyze business objects in memory.

UI interactive components cannot use connected data model with pessimistic concurrency (locks) when select/update objects in Database. They use disconnected data model with optimistic concurrency when select/update objects in Database.

Non-interactive components can use both connected and disconnected data model with pessimistic and optimistic concurrency when select/update objects in Database.

Presentation tier and UI client

UI client can be Web Browser or enIntegrator.

Presentation tier includes Web Controls (ascx) or/and Windows Controls (ocx). Presentation tier has two standard Page containers to host these UI controls: enPortal.aspx Web Form and enIntegrator.

enPortal.aspx is ASP.NET based ASPX Web Page container of any ASCX controls.

enIntegrator is Windows container of Windows OCX Controls.

enIntegrator can run IE Web browser control to communicate with enPortal based Web ascx controls.

An access, invocation and execution of ASCX Web controls and OCX Windows controls are managed by enService Component Profiles.

Web User Interface Processing Model

Name space: Elite.Net.UI.Web

Web Menu processing

Presentation tier can use many types of menus for service selection: Panel Bar, Menu bar, context menu, Tool bar, Tab bar, etc.

Each menu type is constructed from “enService” objects/aliases grouped in folders and sub-folders.

Each sub-menu is represented by a sub-folder of enService Profiles.

To display a certain menu enPortal.aspx selects all enService objects/aliases and sub-folders from the folder, which defines that menu.

Web Forms processing

Web Presentation tier container enPortal hosts ascx controls, which perform Web Forms processing.

To invoke any ASCX control a client has to send URL with multiple arguments to enPortal.aspx. Required argument is enService full name. An optional argument is an object name to be processed by the ASCX control specified in enService.

enPortal.aspx reads the specified enService from enObjectStore.

If access to the enService is denied (role or user do not have access grants to the enService), then ASCX control is not invoked.

If access to the enService is permitted, enPortal.aspx gets all required ASCX control information from enService and invokes ASCX control.

Form processing model is based on base control enControl and control XML Profile <name>.enControl. It requires all ascx controls to inherit base class enControl.

Each Web form (ascx control) has to have XML meta-object <name>.enControl to define Form ⇔ Business Entity update/binding rules.

This enControl object is ASCX Control Profile.

Standard ASP.NET provides one way binding only for Grids and Repeaters (Data table => form grid).

Elite.Net provides universal two-way binding/update for all controls (including nested Grids and Repeaters).

Elite.Net Framework CAD/RAD Studio has enControl Builder, which parses specified ASCX control and builds enControl XML meta-object.

Any ASCX control representing Web form has a collection of embedded standard/custom controls. To bind these embedded controls to properties of a Business Entity, enControl meta-object can have bind path for any embedded control. Bind path specifies Object type and XML path to element/attribute within XML object.

Examples:

Account:a\b\c

Account:a\b\c\@d

Account:a\b\c\@z=300\@x (Prefix "@" is used to indicate XML attribute)

All Post-back and submit events can be processed by base class enControl.

For each embedded class "enControl" uses bind path from meta-object <name>.enControl to update Business Entity property from control "value" property and to update control "value" property from Business Entity property.

To access Business Entity "enControl" creates (named) instance of enObjectSet and uses public methods of enObjectSet/enObject. To create named instance of enObjectSet, enControl has to use meta object <name>.enObjectSet as the argument of the constructor.

Meta object <name>.enObjectSet is used to define processing rules for the Business Entity. It can define services to override any function of and enObject (for example: to override enObject XML export/import serialization by invoking XSLT with certain XSL). It can define services to perform custom validation/transformation for cross object or cross elements dependencies (usually invokes RuleSet service to do it).

When Web Form control finishes building/editing Business Entity in enObjectSet container, it can invoke appropriate Business Service or Save/Update a content of enObjectSet in ObjectStore Database.

To save built/updated Business Objects residing in enObjectSet instance, Web Form control can use method "SaveChanges".

To invoke a Business Service to perform further processing of prepared Business Entity, Web Form control invokes enJob Service and passes it enObjectSet instance as the argument.

enJob Service is defined by meta object <BusinessServiceName>.enService with type=enJob. enJob Service is a standard implementation of Business Service.

Business Service can read/update Business Objects in enObjectSet and then can use method "SaveChanges" to save/update built/updated Business Objects residing in enObjectSet instance.

