

# Kandau: Persistència d'objectes per a aplicacions KDE

Albert Cervera i Areny (albert@wickywicky.net)

Juny 2006

## Resum

La recent popularització de sistemes com Castor o Hibernate, ha fet que la persistència d'objectes sigui un element habitual en desenvolupaments sobre la plataforma Java i es comenci a valorar l'estalvi de temps que suposa una eina d'aquestes característiques. Malauradament però, no és senzill trobar un sistema de persistència d'objectes per C++ decent, principalment per la inexistència de mecanismes d'introspecció en el propi llenguatge. Kandau, aprofita i amplia les facilitats d'introspecció que aporta la llibreria Qt per tal d'oferir d'una forma extremadament simple, persistència d'objectes per a aplicacions escrites en C++.

## 1 Introducció

La persistència d'objectes és avui en dia, un element més de la pila de tecnologies utilitzades en el desenvolupament en Java. Castor [Cas06] i Hibernate [Hib06], són possiblement els exponents més coneguts. També hi ha implementacions de tecnologies semblants en altres llenguatges i plataformes, com ZODB [ZOD06] per Python o NHibernate per .NET [NH06].

Malgrat això, no ha estat fins fa poc que aquestes tecnologies s'han imposat, i malgrat els seus grans avantatges, encara hi ha molts llenguatges i desenvolupadors que les ignoren. Tal és el cas de les aplicacions escrites en PHP, C o C++. Sovint, els desenvolupadors lliguen el seu projecte a un sistema gestor de bases de dades i opten per implementar sobre la marxa la correspondència objecte-relacional.

Un altre cas freqüent són aplicacions que creixen més del previst o han de començar a permetre accés a múltiples usuaris a les dades. L'aplicació ha anat desant les dades en un fitxer propi, mitjançant funcions per la serialització de cada una de les classes i de cop, apareix la necessitat de no lligar-se a un format concret. Sorgeixen nous estàndards, els administradors comencen a demanar poder desar les dades en servidors, etc.

L'entorn d'escriptori KDE, proporciona una plataforma extraordinària pel desenvolupament d'aplicacions gràfiques per sistemes operatius Unix i

ben aviat per Windows. Malgrat això, múltiples aplicacions han hagut de desenvolupar la seva pròpia interfície sobre la qual desenvolupar nous *backends* per a la persistència de les dades.

## 2 Què ofereix Kandau?

Tot tipus d'aplicacions es poden veure beneficiades per l'ús de la llibreria Kandau [CA06]. La poden utilitzar per desar la configuració de l'aplicació en un fitxer o base de dades, però també poden utilitzar-lo per emmagatzemar majors volums d'informació, com la gestió d'una base de dades de contactes o dades de facturació d'una empresa. A més a més, Kandau ofereix altres solucions que poden ser d'utilitat a aplicacions que cerquin, tan sols, eines que permetin un ús intensiu de la introspecció de codi.

En les següents seccions exposarem els tres puntals que formen Kandau.

### 2.1 Introspecció

La plataforma de desenvolupament Qt [Tro06], permet una certa introspecció als desenvolupadors d'aplicacions en C++, llenguatge, que d'altre banda, no té suport per a aquests mecanismes. Essencialment, Qt ofereix la possibilitat de declarar propietats a les classes i iterar-hi a través d'un objecte instanciat que hereti de `QObject`.

Kandau, estén les possibilitats d'introspecció de les Qt permetén:

- Iterar a través de les classes declarades. És a dir, sense necessitat de crear cap objecte, poder saber quines classes hi ha disponibles.
- Iterar a través de les propietats d'una classe, sense necessitat de crear-ne una instància.
- Establir relacions de tipus 1-1, 1-n, n-1 i n-n entre classes i iterar a través d'elles.
- Encastar meta-informació complexa a les declaracions de classes.

Això permet al programador (i al Kandau) poder dibuixar un esquema del model de dades i crear de forma automàtica, per exemple, les taules adients per a una base de dades SQL. A més de permetre crear interfícies gràfiques de forma totalment automatitzada.

La introspecció oferta, per tant, pot ser d'utilitat, no tan sols a aquells que desitgin l'emmagatzemament de la informació, sinó a qualsevol aplicació que vulgui obtenir major informació sobre les classes sense la necessitat d'un precompilador.

