CMDBuild

Open Source Configuration and Management Database

Technical Manual

Versione 0.6.0
Giugno 2007

CMDBuild leverages many great technologies from the open source community:
PostgreSQL, Apache, Tomcat, Eclipse, JasperReports, IReport
We are thankful for the great contributions that led to the creation of that products

CMDBuild è un progetto realizzato e gestito da:

Comune di Udine – Servizio Sistemi Informativi e Telematici

Tecnoteca S.r.l. (www.tecnoteca.com)

Cogitek S.r.l. (www.cogitek.it)

Il sito ufficiale di CMDBuild è http://www.cmdbuild.org

Sommario

Introduzione

CMDBuild è una applicazione Open Source per la configurazione e gestione del database della configurazione (CMDB) degli oggetti in uso presso il Dipartimento IT di una organizzazione.

Gestire un Database della Configurazione significa mantenere aggiornata e disponibile per gli altri processi la base dati relativa agli elementi informatici utilizzati, alle loro relazioni ed alle loro modifiche nel tempo.

CMDBuild si ispira alle “best practice" ITIL (Information Technology Infrastructure Library), ormai affermatesi come "standard de facto", non proprietario, per la gestione dei servizi informatici secondo criteri orientati ai processi.

Con CMDBuild l'amministratore del sistema può costruire autonomamente il proprio CMDB (da cui il nome del progetto), grazie ad un apposito programma di configurazione che consente di aggiungere progressivamente nel sistema nuove classi di oggetti, nuovi attributi e nuove tipologie di relazioni.

Tramite il sistema implementato per la gestione del workflow è poi possibile definire in modo visuale con un editor esterno nuovi processi operanti sulle classi trattate nel sistema, importarli in CMDBuild ed eseguirli secondo i criteri indicati.

Sono infine disponibili sistemi di interfaccia con applicazioni open source esterne specializzate in attività connesse a CMDBuild (Automatic Inventory, Help Desk e prossimamente Document Management). Per le modalità di installazione di tali interfacce si rimanda allo specifico manuale.

I moduli di CMDBuild

Il sistema CMDBuild comprende due moduli principali:

- il Modulo Schema, dedicato alla definizione iniziale ed alle successive modifiche della struttura dati (classi e sottoclassi, attributi delle classi, tipologie di relazioni fra classi)
- il Modulo Gestione Dati, dedicato all'inserimento ed aggiornamento nel sistema dei dati descrittivi e delle relazioni funzionali fra le diverse entità, alla produzione di report e tabulati, nonché alla definizione e controllo dei processi per la gestione dei servizi informatici.

Il presente manuale è dedicato ai tecnici informatici cui sono demandate le attività di installazione delle componenti software e di amministrazione del database.

Sono disponibili sul sito di CMDBuild (http://www.cmdbuild.org) manuali specifici dedicati a:

- Overview concettuale del sistema
- Administrator Manual
- User Manual
- Workflow Tutorial
- External Connectors

Installazione del sistema

L'installazione di CMDBuild richiede l'utilizzo di uno o più server su cui suddividere le componenti logiche costitutive del sistema:

- server web
- componenti di elaborazione
- database

Vengono descritti di seguito i requisiti software richiesti da CMDBuild, le modalità di installazione e configurazione, i criteri di verifica e test.

Nella progettazione dell'infrastruttura sistemistica va considerato che l'attivazione di applicazioni web come quella in oggetto richiede la disponibilità di componenti hardware e di rete dotate di adeguati livelli di sicurezza, sia rispetto accessi esterni indesiderati (firewall, DMZ) che rispetto le esigenze di disponibilità continuativa on line del sistema (backup della linea internet, mirroring hardware) e di adeguate prestazioni di accesso.

Requisiti hardware

Per l'installazione di CMDBuild è richiesto un computer di classe server di recente generazione, avente le seguenti caratteristiche dimensionali minime:

- memoria RAM minimo 1 GB (consigliati 2 GB)
- spazio disco minimo 20 GB (consigliati 40 GB in previsione del prossimo rilascio della funzione di gestione allegati)

Sono altresì consigliati:

- la presenza di un sistema di dischi in configurazione RAID
- un sistema di backup giornaliero dei dati
- un sistema di continuità elettrica per preservare il server da interruzioni anomale dell'alimentazione

Requisiti software

L'installazione di CMDBuild richiede la presenza dei componenti software di seguito elencati.

Sistema operativo

Qualunque sistema operativo supporti gli applicativi sotto elencati (consigliato il sistema operativo Linux sul quale CMDBuild è soggetto a test più estesi).

Database

PostgreSQL 8.0 o superiore (consigliato 8.1), l'installazione base è sufficiente, accertarsi che sia attivato il supporto al linguaggio "plpgsql".

Sito di riferimento: http://www.postgresql.org/

Web Server

CMDBuild richiede l'istallazione di Apache 2.0 e di Jakarta Tomcat 4.5 o superiori (consigliato 5.0, Tomcat 6.0 non è al momento supportato).

Per la versione Tomcat 5.5 sono richieste librerie addizionali, posizionate nella cartella “tomcat55libs”.
Sito di riferimento per entrambi: http://www.apache.org/

Librerie Java

Le librerie Java sono necessarie per il funzionamento di Jakarta Tomcat.
CMDBuild richiede JDK 1.5 o superiori.
Sito di riferimento: http://www.sun.com/

Librerie incluse nel rilascio

CMDBuild contiene una serie di librerie già all'interno del pacchetto di installazione, ed in particolare:

- postgresql-8.0-313.jdbc3.jar per il collegamento a PostgreSQL 8.0
- commons-dbcp-1.2.jar per la gestione del pool di connessione al database
- jasperreports-1.2.2.jar per la produzione di report
- shark* per l'utilizzo del motore di workflow Enhydra Shark
- struts* e commons*, librerie utilizzate da Struts 1.2

Ulteriori informazioni su Struts si possono trovare sul sito di riferimento: http://struts.apache.org/

Per l'eventuale disegno di report custom è utilizzabile l'editor visuale IReport che produce il relativo descrittore in formato compatibile con il motore JaspertReports.

Per l'eventuale disegno di workflow personalizzati è suggerito l'editor visuale JPEd dal quale viene prodotto il relativo descrittore in formato XPDL compatibile con il motore Enhydra Shark.

Tutti i software sopra elencati sono rilasciati con licenza Open Source (ad eccezione eventualmente del Sistema Operativo se si optasse per una soluzione diversa da Linux).

Installazione del sistema

Per l'installazione del sistema può essere predisposta una partizione unica, oppure due partizioni dedicate rispettivamente ai programmi e ai dati. Nella seconda ipotesi la partizione riservata ai dati comprenderà sia le tabelle del database PostgreSQL che una cartella sul filesystem ove archiviare i file allegati alle schede di database (immagini, documenti, ecc).

Come tipo di filesystem in ambiente Linux possono essere utilizzati indifferentemente “reiserfs” o “ext3”, sono sconsigliate altre opzioni meno sperimentate.

Contenuti del rilascio

Il file “tar.gz” rilasciato contiene i seguenti componenti:

```
INSTALL.txt                 file guida, i cui contenuti sono riportato
                            ai punti successivi del presente documento
GPL.txt                     termini della licenza
CMDBuild.Sql                comandi SQL per la creazione di un database vuoto
CMDBuild-demo.Sql           comandi SQL per la creazione di un database di test
shark.sql                   comandi SQL per la creazione dello “schema” di Shark
postgresql-8.0-313.jdbc3.jar Postgres Drivers
\tomcat55libs               Librerie addizionali richieste da Tomcat 5.5 e 6.0
\cmdb                       file di CMDBuild per Tomcat
\cmdb\WEB-INF\src           codice sorgente
\cmdb\WEB-INF\lib           librerie Java
\cmdb\WEB-INF\classes       classi Java
```

Installazione del database

Per l'installazione del database vanno effettuate le seguenti operazioni:

- tramite uno strumento con interfaccia grafica (ad esempio pgAdmin3 di PostgreSQL) o da linea di comando creare il database utilizzando il nome “cmdb” o altro a scelta:

```
CREATE DATABASE cmdb
WITH OWNER = postgres
ENCODING = 'UNICODE'
TABLESPACE = pg_default;
```

- accertarsi che l'encoding sia UNICODE oppure UTF8 (per PostgreSQL v. 8.1 e seguenti)
- eseguire i comandi SQL contenuti nei file CMDBuild.sql o CMDBuild-demo.sql (quest'ultimo contiene dei dati di esempio). Se si è creato il database tramite interfaccia grafica (ad es. PGAdmin) accertarsi che non esista nessun language associato al database, in caso contrario commentare (aggiungendo all'inizio i caratteri “--”) le seguenti righe ad inizio file:

```
-- CREATE TRUSTED PROCEDURAL LANGUAGE 'plpgsql'
-- HANDLER plpgsql_call_handler;
```

Nel caso si voglia utilizzare le funzionalità di workflow è necessario aggiungere nel database appena creato anche lo “schema” ulteriore riservato alle tabelle dati richieste da Shark, che dovrà avere nome “shark”.