Web Search processing

To perform searches of business objects you need to create two XML meta-objects: SQL search template and Search Object schema.

Search SQL XML template name is <name>.SQL.

Search SQL XML template is created using Elite.Net Framework Studio Query Builder.

Search Object is XML object and elements/attributes of search object are used as a collection of variables in search SQL template.

Business Services Tier

Name space: Elite.Net.enJob

Non-interactive components, which implement Business Rules, are called Business Services.

Non-interactive components can have many interfaces, but primary interface is "enJob" interface.

To provide "enJob" interface non-interactive component has to inherit base class "enJob".

To run as Elite.Net Job a component has to have enService meta object and enJobControl meta object.

All enJob Services are invoked via Business Façade (enJobExec component).

A client can use enJobExec to submit enJob Service for asynchronous execution or for synchronous execution.

enJob Service usually runs as stateless component.

Asynchronous enJob Services are executed by enJob Initiator(s).

All persistent enJob Services are submitted to enJob Spool for an execution and are updated/saved in enJob Spool after execution.

Elite.Net Job Spool is a collection of persistent enJob Service queues and archives.

Every enJob Service is submitted to Spool with enObjectSet, enService, and enJobControl objects.

When enJob Service completes, the result of processing can be found in enObjectSet saved in the Spool.

Business Service can read/update Business Objects in enObjectSet and then can use method "SaveChanges" to save/update built/updated Business Objects residing in enObjectSet instance.

enJob Initiators run as Windows Services (each Initiator is a separate process).

Each Initiator has a queue name assigned to it. Multiple Initiators may have same queue name and read/process enJob Services from the same queue.

enJob Initiator includes enJob Scheduler and enJob Business Process managers.

One enJob Service is Recovery-Termination Manager, which can periodically monitor all jobs in execution state and compare Initiator process ID from executing enJob object to actual Process ID of each Initiator. To do it, Recovery-Termination Manager Selects all enJob objects with status=executing from enJob Spool. All enJob objects in the Spool, which do not have actual Initiator with matching Process ID, are reset to status=system failure and post-processing triggers for system failure event are executed for each job.

Business Entity Tier and Object Model

Name space: Elite.Net.ObjectModel

Business Entity of any type (like Account, Unit, Invoice, PaymentPlan, Policy, Coverage, Job, Product, State, Color, Countertop, etc) is the Business Object of that type and a collection of Elite.Net object management components, system objects and meta objects defining creation, editing, mapping, validation, transformation, triggers, events, and other rules for that Business Entity type.

Primary components for handling Business Entities: enObject, enObjectSet, enSchema.

Primary meta-objects: <type>.enSchema, <name>.enObjectSet, <type>.RuleSet.

Elite.Net Business Objects and Meta Objects use data entities with tree structures.

To provide universal parsing/serialization/search/update/add/delete capability for elements of data entities these tree structures must not be static.

They are standard dynamic tree structures in XML format.

Each Business Objects has to have Elite Schema (XML Meta-object with type=enSchema). Elite Schema defines Tree structure (all elements and attributes names, order & types), mapping of elements/attributes to database tables/columns, and elements/attributes validation rules.

Elite.NET uses three (3) standard classes to work in memory with Business Objects in XML format:

- XMLDocument. It is used to represent full business object with direct access to every node.
- XML stream. It is used for passing data and for memory optimization.
- List of pairs (XML node path, XML node value) – it is a table of multiple lists of pairs (XML node path, XML node value). Each pair includes column Schema (XML node path) and column field (XML node value).

Such tables are used to represent a collection of business objects with partial data, where only nodes used by the component are loaded in memory.

All application components work with data represented by XML based Business Objects. All Business Objects are handled by class enObject and enSchema defining the object type. Class enObject uses enSchema and 3 data entity classes above to manipulate nodes of Business Objects.

When any component works with Business Objects (loads/creates/updates them in memory), it must keep them in the enObjectSet container.
enObjectSet container provides direct access to all stored Business Objects via object name or object path or object ID or object type (for Iterator).
Components must not create Business Object instances in the memory of the component outside of enObjectSet container.
When all processed in-memory objects are located in enObjectSet, a component can find them, serialize enObjectSet into/from XML stream to be passed to remote server, export/import any object, etc.

