DATA MANAGEMENT (descrizione)

(VER 0.1 / 0807)

Autori: Andrea Guerrieri

Revisioni:

Data	Autore	Note
24/08/2007	Andrea Guerrieri	Prima stesura

1. INTRODUZIONE

Il sistema fornisce all'utente e al progettista un ambiente integrato per manipolare oggetti. Un oggetto è un elemento del mondo reale (es. una fattura, un utente, un documento) descritto da una serie di attributi (es. nome, data di scadenza, telefono, etc). Gli oggetti vengono rappresentati in una struttura gerarchica, per cui ogni oggetto può contenerne altri oggetti-figlio e ogni oggetto (eccetto la radice) è contenuto in uno o più oggetti-padre.
La struttura ad albero è solo una rappresentazione, il sistema permette di astrarre all'utente dove / come vengono fisicamente recuperati gli oggetti. Un ramo dell'albero potrebbe provenire da un file xml, o da una tabella di un database relazione, o potrebbe essere il risultato di una ricerca su internet. Sarà il progettista a scegliere il modo più opportuno per leggere / scrivere gli oggetti all'interno del server, scelta che potrà essere modificata in qualsiasi momento senza conseguenze per il sistema.
Il file system, anch'esso una struttura gerarchica, non è in grado di fornire un sufficiente livello di astrazione dal punto di vista dell'utente. Infatti i file sono solo aggregati di byte, il cui contenuto informativo non è formalmente definito. Per estrarre le informazioni, è necessario affidarsi ad applicazioni esterne, le quali devono decodificare ed interpretare l'insieme di byte, permettendo poi all'utente di manipolare il contenuto. Dare una visione delle politiche di memorizzazione a basso livello (i file) non è utile dal punto di vista dell'utente. Il sistema deve permettere di manipolare "oggetti" indipendentemente dalla loro codifica e collocazione. Immaginate se il vostro telefono vi mostrasse una finestra "salva con nome" all'arrivo di un sms: a noi non interessa dove o come il telefono memorizzi il messaggio, ma solo di riuscirlo a leggere nel momento in cui ne abbiamo bisogno.
L'organizzazione dei file in cartelle è solo un modo per facilitare la ricerca, permettendo all'utente di raggruppare elementi tra loro correlati. Ma le esigenza di ricerca non sono sempre staticamente definite, ad esempio a volte è più comodo vedere le nostre canzoni raggruppate per artista, altre volte per album, o altre per genere, e la struttura file-cartelle non permette questa flessibilità.
Attorno al file system quindi è stato necessario costruiscono infrastrutture (database, e applicazioni) che permettono di estrarre delle viste dei nostri dati, (es. le media library di itunes, o windows media player) ma quello che manca è un sistema integrato che lo permetta per ogni tipo di oggetto.
Lo scopo del progetto è fornire un sistema accessibile da qualunque luogo e piattaforma, nel quale l'utente può memorizzare e trattare le informazioni a lui utili, che sia estensibile, integrato, e che dia una visione omogenea dei contenuto indipendentemente da dove essi provengano o di come siano codificati. L'ambizione è di fondere i concetto di "file, file system, database, servizio, rete, protocollo, applicazione" nei semplici concetti di Oggetti / Contenitori / Servizi.

2. ORGANIZZAZIONE DEL SISTEMA

I Tipi

Si intende per tipo una categoria al quale un oggetto può appartenere (es. “Fiat punto” è un oggetto di tipo “Automobile”, “Mario Rossi” un oggetto di tipo “Contatto”, mentre “ocean’s 12” un oggetto di tipo “Film”)
Ogni tipo ha un nome univoco per il sistema, ed è descritto da una serie di attributi. Un attributo non è altro che una coppia nome / valore e serve a descrivere una caratteristica dell’oggetto (es. “autore” e “titolo” sono due attributi per un oggetto di tipo “libro”). Il valore dell’attributo può essere di un tipo primitivo (testo, numeri, date, vero/falso), oppure un tipo del sistema (Utente, Contatto, etc) I tipi nel sistema non sono staticamente definiti, ogni modulo può registrare i propri tipi e renderli disponibili all’applicazione tramite il *TypeServer*.
Per ogni tipo è inoltre definito un attributo che identifica univocamente l’oggetto chiamato identificatore o id. Questo significa che se due oggetti hanno lo stesso id e sono dello stesso tipo, devono essere necessariamente uguali. Normalmente l’id è un attributo che viene aggiunto e che non ha nulla a che vedere con l’oggetto reale.