El següent fragment de codi en C++ mostra com es poden llistar totes les classes declarades amb les relacions que s'estableixen entre elles i les propietats de cada una:

```

ClassInfoIterator cit( Classes::begin() );
ClassInfoIterator cend( Classes::end() );
ClassInfo *current;
for ( ; cit != cend; ++cit ) {
    current = (*cit);

    kdDebug() << "Class: " << current->name() << endl;
    kdDebug() << "    Properties:" << endl;
    PropertiesInfoConstIterator pit( current->propertiesBegin() );
    PropertiesInfoConstIterator pend( current->propertiesEnd() );
    PropertyInfo *pro;
    for ( ; pit != pend; ++pit ) {
        pro = *pit;
        kdDebug() << "        Name: " << pro->name() << "    Type: "
            << QVariant::typeName( pro->type() ) << endl;
    }

    kdDebug() << "    Objects:" << endl;
    RelationInfosIterator oit( current->relationsBegin() );
    RelationInfosIterator oend( current->relationsEnd() );
    RelationInfo *obj;
    for ( ; oit != oend; ++oit ) {
        obj = *oit;
        QString n;
        n = obj->isOneToOne() ? "1-1" : "N-1";
        kdDebug() << "        Name: " << obj->name() << "    Class: "
            << obj->relatedClassInfo()->name() << "    Relation: "
            << n << endl;
    }

    kdDebug() << "    Collections:" << endl;
    CollectionInfosIterator lit( current->collectionsBegin() );
    CollectionInfosIterator lend( current->collectionsEnd() );
    CollectionInfo *col;
    for ( ; lit != lend; ++lit ) {
        col = *lit;

        QString n;
        n = col->isNToOne() ? "N-1" : "N-N";
        kdDebug() << "        Name: " << col->name() << "    Class: "
            << col->childrenClassInfo()->name() << "    Relation: "
            << n << endl;
    }
}
}

```

## 2.2 Persistència

La persistència de la informació és el punt més fort del Kandau, i és que un programador acostumat a la creació de classes a l'estil Qt, només ha d'afegir un parell de macros per tal d'obtenir la persistència de les propietats dels seus objectes. A més, Kandau permet al programador establir relacions entre els diferents objectes, i emmagatzemar també aquestes relacions en fer el *commit()*.

Tal com es veu en la declaració de la classe, el que cal fer és: heredar de Object i cridar la macro DCLASS(nom\_de\_la\_classe). No cal mantenir cap variable per la informació de relacions entre objectes.

```
class Author : public Object
{
    Q_OBJECT
    Q_PROPERTY( QString firstName READ firstName WRITE setFirstName );
    Q_PROPERTY( QString lastName READ lastName WRITE setLastName );
    Q_PROPERTY( QString fullName READ fullName );
    Q_PROPERTY( QString biography READ biography WRITE setBiography );
    Q_PROPERTY( uint birthYear READ birthYear WRITE setBirthYear );
public:
    DCLASS( Author );

    void setFirstName( const QString& name );
    const QString& firstName() const;

    void setLastName( const QString& name );
    const QString& lastName() const;

    QString fullName() const;

    void setBiography( const QString& biography );
    const QString& biography() const;

    void setBirthYear( uint year );
    uint birthYear() const;

    Collection* bibliography();

private:
    QString m_firstName;
    QString m_lastName;
    QString m_biography;
    uint m_birthYear;
};
```

En la implementació cal: fer la crida a la macro `ICLASS(nom_de_la_classe)`, establir les relacions a la funció `createRelations()`, implementar les funcions que porten als objectes relacionats mitjançant la macro `GETOBJECT` o `GETCOLLECTION` segons el tipus de relació i afegir la macro `MODIFIED` en les funcions que permeten establir una propietat de l'objecte, per tal que el *Manager* pugui saber si l'objecte s'ha modificat.

```
#include "author.h"

ICLASS(Author);

void Author::createRelations()
{
    COLLECTION( Book );
}

void Author::setFirstName( const QString & name )
{
    MODIFIED;
    m_firstName = name;
}

const QString & Author::firstName( ) const
{
    return m_firstName;
}

void Author::setLastName( const QString & name )
{
    MODIFIED;
    m_lastName = name;
}

const QString & Author::lastName( ) const
{
    return m_lastName;
}

QString Author::fullName( ) const
{
    QString sfirst = m_firstName.stripWhiteSpace();
    QString slast = m_lastName.stripWhiteSpace();
    if ( sfirst != "" && slast != "" )
        return slast + ", " + sfirst;
}
```

```

        else
            return sfirst + slast;
    }

void Author::setBiography( const QString & biography )
{
    MODIFIED;
    m_biography = biography;
}

const QString & Author::biography( ) const
{
    return m_biography;
}

void Author::setBirthYear( uint year )
{
    MODIFIED;
    m_birthYear = year;
}

uint Author::birthYear( ) const
{
    return m_birthYear;
}

Collection* Author::bibliography()
{
    return GETCOLLECTION( Book );
}

