A Seamless Framework for Object-Oriented Persistence in Presence of Class Schema Evolution

Author(s):
Piccioni, Marco

Publication Date:
2012

Permanent Link:

https://doi.org/10.3929/ethz-a-007600524 →

DISS. ETH NO. 20910

A Seamless Framework for Object-Oriented Persistence in Presence of Class Schema Evolution

A dissertation submitted to
ETH ZURICH

for the degree of
Doctor of Sciences

presented by
MARCO PICCIONI
Laurea degree in Mathematics, Università degli Studi di Roma La Sapienza, Italia
Master in Economics, Università Commerciale L. Bocconi, Italia

born
July 7th, 1965

citizen of
Italy

accepted on the recommendation of

Prof. Dr. Bertrand Meyer, examiner
Prof. Dr. Harald Gall, co-examiner
Prof. Dr. Martin Robillard, co-examiner

2012

Abstract

Many object-oriented software systems rely on persistent data. Such systems also evolve over time. These two equally valid needs clash: a system may attempt to retrieve data whose description in the program has changed. A typical scenario involves an object retrieval failure because of an evolution in the corresponding class. Another, more critical one occurs when a retrieval algorithm accepts objects whose class invariants are not valid anymore. This may happen because in most programming languages class invariants are not made explicit in the code. While in the first scenario a runtime failure will typically point out the issue, the second scenario may lead to invalidating the consistency of the whole system.

The research and software development community has mainly been trying to tackle the first scenario above by providing software developers with the possibility of writing transformation functions to adapt old objects to the new classes. This thesis contributes to this effort by devising a software framework that provides a solution to both scenarios described above.

The framework provides a concise model for persistence-affecting class schema modifications, and an infrastructure keeping track of all class versions and system releases, recognizing the class schema modifications statically, that is before they cause retrieval errors. The framework then further supports developers by using heuristics to generate transformation functions. At retrieval time, a version-aware retrieval algorithm leverages the information previously collected, and together with a class invariant validation step handles object retrieval in a safer way. The algorithm is part of a persistence library seamlessly supporting multiple persistence backends.

To establish the relevance of the problem under study and to validate the choice of class schema modifications among the many possible, this thesis presents the results of four empirical studies performed on existing software libraries and applications, written in both Java and Eiffel. These studies suggest that the problem of a semantically consistent class schema

evolution exists, and that the chosen subset of schema modification operators is viable.

This thesis also suggests a measure for the robustness of evolving object-oriented applications that persist their objects. The measure is meant to offer a simple way to evaluate current and future software projects with respect to their ability to retrieve previously stored objects. The proposed measure is computed and evaluated over an existing Java library.

An additional contribution consists in a study of a phenomenon not analyzed before: the evolution of class invariants and its possible effects on object retrieval. The analysis of eight software projects led to the conclusion that while developers do write class invariants, such activities as adding and removing attributes do not lead, as one might expect, to stronger or weaker invariants in term of number of clauses. This suggests that class invariant evolution constitutes one more potential risk to the consistency of the retrieval operations, because for example the newly added attributes' values could be stored and then retrieved without being guarded by a corresponding class invariant clause.

The usability of the framework front-end API is assessed by performing an exploratory study involving 25 object-oriented developers that had never seen the API before. The study results show that while the API can be used for commonplace database access tasks without resorting to external documentation and specific database knowledge, it also has some usability issues that need to be addressed.

In conclusion, this thesis provides a seamless and invariant-safe solution to the problem of class schema evolution of object-oriented software, encompassing the time in which a system is released, the time in which objects are stored, and the time in which object are retrieved.

RIASSUNTO

Molti sistemi software orientati agli oggetti fanno affidamento su dati persistenti. Tali sistemi evolvono anche nel tempo. Questi due aspetti, entrambi legittimi, confliggono, perché un sistema potrebbe leggere dei dati la cui descrizione nel programma è cambiata. Uno scenario tipico vede la lettura di un oggetto fallire per via di una evoluzione nella classe corrispondente. Uno scenario piú critico si ha quando un algoritmo di lettura dati accetta degli oggetti i cui invarianti di classe non sono piú validi. Ciò può accadere perch'e in molti linguaggi di programmazione gli invarianti di classe non sono resi espliciti nel codice. Mentre nel primo scenario un errore a tempo di esecuzione evidenzierà il problema, il secondo scenario potrebbe causare una mancanza di consistenza per l'intero sistema.