Per fare questo è sufficiente, con la stesso strumento già utilizzato (presumibilmente pgAdmin3), connettersi al database di CMDBuild ed eseguire lo script “shark.sql” contenuto nel rilascio. Lo stesso script, oltre a creare il nuovo “schema” e gli oggetti richiesti da Shark, crea anche il “ruolo utente” “shark” utilizzato dal motore Enhydra per connettersi al database (da specificare nel file di contesto descritto successivamente).

Attenzione: essendo il “ruolo utente” definito non a livello di singolo database, ma di “database server”, nel caso si volessero attivare più istanze di CMDBuild si dovrà, prima di generare lo schema “shark” per i database successivi al primo, commentare in “shark.sql” le righe di creazione del “ruolo utente”.

Installazione dell'applicazione

Copiare la directory “cmdb” nella sottocartella “webapps” della directory ove è installato Tomcat (eseguire i passi successivi prima di avviarlo).

Configurazione dell'applicazione

La configurazione di CMDBuild è strettamente dipendente dalla specifica versione di Tomcat utilizzata, essendo presenti significative differenze fra le recenti versioni rilasciate.

Il funzionamento di CMDBuild è stato verificato sulle seguenti versioni di Tomcat: 4.1.34, 5.0.28, 5.5.20 and 6.0.10 .

Le configurazioni da effettuare riguardano i seguenti passaggi:

a) configurazione del file di contesto per i parametri relativi all'applicazione CMDBuild

b) configurazione del file di contesto per i parametri relativi all'utilizzo del database

c) posizionamento del file di contesto

d) installazione libreria interfaccia PostgreSQL

e) eventuale installazione librerie addizionali

f) abilitazione del motore di workflow Enhydra Shark

g) avvio di Tomcat

Passaggio a)

Il presente punto va eseguito solamente se CMDBuild è stato posizionato in una cartella diversa da quella di default:

```
TomcatDIR/webapps/cmdb
```

Supponendo di averlo posizionato nella cartella:

```
TomcatDIR/webapps/cmdbuild
```

editare il file:

```
TomcatDIR/webapps/cmdbuild/META-INF/context.xml
```

Modificare il nodo <Context>, posizionato all'inizio del file XML, come mostrato nei due casi sotto riportati:

```
<Context displayName="CMDBuild"
   path="/cmdb"
   workDir="work/Catalina/localhost/cmdb">

<Context displayName="CMDBuild"
   path="/cmdbuild"
   workDir="work/Catalina/localhost/cmdbuild">
```

Passaggio b)

Il presente passaggio richiede di modificare con un normale editor il file di configurazione:

```
TomcatDIR/webapps/cmdb/META-INF/context.xml
```

per modificare, se necessario:

- il nome, il percorso e la directory di lavoro dell'istanza di CMDBuild
- l'URL del database di CMDBuild scelto in fase di installazione
- username e password necessari a CMDBuild per accedere al database

Per quanto riguarda la sezione relativa al database utilizzato da Shark, NON devono essere modificati username e password, mentre va specificato nuovamente l'URL del database (corrispondente a quello di CMDBuild, in cui risiede anche lo “schema” con le tabelle del workflow).

Per comodità dell'utilizzatore abbiamo ripetuto la descrizione delle operazioni necessarie in funzione della versione di Tomcat utilizzata.

Il file è predisposto con attiva la sezione per Tomcat 4.1 e 5.0 (la versione al momento consigliata e impostata come default) e commentata la sezione per Tomcat 5.5 e 6.0 .

Tomcat 4.1 e 5.0

Spostarsi nella sezione:

```
<!-- BEGIN REGION TOMCAT 4.1 AND 5.0 -->
<Resource name="jdbc/cmdbSrc" type="javax.sql.DataSource"
                                   auth="Container"></Resource>
<ResourceParams name="jdbc/cmdbSrc">
   <parameter>
```

```
            <name>driverClassName</name>
            <value>org.postgresql.Driver</value>
        </parameter>
        <parameter>
            <name>url</name>
            <value>jdbc:postgresql://localhost/{DATABASE NAME}</value>
        </parameter>
        <parameter>
            <name>username</name>
            <value>{POSTGRES USER}</value>
        </parameter>
        <parameter>
            <name>password</name>
            <value>{POSTGRES PASSWORD}</value>
        </parameter>
        <parameter>
            <name>maxActive</name>
            <value>300</value>
        </parameter>
        <parameter>
            <name>maxIdle</name>
            <value>2</value>
        </parameter>
        <parameter>
            <name>maxWait</name>
            <value>5000</value>
        </parameter>
        <parameter>
            <name>factory</name>
            <value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
        </parameter>
    </ResourceParams>

    <Resource name="jdbc/sharkdb" type="javax.sql.DataSource"></Resource>
    <ResourceParams name="jdbc/sharkdb">
        <parameter>
            <name>factory</name>
            <value>org.objectweb.jndi.DataSourceFactory</value>
        </parameter>
        <parameter>
            <name>maxWait</name>
            <value>5000</value>
        </parameter>
        <parameter>
            <name>maxActive</name>
            <value>300</value>
        </parameter>
        <parameter>
            <name>maxIdle</name>
            <value>2</value>
        </parameter>
        <parameter>
            <name>username</name>
            <value>shark</value>    <!-- from CMDBuild 0.60 shark has his own user,
                                    default "shark" with password "shark" -->
        </parameter>
        <parameter>
```

```
            <name>password</name>
            <value>shark</value>
         </parameter>
         <parameter>
            <name>driverClassName</name>
            <value>org.postgresql.Driver</value>
         </parameter>
         <parameter>
            <name>url</name>
            <value>jdbc:postgresql://{HOST}/{SHARK DATABASE NAME}</value>
                                    <!-- from CMDBuild 0.60 the SHARK db is
                                    the same as the CMDBuild one -->
         </parameter>
      </ResourceParams>

      <!-- END REGION TOMCAT 4.1 AND 5.0 -->
```

Modificare i parametri sopra indicati (URL, username e password), salvare il file ed uscire.

Tomcat 5.5 e 6.0

Spostarsi nella sezione:

```
<!-- BEGIN REGION TOMCAT 5.5 AND 6.0
<Resource name="jdbc/cmdbSrc" auth="Container" type="javax.sql.DataSource"
        factory="org.apache.commons.dbcp.BasicDataSourceFactory"
        maxActive="100" maxIdle="30" maxWait="10000"
        driverClassName="org.postgresql.Driver"
        username="{POSTGRES USER}"
        password="{POSTGRES PASSWORD}"
        url="jdbc:postgresql://{HOST}/{DATABASE NAME}"
        defaultAutoCommit="true" removeAbandoned="true"
        removeAbandonedTimeout="60" logAbandoned="true"/>
<Resource name="jdbc/sharkdb" auth="Container"
      type="javax.sql.DataSource"
      factory="org.objectweb.jndi.DataSourceFactory"
      maxActive="300" maxIdle="2" maxWait="5000"
      driverClassName="org.postgresql.Driver" username="shark"
      password="shark" url="jdbc:postgresql://{HOST}/{CMDB DATABASE NAME}"
      defaultAutoCommit="true" removeAbandoned="true"
      removeAbandonedTimeout="60" logAbandoned="true" />

END REGION TOMCAT 5.5 AND 6.0 -->
```

Modificare i parametri sopra indicati (URL, username e password) e rendere attiva la sezione per Tomcat 5.5 e 6.0 modificando la prima riga:

```
<!-- BEGIN REGION TOMCAT 5.5 AND 6.0
```

in:

```
<!-- BEGIN REGION TOMCAT 5.5 AND 6.0 -->
```

e l'ultima riga:

```
END REGION TOMCAT 5.5 AND 6.0 -->
```

in:

```
<!-- END REGION TOMCAT 5.5 AND 6.0 -->
```

Spostarsi sulla sezione per Tomcat 4.1 e 5.0 e commentarla modificando la prima riga:

```
<!-- BEGIN REGION TOMCAT 4.1 AND 5.0 -->
```

in:

```
<!-- BEGIN REGION TOMCAT 4.1 AND 5.0
```

e l'ultima:

```
<!-- END REGION TOMCAT 4.1 AND 5.0 -->
```

in:

```
END REGION TOMCAT 4.1 AND 5.0 -->
```

Salvare il file ed uscire.

