ORI: Una Interfaccia Object-Oriented per Database Relazionali

Sonia Bergamaschi (*), Alessandra Garuti (*), Daniele Montanari (%), Claudio Sartori (#), Alberto Venuta (*)
(*) DSI - Università di Modena {sonia,garuti,venuta@dsi.unimo.it}
(%) TEMA S.p.a {montanari@tema.it}
(#) DEIS - Università di Bologna {csartori@deis.unibo.it}

Le tecnologie orientate agli oggetti sono tra le più utilizzate per lo sviluppo del software in quanto forniscono un modo pratico e produttivo di sviluppare applicazioni. Le applicazioni orientate agli oggetti supportano un ampio numero di istanze di oggetti, un'esigenza che ne deriva é quella di rendere persistenti parte di questi oggetti, ossia di mantenere una copia di essi in memoria secondaria. L'approccio maggiormente utilizzato per la gestione della persistenza dei dati sono tuttora i DBMS relazionali, sia per le numerose funzionalità che supportano (gestione di transazioni, integrità, condivisione tra più utenti e applicazioni, linguaggi standard di interrogazione (SQL)) che per la grande mole di dati preesistenti in formato relazionale, con cui spesso le nuove applicazioni devono interagire.

Nell'ambito di un progetto congiunto tra DEIS e TEMA S.p.a. (gruppo ENI) si é quindi deciso di progettare e sviluppare un modulo software, denominato *ORI*, che consenta di memorizzare e recuperare oggetti memorizzati in modo persistente su un sottostante RDBMS, utilizzando però un'interfaccia object-oriented. Mantenere queste funzionalità in un componente separato permette di isolare le dipendenze che derivano dal sistema di gestione dati utilizzato e contribuisce alla portabilità di un`applicazione. Lo scopo del modulo *ORI* é di rendere disponibile ai programmatori di applicazioni object-oriented un`interfaccia object-oriented ad un RDBMS: in tal modo, il programmatore non deve conoscere la struttura del sottostante database. Il modulo é stato sviluppato in ambiente di programmazione Visual C++ e accede a RDBMS via ODBC. *ORI* non é un modulo indipendente ma é un strato di sofware che può essere aggiunto ad una qualsiasi applicazione object-oriented che deve usarne le sue funzionalità.

Di seguito vengono riportate le premesse che stanno alla base di *ORI*:

1. Ogni oggetto persistente ha due stati: uno nella memoria volatile e uno memorizzato nel database. Lo stato volatile é definito dalla relativa classe, mentre lo stato nel database é descritto da una o più tabelle. La corrispondenza tra i due stati viene stabilita tramite i valori della chiave primaria
2. Lo stato volatile può sopravvivere alla fine di un transazione sul database. Quando la transazione termina correttamente lo stato nella memoria volatile e quello nel database sono consistenti e una nuova transazione può iniziare
3. Una interrogazione ritorna solo e sempre oggetti. Gli oggetti recuperati vengono istanziati
4. Quando un oggetto persistente contenente dei riferimenti ad altri oggetti persistenti viene recuperato, solo il primo livello della gerarchia di aggregazione viene istanziato. Gli oggetti recuperati contengono i valori di attributi semplici ma non i puntatori agli oggetti associati. Un'ulteriore operazione deve essere eseguita per recuperare gli oggetti richiesti.
5. Le procedure di inserimento ed aggiornamento di un oggetto non sono limitate all'oggetto stesso ma vengono propagate nella gerarchia di aggregazione agli oggetti correlati precedentemente instanziati.
6. La gestione di modifiche concorrenti degli oggetti viene risolta mediante un protocollo ottimistico: non si attua nessun tipo di lock sugli oggetti, in caso di conflitto la transazione viene abortita.

In Figura 1 viene descritta l'architettura funzionale di *ORI*.

1. Il punto di partenza é **l'Object Model** dell'applicazione che descrive il dominio dell'applicazione: la struttura delle classi, i metodi e le relazioni tra le classi. Le classi sono differenziate in persistenti e transitorie.
2. Il sottoinsieme delle classi persistenti viene mappato nello schema relazionale (**Schema Relazionale**), tramite il processo denomitato **Classi verso Tabelle.**
3. Lo schema relazionale viene implementato utilizzando un RDBMS tramite il processo di **Generazione dello Schema Interno**, questo processo genera una serie di tabelle che mantengono lo stato persistente dell'applicazione iniziale (**Tabelle dell'Applicazione**).