Oggetti e container

La struttura ad albero viene fornita tramite l’uso di container. Il container è un oggetto in grado di contenere altri oggetti. Ogni oggetto all’interno di un container è identificato dalla coppia tipo / id, Una sequenza di coppie Tipo - Id costituiscono un Path, ovvero un percorso a partire dalla radice che permette di identificare un oggetto nell’albero.

Esempio:
Il contatto è un oggetto con alcuni attributi (Nome, Data di Nascita, Codice fiscale, etc), ma anche un container di Indirizzi, Numero di Telefono, etc (correlati al contatto). L’indirizzo è un oggetto anch’esso con degli attributi (Città, Via, etc) sul quale è possibile compiere delle operazioni (es. visualizzarlo su una mappa, inoltrare un ordine di spedizione, etc)
Per Raggiungere a partire dalla radice l’oggetto di tipo “Indirizzo” chiamato “Sede legale” è necessario specificare questo path **Root/ContactFolder/ContactGroup[Aziende]/Contact[23]/Address[7]** dove il testo tra parentesi quadre rappresenta l’id, mentre quello fuori rappresenta il nome del tipo.

Tipi di oggetto

Gli oggetti dell’albero si possono dividere in 7 categorie. Questa divisione non è vincolante, ma rappresenta semplicemente gli scenari piu comuni di utilizzo:

Oggetto semplice
Oggetto che non contiene figli, chiamato anche foglia dell’albero. Rappresento un dato, un elemento primitivo del sistema.

Oggetto container in sola lettura statico
Oggetto che si comporta da container mostrando alcuni dei suoi attributi / relazioni sottoforma di oggetti figlio (es. oggetto “Telefono” per l’attributo “numero di cellulare” per l’oggetto “Contatto”)

Container di sola lettura dinamico
Container il cui contenuto non può essere scritto dall'utente, ma solamente da moduli dell'applicazione (es. ogni modulo può aggiungere oggetti al nodo radice)

Container di ricerca
Container filtro per avere una vista di oggetti tipicamente forniti da altri container che rispondono a determinate requisiti (es. Tutti i messaggi inviati da un contatto, o tutti i brani di un dato autore)

Container di ricerca esterni
Container che estraggono oggetti a partire da dati non strutturati (es. parsing di pagine html)

Container di scrittura temporanei
Container che mantengono gli oggetti scritti solo per un periodo limitato di tempo, e comunque solo fin quando l'applicazione non viene chiusa. Vengono tipicamente usati per appoggiare oggetti che dovranno subire delle trasformazioni / elaborazioni (es. un oggetto "messaggio" nel container "Posta in uscita" viene rimosso appena il messaggio è stato inviato)

Container di scrittura persistenti
Container che mantengono gli oggetti scritti in modo permanente, tipicamente legati a tabella di un database, file xml, cartella del filesystem, etc

Operazioni

L'utente deve poter manipolare gli oggetti, a questo fine è previsto che su ogni oggetto si possano definire delle operazioni. L'operazione non è altro che un azione che l'utente compie sull'oggetto, per poterne estrarre delle informazioni, o cambiare il suo stato, etc. Alcune operazioni tipiche sono "visualizzare" l'oggetto, modificare i suoi attributi, stampa, etc

Operazioni non contestuali
Sono operazioni valide indipendentemente da dove si trova l'oggetto all'interno dell'albero, tipicamente legate al tipo. (es. stampa, visualizza, etc)