Passaggio c)

Per comodità dell'utilizzatore abbiamo ripetuto la descrizione delle operazioni necessarie in funzione della versione di Tomcat utilizzata.

Tomcat 4.1

Spostare il file:

```
TomcatDIR/webapps/cmdb/META-INF/context.xml
```

in:

```
TomcatDIR/webapps/cmdb.xml
```

Nel caso l'applicazione CMDBuild fosse stato copiato in una directory diversa il file di contesto deve essere rinominato di conseguenza (ad esempio cmdbuild.xml se si fosse utilizzata la directory cmdbuild).

Tomcat 5.0

Spostare il file:

```
TomcatDIR/webapps/cmdb/META-INF/context.xml
```

in:

```
TomcatDIR/conf/Catalina/localhost/cmdb.xml
```

Nel caso l'applicazione CMDBuild fosse stato copiato in una directory diversa il file di contesto deve essere rinominato di conseguenza (ad esempio cmdbuild.xml se si fosse utilizzata la directory cmdbuild).

Tomcat 5.5 e 6.0

Il file:

```
TomcatDIR/webapps/cmdb/META-INF/context.xml
```

va lasciato nella posizione iniziale e con il nome iniziale.

Passaggio d)

Tomcat 4.1, 5.0 e 5.5

Copiare la libreria (nel caso si utilizzino diverse versioni di PostgreSQL è consigliabile scaricare ed utilizzare la corrispondente versione della libreria di interfaccia):

```
postgresql-8.0-313.jdbc3.jar
```

nella cartella:

```
TomcatDIR/Common/lib
```

Tomcat 6.0

Copiare la libreria (nel caso si utilizzino diverse versioni di PostgreSQL è consigliabile scaricare ed utilizzare la corrispondente versione della libreria di interfaccia):

```
postgresql-8.0-313.jdbc3.jar
```

nella cartella:

```
TomcatDIR/lib
```

Passaggio e)

Tomcat 5.5 e 6.0

Nel caso si utilizzasse Tomcat 5.5 o 6.0 è richiesta la presenza di librerie addizionali:

- commons-collections
- commons-dbcp
- commons-pool

Le librerie sopra elencate vanno copiate dalla directory:

```
tomcat55libs/*
```

nel caso di Tomcat 5.5 alla directory:

```
TomcatDIR/common/lib
```

nel caso di Tomcat 6.0 alla directory:

```
TomcatDIR/lib
```

Passaggio f)

Il sottosistema di workflow è al momento testato con Tomcat 5.0 e 5.5 e con PostgreSQL 8.1.

Per abilitare il sottosistema di workflow è necessario editare il file di configurazione "WEB-INF/SharkManager.conf".

La prima sezione contiene le righe da modificare:

```
log4j.appender.Database.File={PATH.TO.SHARK.LOG}/logs/SharkPersistence.log
log4j.appender.XMLOutFormatForPersistence.File={PATH.TO.SHARK.LOG}
      /logs/chainsaw-persistence.log
log4j.appender.PackageEvents.File={PATH.TO.SHARK.LOG}
      /logs/SharkPackageHandlingEvents.log
log4j.appender.DatabaseManager.File={PATH.TO.SHARK.LOG}/logs/dods.log
log4j.appender.XMLOutFormatForPackageEvents.File={PATH.TO.SHARK.LOG}
      /logs/chainsaw-packageevents.log
log4j.appender.SharkExecution.File={PATH.TO.SHARK.LOG}/logs/SharkExecutionFlow.log
log4j.appender.XMLOutFormatForExecution.File=/logs/chainsaw-execution.log
log4j.appender.TP.File={PATH.TO.SHARK.LOG}/logs/tp.log
log4j.appender.TP-IP.File={PATH.TO.SHARK.LOG}/logs/tp-ip.log
```

I file sopra elencati corrispondono ai log di Shark, che possono essere indirizzati nella cartella:

```
TomcatDIR/webapps/{your.cmdb.name}/workflow/
```

Va infine configurato il server di posta per l'eventuale utilizzo del metodo sendMail nell'ambito dei processi definiti:

```
DefaultMailMessageHandler.IncomingMailServer={POP.SERVER}
```

```
DefaultMailMessageHandler.IncomingMailProtocol=pop3
DefaultMailMessageHandler.StoreFolderName=INBOX
DefaultMailMessageHandler.IMAPPortNo=143
DefaultMailMessageHandler.POP3PortNo=110
DefaultMailMessageHandler.SMTPMailServer={SMTP.SERVER}
DefaultMailMessageHandler.SMTPPortNo=25
DefaultMailMessageHandler.SourceAddress={FROM.ADDRESS}
```

Passaggio g)

Al termine delle operazioni sopra descritte riavviare Tomcat.

Test di funzionamento dell'applicazione

Alla conclusione delle operazioni descritte, se tutto è stato fatto correttamente, si potrà aprire il browser e accedere all'indirizzo http://localhost:8080/cmdb/

Nel caso sia stata configurata durante l'installazione di Tomcat una porta diversa da quella standard, tale porta dovrà essere sostituita alla 8080 all'interno della URL sopra specificata.

Si presenterà la seguente schermata:

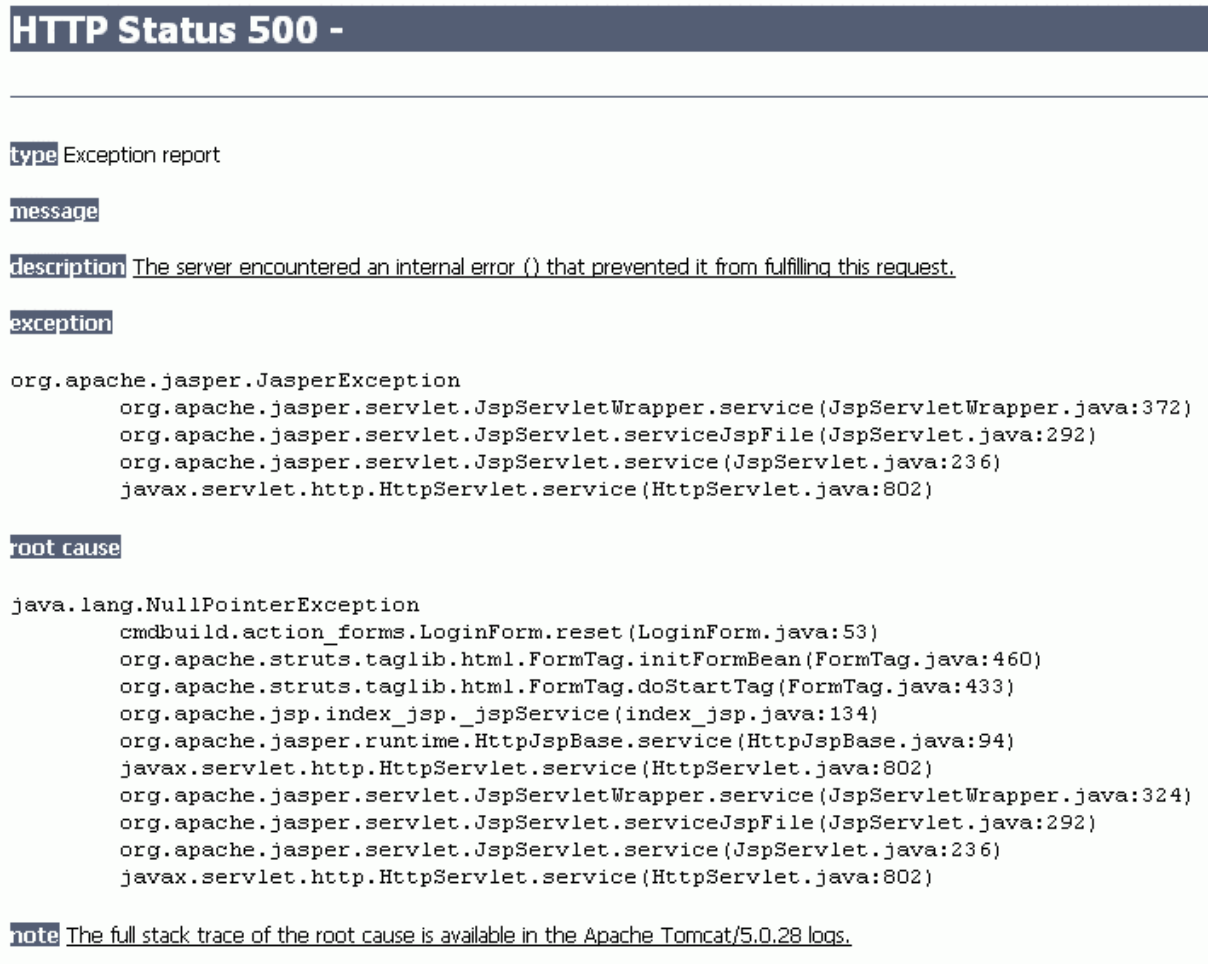
Autenticandosi con l'account:

Username: admin
Password: admin

si potrà a questo punto accedere all'applicazione CMDBuild.