La comunità dei ricercatori e degli sviluppatori software ha tentato di risolvere principalmente il primo scenario menzionato sopra dando agli sviluppatori la possibilità di scrivere funzioni di trasformazione con lo scopo di adattare i vecchi oggetti alle nuove classi. Questa tesi contribuisce a questo tentativo proponendo un framework software che fornisce una soluzione per entrambi gli scenari sopra menzionati.

Il framework propone un conciso modello per i cambiamenti nella struttura delle classi che sono rilevanti rispetto alla persistenza, ed una infrastruttura che tiene conto di tutte le versioni di una classe e di tutti i rilasci di un software, identificando i cambiamenti nella struttura delle classi staticamente, cioè prima che causino errori di lettura. Il framework poi supporta ulteriormente gli sviluppatori usando delle euristiche per generare funzioni di trasformazione. Quando è il momento di leggere i dati, un algoritmo che sa come gestire le versioni di una classe utilizza le informazioni raccolte in precedenza, e con uno step di verifica dell'invariante gestisce la lettura degli oggetti in un modo piú sicuro. L'algoritmo fa parte di una libreria di persistenza che supporta senza soluzione di continuità molti back-end.

Per stabilire la rilevanza del problema sotto studio e per validare la scelta dei cambiamenti nella struttura delle classi tra i molti possibili, que-

sta tesi presenta i risultati di quattro studi empirici condotti su applicazioni e librerie software esistenti, scritte sia in Java che in Eiffel. Tali studi suggeriscono che il problema della consistenza nella semantica dell'evoluzione della struttura delle classi esiste, e che il sottoinsieme scelto di operatori dei cambiamenti nella struttura e' in grado di funzionare.

Questa tesi suggerisce anche una misura per la robustezza di applicazioni orientate agli oggetti che evolvono e salvano gli oggetti stessi. La misura intende offrire un modo semplice per valutare progetti software presenti e futuri rispetto alla loro capacità di leggere oggetti salvati in precedenza. La misura proposta è calcolata e valutata su di una libreria Java esistente.

Un contributo aggiuntivo consiste nello studio di un fenomeno non analizzato in precedenza: l'evoluzione di invarianti di classe e i suoi possibili effetti sulla lettura degli oggetti. L'analisi di otto progetti software ha portato alle seguenti conclusioni: gli sviluppatori di fatto scrivono gli invarianti di classe, ed attività come aggiungere o rimuovere attributi non portano, come potremmo aspettarci, ad invarianti piú forti o piú deboli in termini di numero di clausole. Ciò suggerisce che l'evoluzione degli invarianti di classe costituisce un risco potenziale aggiuntivo per la consistenza delle operazioni di lettura, perché per esempio i valori dei nuovi attributi potrebbero essere salvati e poi letti senza essere verificati dalle corrispondenti clausole dell'invariante di classe.

L'usabilità della front-end API del framework è stato valutato tramite uno studio esplorativo che ha coinvolto 25 sviluppatori orientati agli oggetti che non avevano mai visto l'API prima. I risultati dello studio mostrano che mentre l'API può essere usata per effettuare delle comuni operazioni di accesso ad una base di dati senza utilizzare documentazione esterna o presumere specifiche conoscenze di basi di dati, presenta anche dei problemi di usabilità che devono essere affrontati.

In conclusione, questa tesi propone una soluzione al problema della evoluzione della struttura delle classi nel software orientato agli oggetti che è uniforme e sicura rispetto agli invarianti id classe, e che va dal momento in cui un sistema viene rilasciato al momento in cui gli oggetti vengono letti, passando per quando gli oggetti vengono salvati.