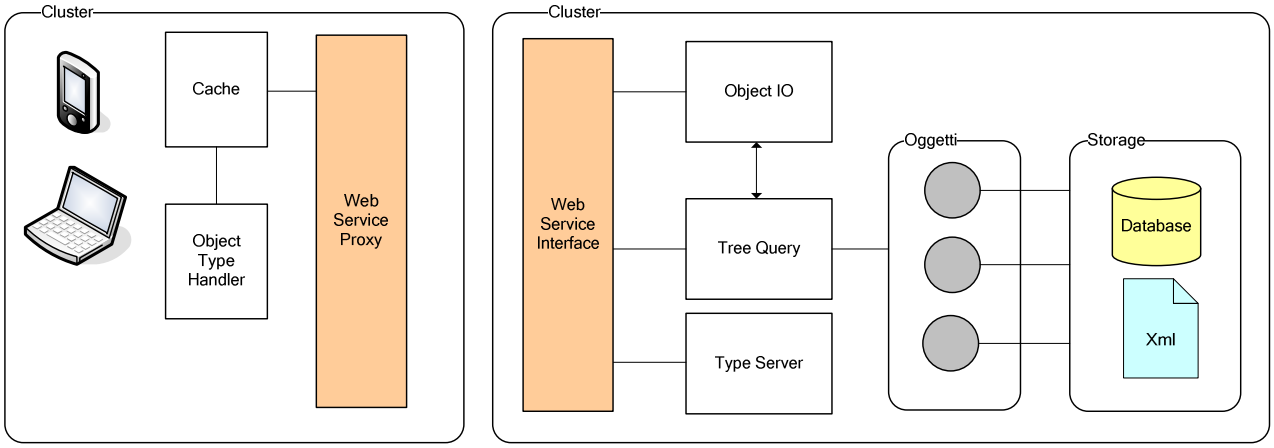
Operazioni contestuali
Sono operazioni eseguite dal container sull'oggetto, e sono valide sono in determinati contesti (es. "elimina" è possibile solo se l'oggetto si trova su un container di lettura/scrittura)

Operazioni server side
Operazioni che richiedono un elaborazione server side, e non necessitano di interazione da parte dell'utente, se non per l'immissione di eventuali parametri. (es. il calcolo del codice fiscali a partire dai dati anagrafici del contatto)

Operazioni client side
Operazioni definite dal client, che solitamente comportano un iterazione da parte dell'utente (come la modifica dei dati di un contatto, l'ascolto di un brano musicale, etc), e che possono avere significato / implementazione diversa da client a client (es. l'operazione stampa non ha significato su un palmare)

3. INFRASTRUTTURE

Il sistema adotta il modello client – server, dove per “server” si intende tutta la parte applicativa che gestisce il catalogo degli oggetti, mentre per “client” la parte che interroga il catalogo e permette di manipolare gli oggetti. Client e server possono risiedere nello stesso processo, in processi differenti o in macchine differenti. Nell’implementazione standard il server espone i suoi servizi tramite web service usando http come protocollo di trasporto e soap come sistema di codifica dei messaggi.



Moduli

L’applicazione è totalmente modulare, rispettando alcune convenzioni è possibile installare un modulo semplicemente copiando l’assembly nella cartella dei moduli aggiuntivi. Un modulo aggiuntivo permette di definire nuovi tipi di oggetto, o di estendere quelli esistenti (aggiungendo ad esempio operazioni, o dando una implementazione alternativa di quelle esistenti). Il modulo contiene normalmente sia la parte server, che la parte per il client standalone .net. Questo per ridurre al minimo il numero di dll da distribuire, poiché la parte di definizione degli oggetti è comune sia a client che a server. L’applicazione minimale è costituita dai seguenti sotto-progetti:

eusoft – Eusoft.dll
Libreria sviluppata dall’autore che estende il .net framework, aggiungendo elementi di interfaccia utente, la gestione dei servizi, la gestione dei dati, etc.

objectTree – ObjectTree.dll
Core dell’applicazione, definisce le interfacce e le classi per la gestione dell’albero (oggetti, container, operazioni) , contiene tutti gli elementi del server http, nonché gli elementi principali del client standalone.

mainModule – ObjectTree.MainModule.dll
Modulo principale che contiene tutti i tipi e gli oggetti primitivi comuni a tutte le applicazioni, tra i quali l’oggetto “Utente”, “Contatto”, e la gestione dei permessi e le autorizzazioni.