I manuali disponibili sul sito (Overview Document, Administrator Manual, User Manual) forniranno il supporto necessario per un efficace utilizzo del sistema.

Nel caso si presentasse invece la seguente schermata:



può essere ipotizzato un errore di connessione con il database, CMDBuild non trova cioè il database oppure sono sbagliati username e password per connettersi a PostgreSQL.

Si consiglia in tal caso di ritornare al paragrafo relativo alla configurazione di CMDBuild e di modificare il file struts-config.xml.

Nel log di Tomcat si possono trovare ulteriori informazioni sull'accaduto, quali ad esempio:

GRAVE: Initializing application data source cmdbSource

org.apache.commons.dbcp.SQLNestedException: Cannot create PoolableConnectionFactory (Attivazione del backend fallita: FATAL: database "cmdbSource " does not exist.)

Nel caso l'errore persista verificare che l'utente di amministrazione del database PostgreSQL (di default 'Postgres') disponga dei necessari permessi di accesso, in genere specificati nel file “pg_hba.conf” escludendo ogni restrizione per l'accesso da localhost:

host all all 127.0.0.1/32 md5

Struttura dell'applicazione

Generalità

Gli strumenti individuati per la scrittura del software comprendono l'ambiente di sviluppo Eclipse e l'editor UML Omondo EclipseUML.

L'applicazione CMDBuild è stata realizzata nel linguaggio JAVA con utilizzo di pagine JSP per l'interfaccia web. L'applicazione è basata sul framework Struts che implementa il paradigma MVC (Model View Controller).

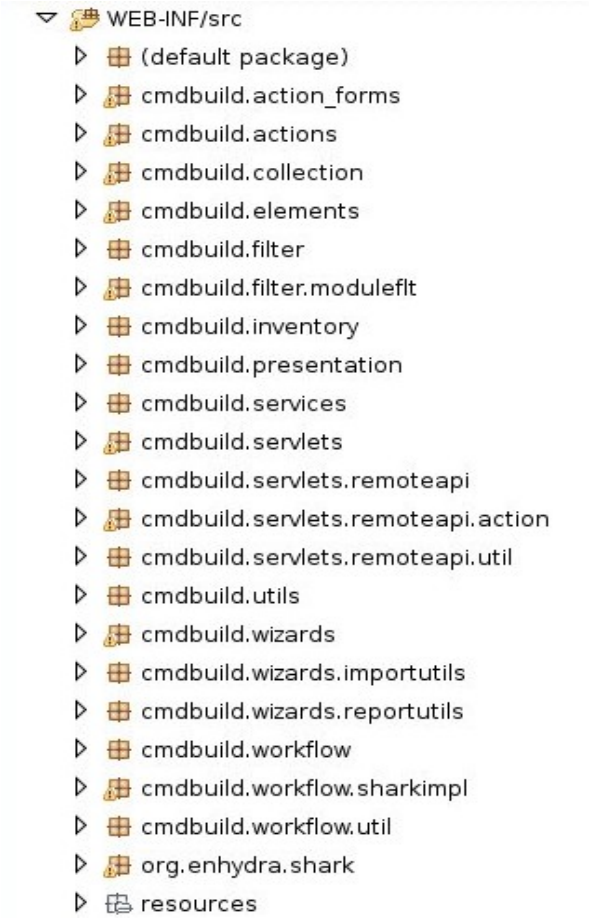
Il "modello" (responsabile tra l'altro della raccolta informazioni) è gestito da classi Java, così come il "controllore" responsabile del comportamento del sistema a seguito di un'azione dell'utente.

La "vista" è rappresentata dalle pagine JSP, che nei casi di generazione complessa del layout sfruttano alcune classi Java per la generazione di codice HTML.

Attraverso un semplice file XML di configurazione, Struts controlla CMDBuild e si occupa dell'interazione dei tre componenti sopracitati.

Albero delle classi

Le classi sono organizzate secondo la seguente struttura:



I diversi insiemi di classi hanno il seguente significato:

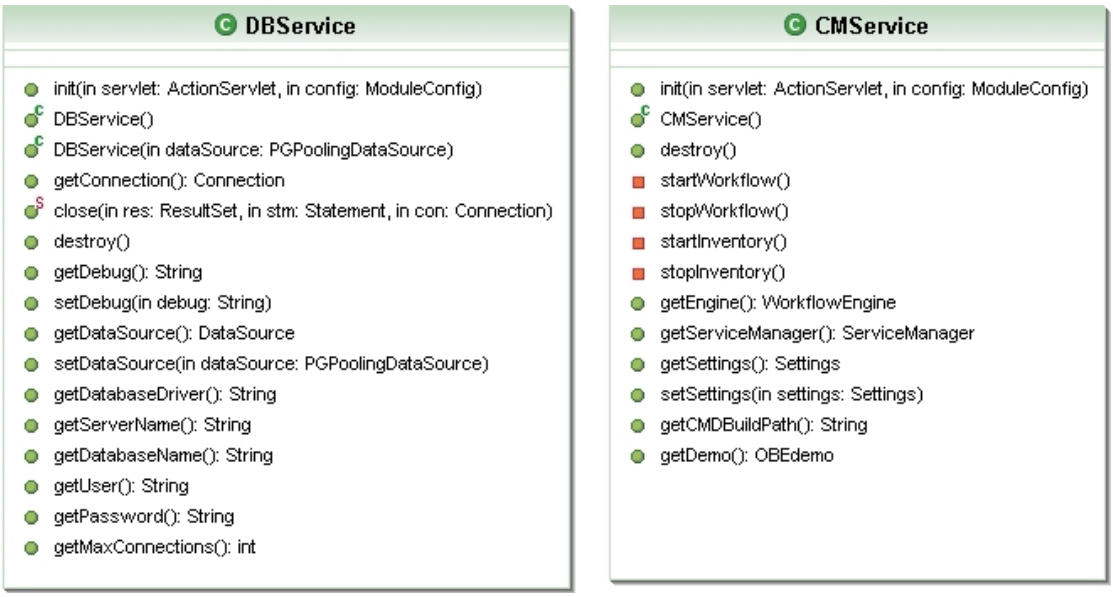
- action_forms: rappresentano il modello di Struts
- actions: rappresentano il controllore di “Struts”
- collection: insieme di classi utilizzate in CMDBuild per il trasporto delle informazioni
- elements: classi delegate all’interrogazione del database e gestori di CMDBuild.
- Filter: filtro delle richieste HTTP, configurabile
- filter.moduleft: filtri per gestione workflow e remote api
- inventory: classi per l’integrazione con OCSInventory
- presentation: classi per la generazioni di codice HTML usate dai gestori
- services: classi “plugin” per la gestione globale di CMDBuild (impostazioni, database, workflow, etc)
- servlets: classe utilizzata per il flusso dei dati in pdf
- servlets.remoteapi: contiene le chiamate remote predisposte in CMDBuild
- servlets.remoteapi.action: azioni corrispondenti alle chiamate remote
- servlets.remoteapi.util: funzioni di utilità (ad esempio traduzione oggetti in stringhe e viceversa)
- utils: generiche classi di utilità
- wizards: classi wizard
- wizard.importutils: classi utili per il wiazard di importazione
- wizard.reportutils: classi utili per il wizard dei report
- workflow: classi di definizione della struttura per la gestione del workflow
- workflow.sharkimpl: implementazione del supporto al workflow tramite Enhydra Shark
- workflow.util: classi di utilità per il workflow