4. L'Object Model dell'applicazione viene utilizzato inoltre per la **Generazione del Codice** che costituisce l'implementazione delle classi , sia le **Classi Transitorie** sia le **Classi Persistenti.**
5. Al sottoinsieme delle classi persistenti viene aggiunto un ulteriore strato di software (**Object Wrapper Layer**) che mette a disposizione di ogni singola classe un'interfaccia pubblica per l'interazione con il RDBMS sottostante. Le classi transitorie dell'applicazione associate alle classi persistenti non devono conoscere lo schema relazionale ma solamente utilizzare l'interfaccia pubblica.
6. Per accedere al database *ORI* deve conoscere come gli oggetti (classi) sono mappati nelle corrispondenti tabelle. Queste informazioni risiedono in un **Database di Configurazione** ottenuto a partire dallo Object Model dell'applicazione e dallo Schema Relazionale tramite il processo denominato **Generazione del Mapping**.

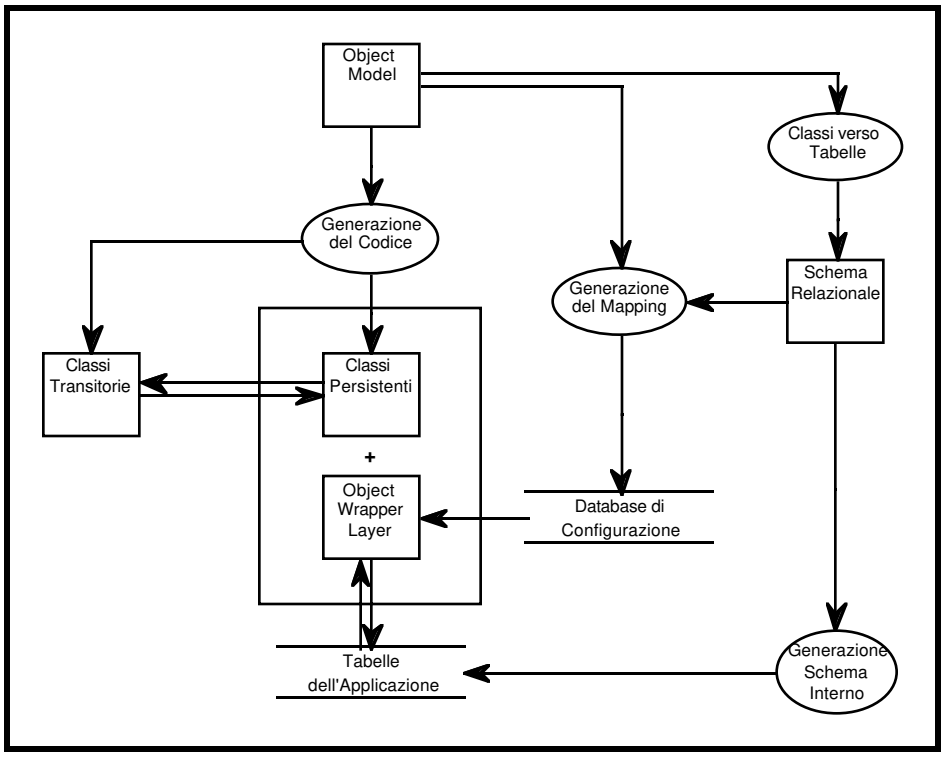

Figura 1 -Architettura Funzionale

L'interfaccia pubblica messa a disposizione da *ORI* gestisce due tipi di operazioni:

- l'inserimento, la modifica e la cancellazione di un singolo oggetto.
- il recupero di un insieme di oggetti tramite la specifica della condizione di ricerca. Un predicato di ricerca è costituito da un insieme di condizioni semplici sugli attributi della classe. Gli oggetti recuperati sono solo una vista di primo livello degli oggetti memorizzati; essi contengono i valori degli attributi che mappano su domini semplici ma non puntatori ad oggetti associati. Le interrogazioni complesse vengono espresse in maniera navigazionale tramite l'invocazione di interrogazioni semplici.