Meta Object <name>.enObjectSet is used to define processing rules for the Business Entity. It can define services to override/pre-process/post-process any function of enObject for specified Object type or all types (for example: to override enObject XML export/import serialization by invoking XSLT with specified XSL). It can define services to perform custom validation/transformation for inter-object or inter- elements dependencies (usually invokes RuleSet service to do it).

The processing rules in enObjectSet fall into 3 categories: Pre-processing, Processing, Post-processing.

Pre-Processing is used for input validation/transformation.

Processing is used to override standard functions.

Sample Business Entity trigger rules:

```
- <xml>
- <Trigger ObjectType="Account">
- <Event function="Add">
  - <Actions type="PreProcess">
    <Service name="Job1" />
    <Service name="Job2" />
  </Actions>
  - <Actions type="Process">
    <Service name="Job5" />
  </Actions>
</Event>
</Trigger>
- <Trigger ObjectType="">
- <Event function="SaveChanges">
  - <Actions type="PreProcess">
    <Service name="Job6" />
  </Actions>
  - <Actions type="Process">
    <Service name="Job9" />
  </Actions>
  - <Actions type="PostProcess">
    <Service name="Job5" />
  </Actions>
</Event>
</Trigger>
</xml>
```

Data Access Tier

Name space: Elite.Net.Data

Object Data Access is provided by enObjectStore.

enObjectStore can use enObjectSet container to update Database and to load objects from Database to enObjectSet.

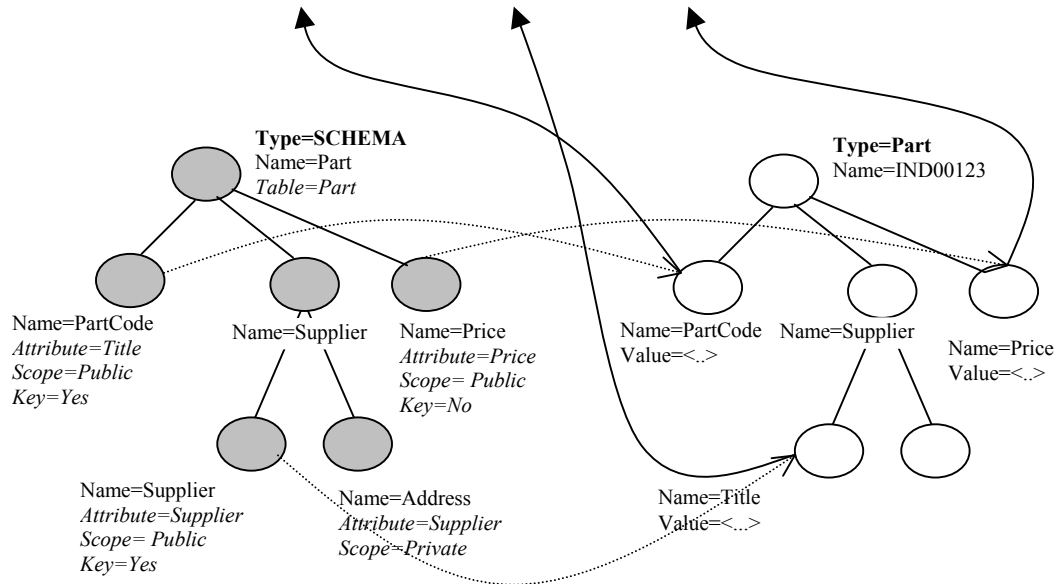
enObjectStore can also accept/return individual objects (not located in enObjectSet).

Every XML business object has a type defined by XML object Schema. XML Object Schema is a key meta-object used for mapping objects to/from relational databases. Schema of a "XYZ" type object is the "enSchema" type object, which has the same tree structure of elements/attributes as a "XYZ" type object and contains meta-information about "XYZ" type objects. Meta-information defines such properties of an object and every attribute as storage, validation rules, mapping to database tables, etc. Use of Schema by Object Store for object/relational mapping allows avoid programming of mapping and other functions for each object type.

XML Object/Relational Mapping using XML Object SCHEMA.

Table "Part"

ObjectID	PartCode	Supplier	Price
----------	----------	----------	-------	-----	-----



Relational/XML Object mapping. Virtual objects technology for integration with existing client/server and legacy applications and upgrading them to Elite.Net applications.