Diagrammi UML

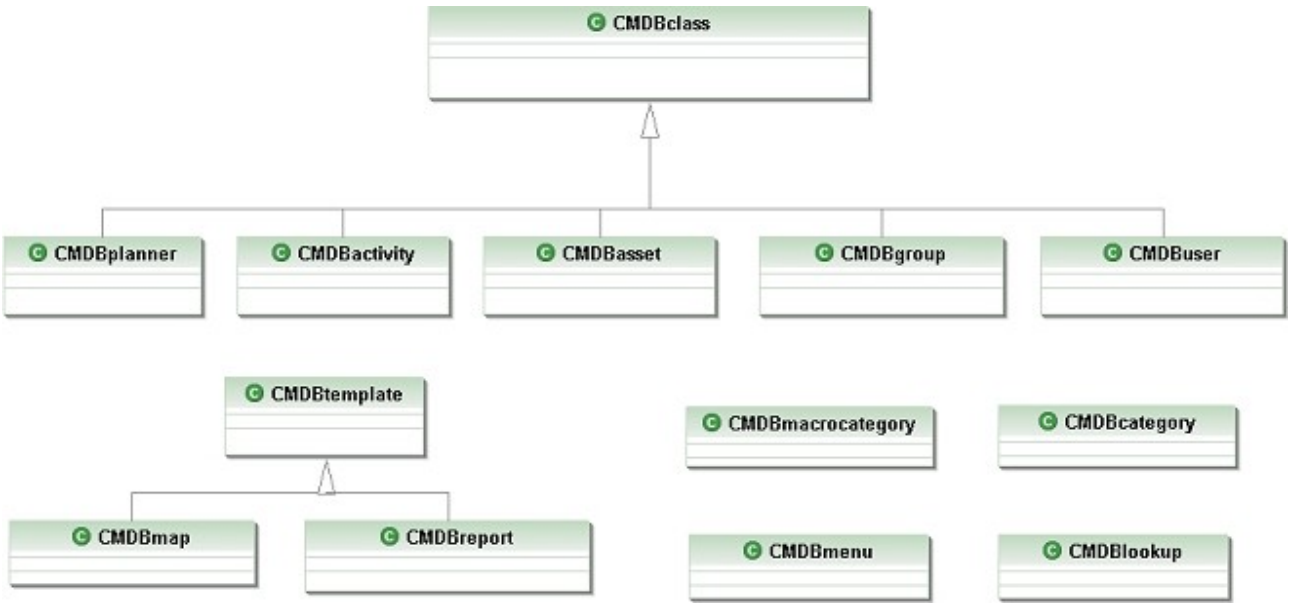
Si riportano di seguito alcuni diagrammi UML esemplificativi dell'architettura delle classi di CMDBuild.

Il seguente diagramma descrive la classe che gestisce la connessione al database e la classe per la gestione della configurazione.

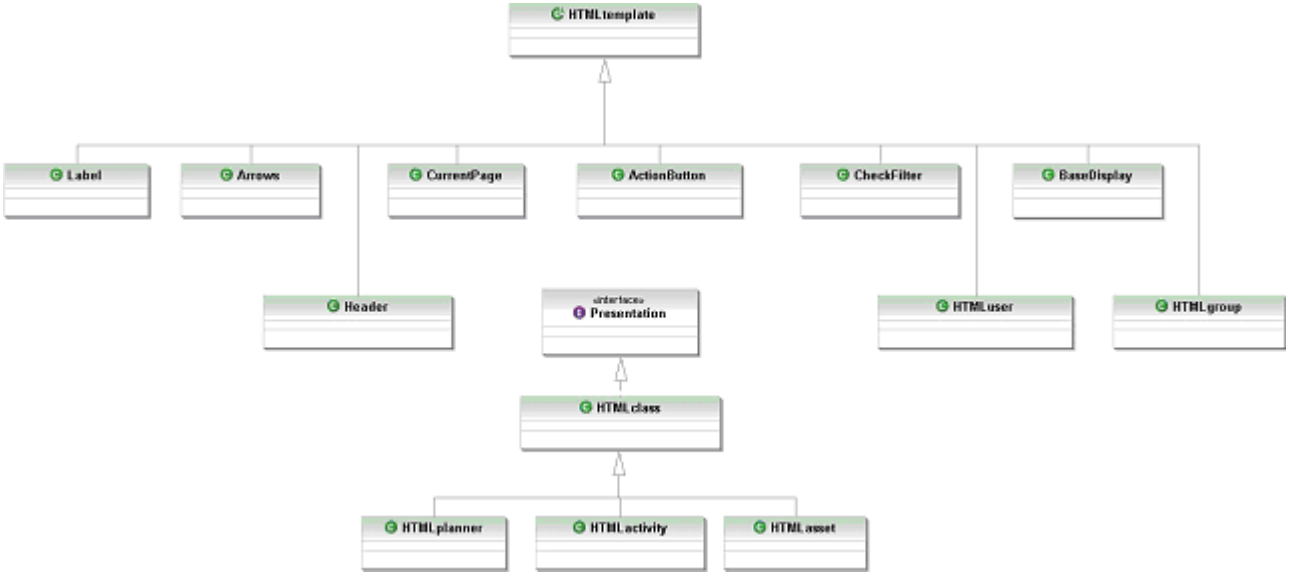
Entrambe le classi vengono caricate all’avvio dell’applicazione (avvio di Tomcat).



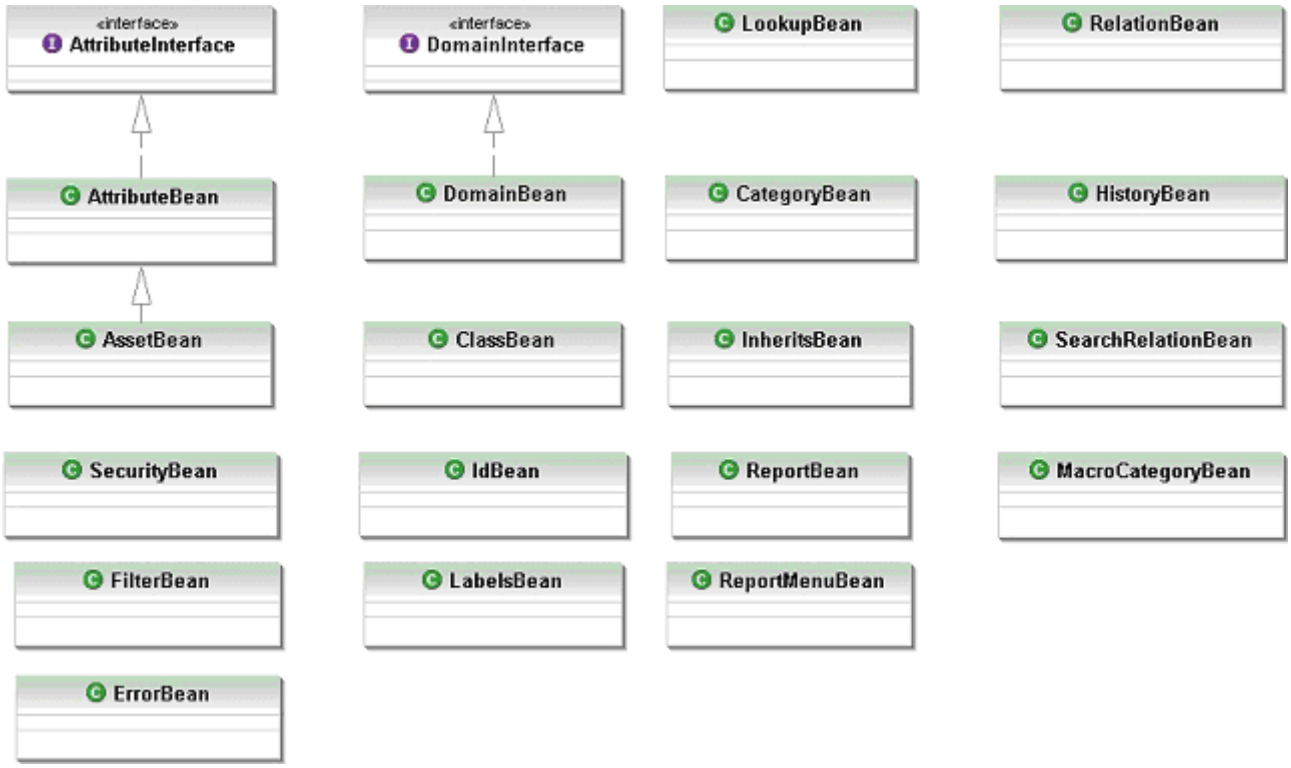
Segue il diagramma UML della classi e delle interfacce relativa alla gestione delle tabelle e dei domini nel database:



Il diagramma UML delle classi java delegate alla generazione delle schede di CMDBuild per l'editing dei dati è il seguente:



Gli oggetti sotto elencati sono invece utilizzati per il trasferimento delle informazioni all'interno di CMDBuild:



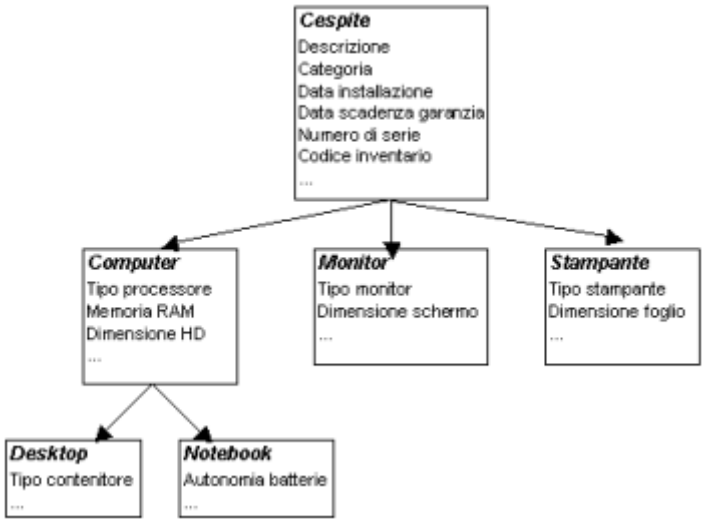
Progettazione del database

Criteri di base

PostgreSQL

La progettazione del database ha dovuto rispondere ad un primo insieme di requisiti di base:

- gestire la strutturazione gerarchica a più livelli delle classi (superclassi / sottoclassi), al fine di poterle specializzare mantenendo nelle superclassi gli attributi generali



- gestire relazioni “molti a molti” fra le classi
- tracciare la storia completa delle modifiche dei dati e delle relazioni nel tempo

Per tutte e tre le esigenze è stato individuato come particolarmente interessante il meccanismo di “derivazione” fra classi reso disponibile da alcuni database object-relational ed in particolare, in ambito open source, da PostgreSQL.

Si è quindi deciso di utilizzare PostgreSQL come database di supporto a CMDBuild, ottenendo in effetti di disegnare in modo particolarmente naturale le strutture sopra descritte.

Strutturazione a più livelli gerarchici

Tramite la parola chiave “INHERITS” è possibile infatti in PostgreSQL creare una tabella che ne “specializza” un’altra, aggiungendo alcuni attributi specifici e ritrovandosi tutti gli attributi definiti nella superclasse.

Esempio:

```
CREATE TABLE "Asset"
(
  "Id" numeric NOT NULL DEFAULT nextval('Asset_SEQ'::text),
  "Description" varchar(250),
  "SerialNo" varchar(40),
  "VersionNo" varchar(32),
  "InstallationDate" timestamp,
  "WarrantyExpireDate" timestamp,
```

```
  "State" varchar(16),
  "StateDate" timestamp,
  CONSTRAINT asset_pkey PRIMARY KEY ("Id")
)
CREATE TABLE "Monitor"
(
  "MonitorType" varchar,
  "ScreenSize" varchar(16),
  "MaxScreenRes" varchar(16)
) INHERITS ("Asset")
```

Relazioni "molti a molti"

Le diverse tipologie di relazioni fra classi vengono implementate ciascuna con una apposita tabella di relazioni "molti a molti", creata tramite derivazione da una superclasse predefinita allo scopo di semplificare la creazione delle sottoclassi e di garantirne l'omogeneità strutturale.

Esempio:

```
CREATE TABLE "Map"
(
  "Id" numeric NOT NULL DEFAULT nextval('Map_SEQ'::text),
  "Id1" numeric NOT NULL,
  "Id2" numeric NOT NULL,
  "Value" varchar(40),
  CONSTRAINT map_pkey PRIMARY KEY ("Id")
)
CREATE TABLE " Map_aggregazione"
(
) INHERITS ("Map")
```

Storia delle modifiche

Anche per la gestione della storia delle modifiche viene sfruttato il meccanismo di derivazione delle classi in PostgreSQL, creando una classe derivata per ogni tipologia di oggetto, nella quale tramite appositi trigger del database verrà archiviato il record corrente, prima di modificarne gli attributi, associandogli l'"Id" del record base, la data di modifica ed il login dell'operatore che ha effettuato la modifica.

Esempio:

```
CREATE TABLE "Map"
(
  "Id" numeric NOT NULL DEFAULT nextval('Map_SEQ'::text),
  "Id1" numeric NOT NULL,
  "Id2" numeric NOT NULL,
  "Value" varchar(40),
  CONSTRAINT map_pkey PRIMARY KEY ("Id")
)

CREATE TABLE "Monitor_history"
(
  "HistoryId" numeric NOT NULL,
  "HistoryDate" timestamp NOT NULL DEFAULT now()
) INHERITS ("Monitor")
```

Tramite lo stesso meccanismo viene gestita la storia delle modifiche nelle relazioni, creando cioè una classe derivata per ogni tipologia di relazione, nella quale tramite appositi trigger di database verranno archiviati i record di relazione correnti, prima di eliminarli, associandogli l'"Id" del record base, la data di modifica ed il login dell'operatore che ha effettuato la modifica.

Flessibilità

Il secondo insieme di requisiti cui la progettazione del database ha dovuto rispondere riguarda il supporto alla strutturazione autonoma del modello dati, ed in particolare:

- definizione di nuove classi
- definizione degli attributi delle classi
  - di tipologia base: stringa, data e ora, interi, numerici, booleani, memo
  - di tipologia complessa: lookup (basati sulla omonima tabella di decodifica), reference (analoghi alle foreign key dei database relazionali, ma legati ai domini)
- definizione di nuovi domini (tipologie di relazione) fra le classi

Per migliori garanzie di coerenza intrinseca del sistema si è scelto di archiviare le informazioni necessarie alla gestione dei meccanismi sopra descritti non all'interno di un "dizionario" esterno, ma nel "dizionario" stesso di PostgreSQL, utilizzando il campo "Commento" degli oggetti interessati tramite un apposito formalismo di codifica ed interpretazione.

Una descrizione dettagliata di tale formalismo è descritta nel seguito del presente capitolo (paragrafo "Descrittori integrativi").

Sono altresì elencate e descritte nel seguito le "function" definite nel database PostgreSQL per "incapsulare" al suo interno anche la logica applicativa di gestione delle classi e dei domini creati dinamicamente ed autonomamente tramite il Modulo Schema (paragrafi "Funzioni" e "Viste").