La Figura 2 descrive l'interfaccia pubblica di *ORI*:

1. **ApplicationData** è la classe base, ereditata da ciascuna classe persistente dell'applicazione, in cui sono dichiarati i metodi per l'inserimento (**insertObj()),** l'aggiornamento (**updateObj()**) e cancellazione (**deleteObj()**) di un oggetto.
2. Per ciascuna classe persistente viene definita una classe **Select** specializzazione di quella mostrata in figura che gestisce la funzionalità di recupero degli oggetti.
3. La condizione di ricerca dell'interrogazione viene espressa dall'associazione **IsComposedOf** tra la classe Select ed un insieme di fattori booleani (**BooleanFactor**). Ciascun fattore booleano è costituito da una terna: nome attributo, operatore, valore attributo.
4. Il risultato dell'interrogazione viene espresso dalla associazione **HasResult** tra la classe Select e un insieme di istanze di una classe dell'applicazione.

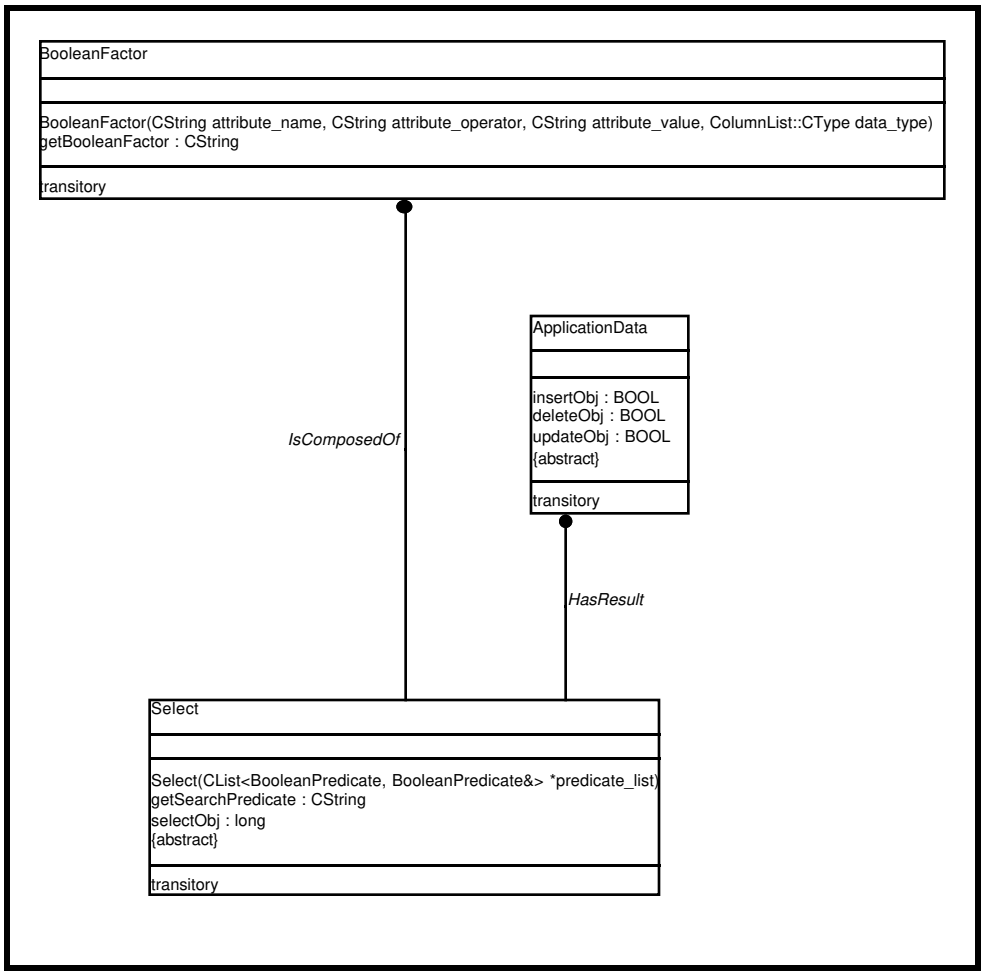

Figura 2 - Object Model di ORI

Con l'attuale implementazione l'uso di *ORI* in una applicazione richiede la codifica delle definizioni delle classi persistenti. Un rilevante miglioramento previsto e' l'implementazione della generazione automatica del codice in base all'interfaccia delle classi e alla struttura relazionale.