ABSTRACT

Call Level Interfaces (CLI) are software API used for building business tiers of relational database applications whenever performance is a key requirement. Nevertheless, their use is cumbersome, mainly in large database applications with many and complex Create, Read, Update and Delete (CRUD) expressions. CLI are low level API conveying several difficulties during the development process of relational business tiers. Four of them are herein emphasized: 1) Programmers need to master the schemas of the underlying databases; 2) the same CRUD expression is frequently re-written to address different business needs; 3) CLI are not suited to cope with evolving business tiers and, finally, 4) CLI do not provide any feature to decouple development process of relational business tiers from the development process of application tiers. To tackle these difficulties, this paper proposes an architecture for building reusable relational business tier components based on CLI herein referred to as the Reusable Business Tier Architecture (RBTA). It relies on a customizable wide typed service to address a business area, such as accountability. The typed service is able to manage CRUD expressions, deployed at runtime, on behalf of application tiers and in accordance with users’ needs. The only constraint is that the required service to manage each CRUD expression must be a sub-set of the implemented wide typed service. A proof of concept is also presented.

Keywords: Business Tiers, Call Level Interfaces, Components, Databases, Middleware, Software Architecture, Software Reuse

INTRODUCTION

Object-oriented and relational paradigms are simply too different to bridge seamlessly, leading to a set of difficulties informally known as impedance mismatch (David, 1990). Impedance mismatch derives from the diverse foundations of both paradigms and it has been an open issue for more than 50 years (Cook & Ibrahim, 2005). To tackle the impedance mismatch, several solutions have been proposed, including Call Level Interfaces (CLI), Embedded SQL,
object-to-relational mapping techniques (O/RM), language extensions and persistent frameworks. These solutions are geared to deal with and hide all the complexity of the translation between the two paradigms. Among the available solutions, CLI are considered the correct option whenever a fine tune control is needed to interact with databases (Cook & Ibrahim, 2005). In spite of this important advantage, CLI also convey some drawbacks. Four of the most relevant drawbacks are: 1) programmers need to completely master the schemas of the underlying databases; 2) Create, Read, Update and Update (CRUD) expressions are frequently re-written over and over again whenever similar business needs have to be addressed; 3) whenever a new CRUD expression is needed there is no other option than write the necessary source-code to manage its execution and 4) CLI do not provide any architectural policy to decouple development process of business tiers from the development process of application tiers. These drawbacks are increasingly relevant when schemas of databases get more and more complex, when the number of CRUD expressions increases and when the complexity of CRUD expressions increases. In reality, CLI are general low level API that do not provide any architectural policy to be followed during the development process of business tiers to address any of the mentioned drawbacks. CLI were mainly devised to address the impedance mismatch issue. To tackle these CLI drawbacks, a research has been carried out in the context of Component-Based Software Engineering (Heineman & Councill, 2001; Szyperky, Gruntz, & Murer, 2002). Component-based development aims at composing software units from other pre-built software units (Heineman & Councill, 2001). At the end, a final software system is not built as a unique block but as a composite of software units known as components (Kung-Kiu & Zheng, 2007). A key aspect for the success of any component is its capability of being adapted to be reused (Bracciali, Brogi, & Canal, 2005) which is improved if another key aspect is also considered: the reuse of computation (Elizondo & Lau, 2010). Thus, this research is focused on creating a software architecture for building adaptable business tier components, herein referred to as the Reusable Business Tier Architecture (RBTA). The adaptation process is carried out in two phases: the static phase and the dynamic phase. During the static phase, reusable business tier components (RBTC) are automatically built from a business schema to comprise a wide typed service able to address a business area, such as accountability, warehouse or sales. During the dynamic phase, CRUD expressions are deployed to RBTC, at runtime, into a pool to address specific users’ needs. RBTC are capable of managing any CRUD expression the schema of which is a sub-set of the implemented wide typed service.

Throughout this paper, all examples are based on Java, Java Database Connectivity (JDBC) (Parsian, 2005), T-SQL (SQL Server) and Northwind Microsoft database (Microsoft). The presented code may not execute properly since we only show the relevant parts for the points under discussion.

This paper is structured as follows: section 2 presents the motivation; section 3 presents an overview of the related work; section 4 introduces Call Level Interfaces; section 5 presents the RBTA; section 6 discusses some aspects of RBTA and, finally, section 7 concludes and presents future work.

**MOTIVATION**

Software systems have increasingly played a key role in small, medium and large organizations by managing the data from which everyday decisions are taken. Data is mostly kept and managed by database management systems. Among the several paradigms, the relational database management systems (RDBMS) continue to be one of the most successful to manage data and, therefore, to build database applications. To be useful, data needs to be inserted, updated and retrieved to be processed. Here CLI play a key role whenever a fine tune control is required to manipulate the interaction with RDBMS. Nevertheless, their use is cumbersome mainly
Related Content

Person Identification Using Top-View Image with Depth Information
[www.igi-global.com/article/person-identification-using-top-view-image-with-depth-information/89776?camid=4v1a](www.igi-global.com/article/person-identification-using-top-view-image-with-depth-information/89776?camid=4v1a)

Information Theoretic XSS Attack Detection in Web Applications
[www.igi-global.com/article/information-theoretic-xss-attack-detection-in-web-applications/118145?camid=4v1a](www.igi-global.com/article/information-theoretic-xss-attack-detection-in-web-applications/118145?camid=4v1a)

Requirements Engineering for Courseware Development
[www.igi-global.com/chapter/requirements-engineering-courseware-development/28409?camid=4v1a](www.igi-global.com/chapter/requirements-engineering-courseware-development/28409?camid=4v1a)