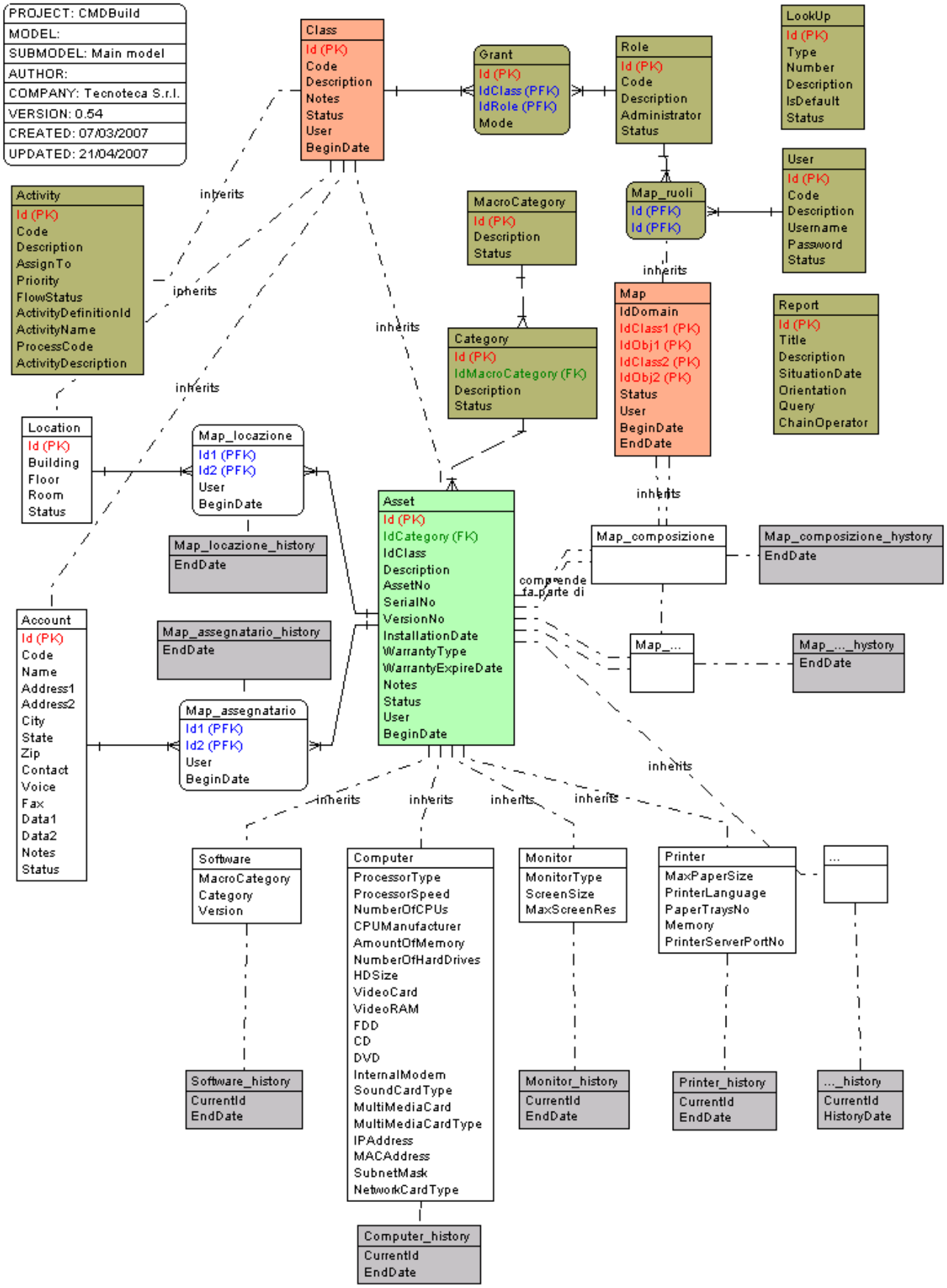
Schema risultante

Dai requisiti sopra ricordati e da ulteriori necessità di gestione individuate è risultato il seguente schema di base, contenente le classi e le strutture generali sopra descritte, riferite a titolo di esempio ad alcune classi applicative più diffuse.

I colori scelti per lo sfondo dei diversi gruppi di classi sono:

- rosso per la classe "class", da cui sono derivate tutte le classi contenenti oggetti utente
- giallo per la classe "map" e per le classi di relazione derivate
- verde per la classe "asset" (o cespite), la classe più importante per gli utenti di CMDBuild
- bianco per le altre classi utente
- marrone per le classi di sistema
- grigio per le classi dedicate alla storicizzazione dei dati e delle relazioni

Lo schema riportato è ovviamente del tutto indicativo per quanto riguarda le "classi applicative", potendo queste essere create e strutturate autonomamente da ogni utente di CMDBuild in funzione delle proprie necessità.


PROJECT: CMDBuild
MODEL:
SUBMODEL: Main model
AUTHOR:
COMPANY: Tecnoteca S.r.l.
VERSION: 0.54
CREATED: 07/03/2007
UPDATED: 21/04/2007
Class
Id (PK)
Code
Description
Notes
Status
User
BeginDate
Grant
Id (PK)
IdClass (PFK)
IdRole (PFK)
Mode
Role
Id (PK)
Code
Description
Administrator
Status
LookUp
Id (PK)
Type
Number
Description
IsDefault
Status
User
Id (PK)
Code
Description
Username
Password
Status
Map_ruoli
Id (PFK)
Id (PFK)
Activity
Id (PK)
Code
Description
AssignTo
Priority
FlowStatus
ActivityDefinitionId
ActivityName
ProcessCode
ActivityDescription
MacroCategory
Id (PK)
Description
Status
Category
Id (PK)
IdMacroCategory (FK)
Description
Status
Map
IdDomain
IdClass1 (PK)
IdObj1 (PK)
IdClass2 (PK)
IdObj2 (PK)
Status
User
BeginDate
EndDate
Report
Id (PK)
Title
Description
SituationDate
Orientation
Query
ChainOperator
Location
Id (PK)
Building
Floor
Room
Status
Map_locazione
Id1 (PFK)
Id2 (PFK)
User
BeginDate
Map_locazione_history
EndDate
Asset
Id (PK)
IdCategory (FK)
IdClass
Description
AssetNo
SerialNo
VersionNo
InstallationDate
WarrantyType
WarrantyExpireDate
Notes
Status
User
BeginDate
Map_composizione
Map_composizione_hystory
EndDate
Account
Id (PK)
Code
Name
Address1
Address2
City
State
Zip
Contact
Voice
Fax
Data1
Data2
Notes
Status
Map_assegnatario_history
EndDate
Map_assegnatario
Id1 (PFK)
Id2 (PFK)
User
BeginDate
Map_...
Map_..._hystory
EndDate
Software
MacroCategory
Category
Version
Computer
ProcessorType
ProcessorSpeed
NumberOfCPUs
CPUManufacturer
AmountOfMemory
NumberOfHardDrives
HDSize
VideoCard
VideoRAM
FDD
CD
DVD
InternalModem
SoundCardType
MultiMediaCard
MultiMediaCardType
IPAddress
MACAddress
SubnetMask
NetworkCardType
Monitor
MonitorType
ScreenSize
MaxScreenRes
Printer
MaxPaperSize
PrinterLanguage
PaperTraysNo
Memory
PrinterServerPortNo
...
Software_history
CurrentId
EndDate
Monitor_history
CurrentId
EndDate
Printer_history
CurrentId
EndDate
..._history
CurrentId
HistoryDate
Computer_history
CurrentId
EndDate
inherits
comprende
fa parte di

Descrittori integrativi

CMDBuild utilizza per la gestione dei propri meccanismi di gestione informazioni relative a classi ed attributi aggiuntive rispetto a quelle trattate dal dababase PostgreSQL.

Tali informazioni vengono archiviate nel campo “Comment” rispettivamente di classi ed attributi, con la seguente sintassi:

Tipo oggetto	Tipo Elemento	Formato Tag
Classe (tabella)	Modo di utilizzo, per distinguere classi di servizio, classi non modificabili e classi utente	MODE: reserved | read | write *reserved = non visualizzato* *read = non modificabile* *write = modificabile*
	Tipo, per distinguere classi e domini (nella fattispecie classe)	TYPE: class | domain *class = classe* *domain = dominio*
	Descrizione della classe, normalmente una stringa più esplicativa del nome	DESCR: *valore (stringa)*
	Flag superclasse	SUPERCLASS: true | false *true = superclasse* *false = non superclasse*
	Gestore della classe: gestore base (“class”) o nome del metodo custom java implementato	MANAGER: *gestore (stringa)*
	Stato della classe, utilizzato per registrare la cancellazione logica	STATUS: active | noactive *active = attivo* *noactive = cancellato*
Attributo (colonna)	Modo di utilizzo, per distinguere attributi di servizio (ad esempio Id, data validità, ecc), attributi non modificabili (derivati da superclassi) e attributi utente	MODE: reserved | read | write *reserved = non visualizzato* *read = non modificabile* *write = modificabile*
	Descrizione dell'attributo, normalmente una stringa più esplicativa del nome	DESCR: *valore (stringa)*
	Indice dell'attributo (posizione nella lista degli attributi)	INDEX: *valore (numero intero)*
	Riferimento alla lista delle voci di decodifica per attributi di tipo LookUp	LOOKUP: *valore (LookUp.Type)*
	Riferimenti alle modalità di gestione dei campi “Reference”, con indicazioni del dominio associato, della modalità di cancellazione (non ammessa, a cascata), del fatto che la classe corrente sia la prima o la seconda della coppia in relazione	REFERENCEDOM: *valore (stringa)* REFERENCETYPE: *restrict = non ammessa* *cascade = a cascata (non ancora implementata)* REFERENCEDIRECT:true | false *true = prima classe nella coppia* *false = seconda classe nella coppia*
	Flag campo data da trattare come “scadenza”	DATEEXPIRE: true | false

		true = campo scadenza *false = non campo scadenza*
	Flag campo base per visualizzazione (ad esempio sarà visualizzato anche nelle pagine “a tabulato” contenenti un numero ridotto di colonne)	BASEDSP: true | false *true = base per visualizzazione* *false = non base per visualizzazione*
	Stato della classe, utilizzato per registrare la cancellazione logica	STATUS: active | noactive *active = attivo* *noactive = cancellato*
Dominio (tabella)	Modo di utilizzo, per distinguere domini di servizio e domini utente	MODE: reserved | read | write *reserved = non visualizzato* *read = non modificabile* *write = modificabile*
	Tipo, per distinguere classi e domini (nella fattispecie dominio)	TYPE: class | domain *class = classe* *domain = dominio*
	Prima classe della coppia in relazione	CLASS1: *valore (stringa)*
	Seconda classe della coppia in relazione	CLASS2: *valore (stringa)*
	Descrizione diretta	DESCRDIR: *valore (stringa)*
	Descrizione inversa	DESCRINV: *valore (stringa)*
	Flag dominio da gestire rappresentando le due classi in relazione in forma master-detail	MASTERDETAIL: true | false *true = gestione master-detail* *false = non gestione master-detail*
	Cardinalità	CARDIN: 1:N | N:1 | N:M
	Stato del dominio, utilizzato per registrare la cancellazione logica	STATUS: active | noactive *active = attivo* *noactive = cancellato*

Funzioni

Si è scelto di utilizzare in modo esteso le “Function” di PostgreSQL per “incapsulare” al suo interno anche la logica applicativa di gestione delle classi e dei domini.