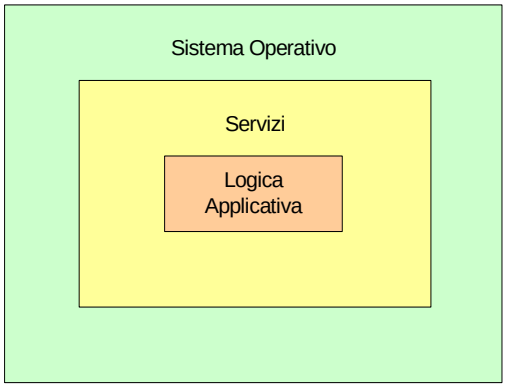
desktopClient – dmclient.exe
Contiene l’eseguibile per il client .net standalone.

Strutture comuni client / server

Realizzare un applicazione significa descrivere un processo che permetta, attraverso degli strumenti, di realizzare le specifiche. Questo processo è del tutto simile ad un qualsiasi altro processo di costruzione, anche in ambito non informatico. Ci sono le materie prime necessarie alla lavorazione, i macchinari che operano sulle materie prime, e ne facilitano l'assemblaggio e la trasformazione, un piano che indichi come usare i macchinari e le materie prime per realizzare il lavoro, e il personale che esegue il lavoro.
Anche nell'applicazione si possono individuare questi quattro elementi: le materie prime sono raffigurate dalle classi e gli oggetti che rappresentano i dati (*entità*), i macchinari da interfacce chiamate *servizi*, il piano di lavoro dalla *logica applicativa*, mentre il personale dal sistema hardware/software.

Entità
Rappresentano i dati, che possono essere usati e scambiati tra l'applicazione e i servizi. Le entità normalmente non contengono metodi e operazioni, sono solo parti di scambio.

I Servizi
Sono parti satellite, che non realizzano la logica applicativa, ma elementi di supporto che forniscono tutto ciò di cui l'applicazione ha bisogno. Un servizio rappresenta un concetto che dovrebbe essere immutabile e indipendente dalla realizzazione concreta. In base a questa logica, è possibile astrarre tra il cosa è necessario e il come questo venga fornito, permettendo di scegliere di implementare in modo diverso alcune parti dell'applicazione, senza compromettere la stabilità e il funzionamento dell'intero sistema. Questa struttura facilita inoltre il porting dell'applicazione tra sistemi diversi, permettendo di lasciare invariata la logica applicativa, e cambiare (laddove sia necessario) solo l'implementazione di alcuni servizi. Infine lo stesso servizi può essere isolato, e utilizzato anche per applicazioni differenti, poiché è molto probabile che due applicazioni abbiamo bisogno degli stessi servizi.



Logica applicativa
Usa i servizi e le entità al fine di realizzare le specifiche.

Servizi comuni

Esiste un'unica classe statica a livello di applicazione, chiamata *AppService*. Questa classe mantiene il catalogo di tutti i servizi dell'applicazione, che possono sia essere richiamati sia tramite il metodo generico *Get*<> (passando come argomento l'interfaccia / servizio), sia tramite proprietà. Qui elencati i servizi comuni a tutte le applicazioni:

Application - *IApplication*
Rappresenta l'applicazione stessa, e contiene i metodi per l'avvio / terminazione.

Language - *ILanguageService*
Fornisce il servizio di traduzione di stringhe di testo in diverse lingue.

ObjectConversion - *IObjectConversionService*
Servizio di conversione tra oggetti di tipo diverso.