```

En el fragment de codi següent, és pot veure com és de senzill desar un parell d'objectes, a una base de dades PostgreSQL, creant-ne abans les taules necessàries.

```

Classes::setup();

QSqlDatabase *db = QSqlDatabase::addDatabase( "QPSQL7" );
db->setDatabaseName( "bookshelf" );
db->setUserName( "user" );
db->setPassword( "password" );
db->setHostName( "localhost" );
if ( ! db->open() ) {
    kdDebug() << "Failed to open database: " << db->lastError().text()

```

```

        << endl;
    return 1;
}
DbBackendIface *backend = new SqlDbBackend( db );

Manager *manager = new Manager( backend );

// Create necessary tables
manager->createSchema();

ObjectRef<Author> monzo = Author::create();
monzo->setFirstName( "Quim" );
monzo->setLastName( "Monzó" );
monzo->setBirthYear( 1952 );
monzo->setBiography( "Born in Barcelona in 1952, has worked as a graphic"
    " designer, translator, journalist and has contributed to radio"
    " and TV programmes." );

ObjectRef<Book> benzina = Book::create();
benzina->setTitle( "Benzina" );
benzina->setIsbn( "84-7727-434-7" );
benzina->setYear( 1983 );
monzo->bibliography()->add( benzina );

kdDebug() << "The author of the book 'Benzina' is "
    << benzina->author()->fullName() << endl;
// Output: "The author of the book 'Benzina' is Monzó, Quim"

manager->commit();

```

Cal dir, que si es volgués emmagatzemar la informació en un fitxer XML, tan sols caldria treure el codi de connexió a la base de dades i utilitzar aquest altre motor (o un que programéssim nosaltres mateixos):

```
DbBackendIface *backend = new XmlDbBackend( "bookshelf.xml" );
```

Una altra qüestió a fer notar sobre el codi anterior, és que tan bon punt s'ha afegit el llibre a la bibliografia de l'autor, la relació llibre-autor queda establerta també per determinar qui és l'autor del llibre. Això es produeix de forma automàtica, ja que el *Manager*, sap que la relació n-1 que hi ha entre el llibre i l'autor, és la mateixa 1-n que hi ha entre l'autor i els seus llibres.

### 3 Desenvolupament RAD (Desenvolupament ràpid d'aplicacions)

Kandau, a més dels mecanismes de persistència, pretén simplificar el desenvolupament d'aplicacions senzilles i monòtones que poden fer ús de la introspecció per generar diàlegs d'interacció amb l'usuari. De fet, mitjançant els *widgets* que incorpora la llibreria, es pot navegar i actualitzar les dades de les classes que hom crea sense escriure ni una sola línia de codi per a la generació dels diàlegs.

La idea, és crear un conjunt d'elements gràfics que permetin desenvolupar de forma ràpida aplicacions basant-se en les classes i en les seves relacions, en lloc de fer-ho a partir de taules en una base de dades, com és típic d'Access o Kexi. El desenvolupament de l'aplicació a partir de taules, no és tant natural com fer-ho a partir de classes pel desenvolupador de C++.

### 4 Conclusions

Kandau és una llibreria per a la persistència d'objectes en C++ que intenta mantenir la facilitat d'ús i ofereix eines pel desenvolupament ràpid d'interfícies gràfiques d'usuari. Aquesta facilitat d'ús però, no treu funcionalitats i permet emmagatzemar, de forma trivial, les dades a una base de dades PostgreSQL o en un fitxer XML, només variant una línia de codi. Fins i tot, l'estructura de la base de dades es pot crear automàticament.

Aquestes possibilitats, unides a la gran potència de les llibreries KDE fan que el desenvolupament d'aplicacions en C++, sigui cada dia més senzill i ràpid i permeten al programador centrar-se en els problemes propis de la seva aplicació, sense necessitat d'escriure una vegada i una altra codi molt semblant projecte darrere projecte.

### Referències

- [CA06] Albert Cervera Areny. *Pàgina principal del projecte Kandau* (<http://kandau.berlios.de>), 2006.
- [Cas06] Castor. *Pàgina principal del projecte Castor* (<http://www.castor.org>), 2006.
- [Hib06] Hibernate. *Pàgina principal del projecte Hibernate* (<http://www.hibernate.org>), 2006.
- [KDE06] KDE. *Pàgina principal del projecte KDE* (<http://www.kde.org>), 2006.
- [NH06] NHibernate. *Pàgina principal del projecte NHibernate* (<http://www.hibernate.org/343.html>), 2006.

- [Tro06] Trolltech. *Qt Reference Documentation* (<http://doc.trolltech.com/>), 2006.
- [ZOD06] ZODB. *Pàgina principal del projecte ZODB* (<http://www.zope.org/Wikis/ZODB/FrontPage>), 2006.
- [ZR04] Richard Moore Zack Rusin, Till Adam. *Common Programming Mistakes* (<http://developer.kde.org/documentation/other/mistakes.html>), 2004.