Segue la lista delle principali funzioni utilizzate:

NOME	UTILIZZO	PARAMETRI
createattribute	Creazione dinamica nuovo attributo	*ClassName = nome classe* *AttributeName = nome attributo* *AttributeIndex = indice attributo (posizione)*

		AttributeDesc = descrizione attributo *AttributeType = tipo attributo (boolean, date, decimal, double, integer, reference, lookup, string, text, timestamp)* *AttributeLookUp = lista LookUp collegata* *AttributeReference =* *AttributeReferenceDomain = dominio per tipo reference* *AttributeReferenceType = tipo gestione cancellazione per tipo reference* *AttributeReferenceDirect = prima o seconda classe nella coppia* *AttributeDateExpire = flag data tipo scadenza* *AttributeMandatory = obbligatorio* *AttributeDefault = valore default* *AttributeBaseDSP = base per visualizzazione* *AttributeStatus = attributo attivo o eliminato*
createclass	Creazione dinamica nuova classe	*ClassName = nome classe* *ClassDesc = descrizione classe* *ClassStatus = classe attiva o eliminata* *ParentClass = classe da cui eredita* *SuperClass = flag superclasse* *ClassManager = gestore classe*
createdomain	Creazione dinamica nuovo dominio	*DomainName = nome dominio* *DomainClass1 = nome classe1* *DomainClass2 = nome classe2* *DomainDescrDir = descrizione diretta* *DomainDescrInv = descrizione inversa* *DomainMasterDetail = flag master-detail* *DomainCardin = cardinalità* *DomainStatus = dominio attivo o eliminato*
createhistory	Creazione tabella derivata per storicizzazione classe (trigger esclusi)	*ClassName = nome classe*
createrelationhistory	Creazione tabella derivata per storicizzazione dominio (trigger esclusi)	*DomainName = nome dominio*
createrelationtriggers	Creazione triggers per la storicizzazione del dominio	*DomainName = nome dominio*
createtriggers	Creazione triggers per la storicizzazione della classe	*ClassName = nome classe*
deleteattribute	Cancellazione attributo (da utilizzarsi solamente a tabella vuota)	*ClassName = nome classe* *AttributeName = nome attributo*
deleteclass	Cancellazione classe (da utilizzarsi solamente a tabella vuota)	*ClassName = nome classe*

deletedomain	Cancellazione dominio (da utilizzarsi solamente a tabella vuota)	*DomainName = nome dominio*
deleterelationtriggers	Cancellazione triggers per la storicizzazione del dominio	*DomainName = nome dominio*
deletetriggers	Cancellazione triggers per la storicizzazione della classe	*ClassName = nome classe*
deleteview	Cancellazione vista	*ViewName = nome vista*
modifyattribute	Modifica attributo	*ClassName = nome classe* *AttributeName = vecchio nome attributo* *AttributeNewName = nuovo nome attributo* *AttributeIndex = indice attributo (posizione)* *AttributeDesc = descrizione attributo* *AttributeType = tipo attributo (boolean, date, decimal, double, integer, reference, lookup, string, text, timestamp)* *AttributeLookUp = lista LookUp collegata* *AttributeReference =* *AttributeReferenceDomain = dominio per tipo reference* *AttributeReferenceType = tipo gestione cancellazione per tipo reference* *AttributeReferenceDirect = prima o seconda classe nella coppia* *AttributeDateExpire = flag data tipo scadenza* *AttributeMandatory = obbligatorio* *AttributeDefault = valore default* *AttributeBaseDSP = base per visualizzazione* *AttributeStatus = attributo attivo o eliminato*
modifyclass	Modifica classe	*ClassName = nome classe* *ClassNewName = nuovo nome classe* *ClassDesc = descrizione classe* *ClassStatus = classe attiva o eliminata* *SuperClass = flag superclasse* *ClassManager = gestore classe*
modifydomain	Modifica dominio	*DomainName = nome dominio* *DomainNewName = nuovo nome dominio* *DomainClass1 = nome classe1* *DomainClass2 = nome classe2* *DomainDescrDir = descrizione diretta* *DomainDescrInv = descrizione inversa* *DomainMasterDetail = flag master-detail* *DomainCardin = cardinalità* *DomainStatus = dominio attivo o eliminato*

Viste

Per un più immediato accesso al “dizionario dati allargato”, costituito dal Data Dictionary di PostgreSQL e dai descrittori integrativi archiviati nei campi “Comment”, sono state definite ed utilizzate le viste descritte alla tabella successiva.

Nome	Utilizzo
cmdbclasscatalog	Catalogo completo delle classi con gli attributi gestiti in CMDBuild
cmdbdomaincatalog	Catalogo completo dei domini con gli attributi gestiti in CMDBuild
cmdbgroupcatalog	Lista gruppi utenti definiti in CMDBuild
cmdbpolicycatalog	Lista permessi definiti in CMDBuild per i singoli utenti
navigationtree	Lista gerarchica delle classi utente definite in CMDBuild (utilizzato come albero di navigazione nel programma)
relationlist	Lista relazioni archiviate in CMDBuild
relationhistorylist	Lista relazioni archiviate in CMDBuild inclusi record cancellati e storicizzati

API di CMDBuild

Generalità

[da completare]

Lista delle API

[da completare]

Attuali limitazioni note del sistema

Export / Import

Si è riscontrato che in PostgeSQL il comando "pg_dump", anche se lanciato con opzione "-o" salva (e consente a "pg_restore" di ripristinare) gli "oid" delle righe delle tabelle, ma non salva (e quindi non consente il ripristino) gli "oid" delle classi all'interno del catalogo di sistema.

Essendo tali "oid" utilizzati da CMDBuild come propri riferimenti interni (classe “Class” e derivate, classe “Map” e derivate), a seguito del restore di un database risultano non corrette le relazioni stabilite fra gli oggetti di CMDBuild.

E' stato realizzato e reso disponibile un comando di ripristino che va eseguito successivamente ad ogni "restore" e che aggiorna i riferimenti in questione a partire da informazioni che sono comunque disponibili nel sistema (sottolineiamo a questo proposito che non esiste alcun rischio di perdita di dati).

Campi tipo “reference”

La struttura di CMDBuild consente la definizione di campi di tipo “reference” fra gli attributi di una superclasse ed il loro utilizzo nell'ambito delle classi da questa derivate.

Non è al momento gestito il caso di campi “reference” ereditati lungo catene “superclassi – sottoclassi” di lunghezza superiore a due (non è quindi ad esempio possibile definire un campo “reference” fra la superclasse “asset” e la classe “fattura”ed utilizzarlo nella sottoclasse “notebook” se esiste una ulteriore superclasse “computer” fra “asset” e “notebook”).

Non è al momento possibile definire campi reference fra una classe normale ed una superclasse.

Essendo i campi reference gestiti tramite le foreign key di PostgreSQL, risentono di alcune limitazioni tuttora non risolte (vedi articolo successivo e in particolare il punto 5.8.1):

http://www.postgresql.org/docs/8.2/static/ddl-inherit.html

Data la potenziale criticità di tale limitazione sarà prossimamente rilasciata una patch che consenta la definizione di campi reference fra una classe normale ed una superclasse, rinunciando alla definizione della “foreign key” nel database e rinunciando quindi provvisoriamente ai corrispondenti controlli automatici di coerenza intrinseci nel database.

Sempre relativamente ai campi di tipo “reference” non è al momento gestita l'opzione di cancellazione in modalità “cascade”, cioè con eliminazione automatica di tutti gli elementi collegati anche su più livelli al record da cancellare (ad esempio non si può cancellare la scheda di un “fornitore” se esiste una scheda “contatti” con un campo di tipo “reference” sulla classe “fornitori” ed esistono dei contatti ancora attivi per quel fornitore).

E' quindi al momento necessario provvedere manualmente alla cancellazione degli elementi collegati prima di poter cancellare la scheda principale.

Tale limitazione è legata al fatto che le uniche cancellazioni ammesse in CMDBuild sono cancellazioni di tipo “logico”, il che non consente di usufruire delle opzioni di cancellazioni previste in PostgreSQL per le “foreign key”.