Logger - *ILogService*:

Fornisce un registro nel quale l'applicazione può scrivere informazioni diagnostiche, tipicamente usato per riportare bug e eccezioni.

Modules - *IModulesService*
Gestisce i moduli aggiuntivi (plug-in) per l'applicazione.

Animation - *IAnimationService*
Gestisce le animazione, ovvero la transizione temporale di un attributo di un oggetto tra due valori, secondo una certa funzione di interpolazione, tipicamente usato per applicazioni grafiche.

Panels - *IPanelsService*
Mantiene il catalogo di tutti i pannelli del sistema, e il gestore del layout dei pannelli. Per maggiori informazioni vedere la sezioni "Interfaccia a pannelli" nel modulo "Client"

Storage - *IStorageService*
Mantiene il catalogo di tutti gli storage. Per maggiori informazioni vedere la sezione "Storage" nel capitolo "Server".

Authentication – *IAuthenticationService*
Gestione utenti, e validazione nome utente e password.

UserSession – *IUserSessionService*
Mantiene le informazioni di sessione per ogni utente che accede all'applicazione. Per sessione si intendono tutte le informazioni di stato specifiche per utente.

Database – *IDatabaseService*
Fornisce il servizio di connessione ad un DBMS, permettendo di inoltrare comandi ed effettuare query, supponendo l'esistenza di un unico database per applicazione.

ObjectEncoding – *IObjectEncodingService*
Mantiene il catalogo di tutte gli encoder / decoder. Un encoder è una classe che permette di trasformare un oggetto in uno strema di byte, secondo un certo formatto (mime type), mentre un decoder si occupa del processo inverso.

Modelli di connessione client / server

Come accennato in precedenza, la divisione client / server è puramente concettuale, una semplice divisione di competenze. A seconda degli scenari di utilizzo del sistema e' possibile scegliere tra più di un modello di connessione.

Server e client .net standalone nello stesso processo
Modello adottato quando l'applicazione viene usata principalmente in una sola macchina, fonde i componenti client / server in un unico processo, e permette di evitare l'installazione di un web server, e l'overhead della comunicazione http/xml

Server e client .net standalone nello stesso processo, con banca dati remota
Simile al modello precedente, con l'unica differenza che la banca dati è condivisa e situata in una macchina dedicata. Modello ottimale nel caso si vogliano avere maggiori prestazioni ed evitare il server web, ma con l'accesso diretto alla banca dati è possibile che venga a meno la sicurezza.

Server http / client qualunque

Modello più versatile nel quale il server risiede in una macchina dedicata e espone i suoi servizi sottoforma di webservice, permettendo ad un qualunque client di connettersi tramite http (anche sviluppato in piattaforme / linguaggi diversi). Il server http verifica le autorizzazioni, e maschera l'accesso alla banca dati, ma introduce un forte overhead sia in termini di banda che di prestazioni dovuto alla codifica / decodifica xml degli oggetti.

Il server

Costituiscono la parte "server" tutti i servizi che si occupano della gestione degli oggetti (memorizzazione e ricerca). Alcuni di questi servizi sono esposti sottoforma webservice. La libreria *objectTree* contiene tutto il codice sorgente, compresi i webservice e le webform. Il server http semplicemente dipende da questa libreria, e dichiara il file aspx / asmx che derivano da classi contenute in questa libreria.