Elite Object Store concept of virtual objects & classification for RDBMS provides facility to build a view of relational tables as XML business objects stored in hierarchical folder system supporting URL links. This view is build externally to application data tables and tables are not changed. Using virtual XML object Wizard developers define objects from attributes of different tables. Wizard automatically generates XML Object Schema for Relational/Object mapping. Schema defines rules on how different attributes of multiple tables united into one object and defines an order of hierarchy for attributes classification. As a result through virtual objects and classification users get an integrated view of data in multiple tables.

The order of objects classification defines virtual folders hierarchy.

After defining an XML object structure and classification a developer defines GUI view of an object. To do it a developer uses Wizard to read .NET form and generate "control" XML meta-object. When an object is referenced by URL link and loaded from Elite Object Store, Elite Web-Integrator automatically builds ASP.NET Form page by filling ASP.NET controls with an XML object attributes using Control Schema.

Using virtual classification and objects for existing applications user can view data and update it. When saving new or modified objects Elite Object Store can directly update application data tables using Object Schema or invoke Application transactions, if API is provided.

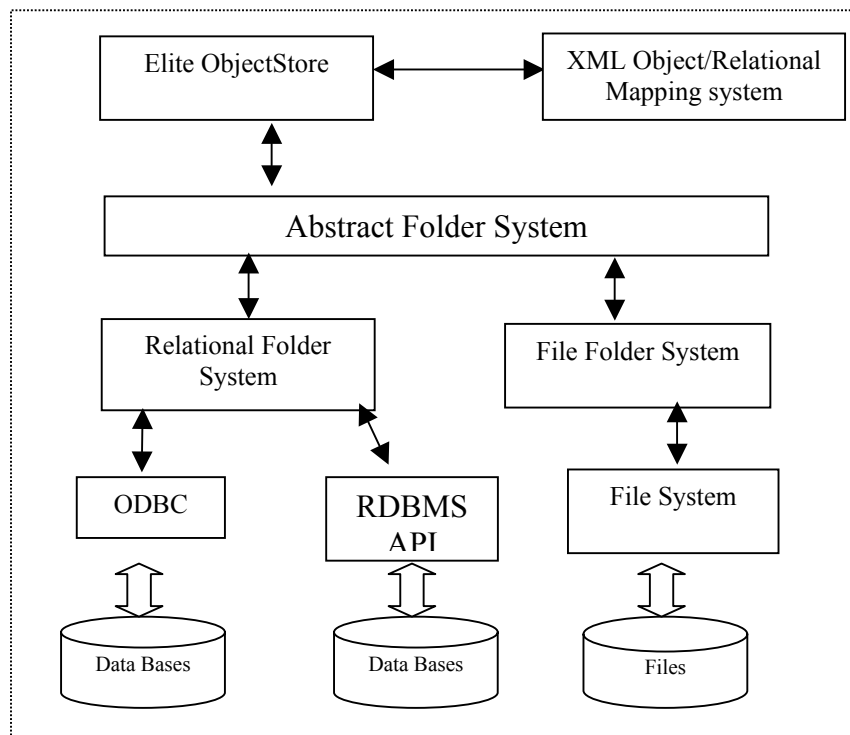
Existing client/server or legacy systems (ERP, Financial applications, Healthcare, Insurance, in-house applications, etc.) usually do not allow users to personalize system and define own views. Many of these systems have accumulated large volume of data, use old graphical user interface (GUI), not WEB enabled, or have limited functions.

Elite.Net Framework allows customers to upgrade existing systems to advanced .NET applications without rewriting them and with minimal efforts.

So customers increase functionality and features of existing applications, while preserving data and old functions.

Object Directory model

Elite.Net ObjectStore can be used to define hierarchical directory view for business objects of any application.



Best Practice Sample Reference Applications Case Study:

1. Pet Shop
2. Survey/Data Capture
3. Account Maintenance

For more information please download best practice reference applications or send us your questions.

PIE Systems International, Inc.

For more information, contact our office:

E-mail: info@piesystems.net

Web: www.piesystems.net

Tel: (310) 925-1208



Copyright © 2003-2004 PIE Systems International, Inc. All rights reserved.