Storage

Esistono diversi sistemi per memorizzare in modo persistente delle informazioni, e a secondo delle esigenze, uno può risaltare più vantaggioso dell'altro. Ma nel ciclo di vita di un'applicazione, queste esigenze possono cambiare, ed è quindi spesso necessario cambiare sistema per meglio rispondere ai nuovi requisiti. Inoltre sarebbe bello poter scegliere, a livello di entità e non globalmente, un luogo dove scrivere dati, senza perdere il concetto di relazione. (es. mettere in relazione un utente che sta sul database di windows, con una tabella che è in un database sql). Fortunatamente la diversità tra questi sistemi è puramente fisica, dal punto di vista concettuale tutti forniscono la stessa astrazione.
Uno storage rappresenta un sistema nel quale memorizzare oggetti dello stesso tipo. Lo storage, rappresentato dall'interfaccia generica *IStorage*<>, contiene tutti i metodi per aggiungere, eliminare, aggiornare, e ricercare gli oggetti contenuti.
Il servizio di Storage mantiene il catalogo di tutti gli storage presenti nel sistema, cosi, dal punto di vista sistemistico, è possibile associare ad ogni tipo lo storage più idoneo, e da quello della logica applicativa usare lo storage in modo astratto, ignorando come/dove fisicamente gli oggetti vengono memorizzate. Con questa tecnica, è possibile mettere in relazione anche dati che si trovano in sistemi diversi, come una tabella di un database, con un file xml, e soprattutto avere la liberta di cambiare storage, senza apportare modifiche alla logica dell'applicazione.
Grazie ad un apposito generatore, è possibile definire le entità a livello concettuale, lasciando a quest'ultimo il compito di implementare in modo opportuno le interfacce *IStorage*, in base al sistema di memorizzazione scelto.

TypeServer

Rappresentato dall'interfaccia *ITypesService*, permette ad un modulo di registrare / leggere le informazioni su un tipo (vedere *TreeObjectTypeInfo*). Alcune informazioni sul tipo sono statiche (ovvero non dipendono dal valore degli attributi di una sua istanza) e altre sono dinamiche. Questo servizio tiene conto solo delle informazioni statiche e mantiene inoltre il catalogo di tutti gli attributi di cui è composto un tipo. In linea teorica, non dovrebbero esistere attributi con lo stesso nome di significato diverso e ogni attributo dovrebbe essere univoco per il sistema. (es. "nome" "eta" "citta" dovrebbero avere lo stesso significato, indipendentemente dal tipo che le dichiara). Essendo comunque attualmente un vincolo troppo forte, è possibile registrare attributi con lo stesso nome, ma con significato diverso da tipo a tipo.

ObjectTree

Come descritto in precedenza un oggetto può comportarsi da contenitore di altri oggetti. La relazione contenitore / contenuto fornisce all'intero sistema una struttura ad albero, per cui è lecito definire un oggetto "nodo" dell'albero, e un container "ramo" dell'albero.

Questo servizio (interfaccia *ITreeObject*), fornisce l'astrazione di albero, permettendo di identificare gli oggetti tramite path. Alla base dell'albero c'è l'oggetto radice, un container a cui ogni modulo può accodare figli (che possono essere a loro volta altri container). Tramite il metodo *Query*, è possibile avere l'elenco di tutti i figli di un container (identificato da un path) che soddisfano dei criteri di ricerca. Un ricerca semplice permette di filtrare gli oggetti in base al loro tipo e al valore dei loro attributi (es. tutti gli oggetti di tipo Contatto con l'attributo "città di residenza" uguale a "Roma"). Se il container è collegato ad uno storage, la ricerca verrà delegata a quest'ultimo, altrimenti si farà uso della reflection rendendo la ricerca estremamente lenta e costosa.
E' possibile effettuare anche ricerche in profondità, ovvero che non coinvolgono solo i figli diretti del container, ma anche i discendenti (figli dei figli). Poiché la struttura ad albero è puramente virtuale, è possibile che in questo tipo di ricerca si creino dei loop a causa di riferimenti circolari (esempio più semplice, un figlio con un figlio che punti al padre), per cui la ricerca "in profondità" non è abilitata su tutti i container.
Il risultato di una query, non sono direttamente gli oggetti, ma una lista di classi *TreeObjectInfo*. Questa classe contiene tutte le informazioni utili al client per rappresentare e identificare l'oggetto, tra cui id, tipo e nome. La scelta del nome è delegata dall'oggetto stesso, e normalmente è una combinazione del valore dei suoi attributi (es. il nome di un oggetto di tipo "Contatto" è dato dagli attributi "titolo" + "nome" + "cognome", quello di un documento fiscale da "Fattura " + numero progressivo, etc), e sebbene sia permesso avere oggetti diversi con lo stesso nome, è opportuno scegliere una combinazioni di attributo che limiti le ambiguità.
Il servizio permette anche di enumerare tutte le operazioni server side, contestuali o non, di un oggetto dell'albero, ma non la loro esecuzioni. Alcune operazioni possono richiedere molto tempo per il loro completamento, ed è quindi necessario dare un riscontro all'utente sullo stato di avanzamento di queste operazioni. Gli oggetti che eseguono operazioni lente lato server, possono implementare l'interfaccia *IExecutionProgress*, e tramite il metodo *GetOperationProgress* di questo servizio, ricevere informazioni sullo stato di avanzamento.

ObjectIO
A causa di una forte limitazione nell'implementazione dei webservice su ASP.NET 2.0, non è possibile usare i tipi generici all'interno di un webservice, ovvero i file asmx che rappresentano il servizio, non possono derivare da un tipo generico ne esplicitando gli argomenti ne passandoli come parametro. Inoltre per problemi strutturali gli argomenti, e i valori di ritorno di un webservice devono essere staticamente definiti (con WSDL) di conseguenza un metodo di un webservice, non può accettare come parametro o valore di ritorno un oggetto che sia una sotto-classe di quello dichiarato (es. non è possibile dichiarare un metodo che accetti un *object* come parametro).
A causa di questi problemi non è possibile avere un servizio generico che permetta di scambiarsi oggetti tra client e server, ma ogni tipo deve avere il proprio servizio di IO. Il servizio è rappresentato dall'interfaccia generica *ITreeObjectIO<>*, ed è implementato nella classe *ApplicationTreeObjectIO<>* che può essere usata nella maggior parte dei casi. Compito di questo servizio, è fornire le primitive di IO tra client e server, tra le quali lettura, aggiornamento, e inserimento e validazione di un oggetto.
Alcuni oggetti possono dichiarare operazioni server side oppure hanno bisogno di una gestione particolare (es. nel caso di un utente, il cambio della password non può essere fatto con una semplice update, ma per motivi di sicurezza bisogna fornire sia la vecchia che la nuova password).
devono essere raggruppate in una particolare interfaccia implementata dall'objectIO. Ogni operazione server side, ha normalmente un corrispettivo metodo all'interno dell'oggetto, per cui compito di questa interfaccia è recuperare l'oggetto a partire dal path, verificare le autorizzazioni, invocare il metodo, e restituire il valore di ritorno.
In conclusione, possiamo dire che il servizio *IObjectTree*, da solo una visione descrittiva dell'albero, e contiene tutti metodi che non comportano trasferimento del oggetto tra client e server.

L’objectIO invece permette di trasferire l’oggetto da server a client (e viceversa), e di eseguire su di esso operazioni specifiche per quell’oggetto.

Report

Il Client

Interfaccia a pannelli

Operation e Object Handler

L’ ObjectHandler ha il compito di gestione degli oggetti di un certo tipo lato client. Questa gestione comprende:

- Implementare le operazioni comuni
- Restituire l’objectIO tramite il quale leggere / scrivere il tipo gestititi

Dare una rappresentazione visiva lato client (icona, etc)

4. CREAZIONE DI MODULI AGGIUNTIVI

Una classe, per poter essere un oggetto dell'albero, deve implementare l'interfaccia *ITreeObject*. Questa interfaccia contiene solo il metodo *GetInfo*, che deve restituisce una struttura *TreeObjectInfo* (vedere la sezione ObjectTree). I container di sola lettura invece, devono implementare anche *ITreeObjectContainer*. Questa interfaccia aggiunge i metodi per fare query sul container, e recuperare un oggetto contenuto a partire dal suo id / tipo. I container di lettura/scrittura infine devono implementare *IWriteableTreeObjectContainer*, che contiene anche i metodi per aggiungere, aggiornare, eliminare un oggetto all'interno del container.