HARRAN ÜNİVERSİTESİ MÜHENDİSLİK DERGİSİ

HARRAN UNIVERSITY JOURNAL of ENGINEERING

e-ISSN: 2528-8733 (ONLINE)

C++ Programlama Dili için Yerel Nesne-İlişkisel Haritalama Yaklaşımı

A Native Approach to Object-Relational Mapping (ORM) for C++ Programming Language

Yazar(lar) (Author(s)): *Abdullatif KALLA*[1], *Mehmet Emin TENEKECİ*[2]

[1] ORCID ID: 0000-0001-5125-0411
[2] ORCID ID: 0000-0001-5944-4704

Bu makaleye şu şekilde atıfta bulunabilirsiniz (To cite to this article): Kalla A., Tenekeci M.E., "C++ Programlama Dili için Yerel Nesne-İlişkisel Haritalama Yaklaşımı", ***Harran Üniversitesi Mühendislik Dergisi***, 8(2): 151-158, (2023).

DOI: 10.46578/humder.1219421

HRU Muh Der, 8(2): 151-158 (2023)

Harran Üniversitesi
Mühendislik Dergisi

Araştırma Makalesi

C++ Programlama Dili için Yerel Nesne-İlişkisel Haritalama Yaklaşımı

Abdullatif KALLA[1], Mehmet Emin TENEKECİ[1,*]

[1]*Harran Üniversitesi, Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü, 63100, Haliliye/ŞANLIURFA*

Makale Bilgisi

Başvuru: 15/12/2022
Yayın: 31/08/2023

Anahtar Kelimeler

C++
SQL
ORM
Veritabanı

Keywords

C++
SQL
ORM
Database

Öz

En popüler programlama dillerinin çoğu, nesne yönelimli olarak bilinen dillerdir. Bu diller ile geliştirilmiş uygulamaların çoğu da, ilişkisel veritabanları kullanmaktadır. Yapısal sorgulama dili olan SQL ise ilişkisel bir Veritabanı ile etkileşim kurmak için kullanılan ve verileri düzenlemek, yönetmek ve almak için kullanılan, standart haline gelmiş bir araçtır. Bu iki teknolojinin arasındaki temel farklılıklar nedeniyle sorunsuz bir şekilde birlikte çalışmamaktadır. Aralarındaki uyumsuzluklar, nesne/ilişkisel empedans uyumsuzluğu sorunları olarak ortaya çıkmaktadır. Bu nedenle, verileri o iki ortam arasında dönüştürebilmek için bir haritalama katmanına sahip olmak oldukça gereklidir. Bu soyutlama katmanı, uygulama nesnelerini otomatik olarak veritabanı kayıtlarına eşler; bu şekilde veritabanı ile etkileşim için gerekli kod miktarında azalma olmaktadır. Ayrıca verilerin tutulduğu veri tabanını yazılımcıdan soyutlamaya yardımcı olur. Böylelikle, kodun testini ve bakımını daha kolay kılar. Bu tür araçlara nesne/ilişkisel haritalama aracı denilir (Object/Relational Mapping - ORM). Çoğu programlama dili için çeşitli ORM araçları geliştirilmiştir. Ancak C++ için etkili bir ORM aracı bulunmamaktadır. Bu çalışmada C++'ın şablon (template) programlama ile esnek adresleme özellikleri kullanılarak veritabanı işlemleri için kullanılan SQL komutları seri hale getirilmektedir. Geliştirilen C++ ORM katmanı yerel olarak uygulamak için yeni bir yaklaşım tanıtılmaktadır.

A Native Approach to Object-Relational Mapping (ORM) for C++ Programming Language

Abstract

The most popular programming languages are object-oriented, and most of their applications use relational databases. SQL, a structured query language, is a standardized tool for interacting with a relational Database and used to organize, manage, and retrieve data. Due to the fundamental differences between these two technologies, they do not work together seamlessly. Incompatibilities between them manifest as object/relational impedance mismatch issues. Hence, it is quite necessary to have a mapping layer to be able to transform the data between those two worlds. This abstraction layer automatically maps application objects to database records; in this way, the amount of code required to interact with the database is reduced. It also helps in isolating the database where the data is kept from the software developer. This makes the code easier to test and maintain. Such tools are called Object/Relational Mapping (ORM) tools. Various ORM tools have been developed for almost every programming language. However, there is no effective ORM tool for C++. In this study, SQL commands used for database operations are serialized by using two flexible features of C++; templates and pointers. A new approach is introduced to natively implement the ORM layer in C++.

1. GİRİŞ (INTRODUCTION)

Nesne yönelimli programlama (Object-Oriented Programming - OOP), veriler ve veriler için kullanılan prosedürlerin birlikte bir sınıf olarak tanımlanıp kullanılabildiği bir programlama teknolojisidir. OOP teknolojisi ile geliştirilen yazılımlarda, kullanılan yapılara ait sınıflar tanımlanır. Bu sınıflardan oluşturulan nesnelerden veri okuma ve değiştirme gibi işlemler nesne içerisinde tanımlanmış fonksiyonlar ile gerçekleştirilir. OOP'de nesneler için yeni bir veri tipi olarak sınıflar (class) tanımlanır. Bu sınıflardan

*İletişim yazarı, e-mail: etenekeci@harran.edu.tr

nesne (object) oluşturulur. Oluşturulan nesnelerin özellikleri sınıfın veri değişkenleri (attribute) ve işlevleri ise metotları (method) tanımlar.

C++ programlama dili, C programlama dilinin özelliklerini miras almış ve nesne kavramını eklemiştir. Geliştirilen bu yeni dil, şablon meta programlama (Template Metaprogramming - TMP) yetenekleri bulunmaktadır. C++, ilk nesne yönelimli programlama dili olmamasına rağmen en yaygın kullanım alanı bulan dillerden biridir. C++ orta seviyeli bir dil olması ile birlikte performans açısından yazılımcılar tarafından tercih edilmektedir.

Veritabanları verilerin standart bir şekilde tutulmasını sağlamak için geliştirilmiştir. Öncesinde uygulama yazılımları ihtiyaçlarına göre farklı dosya yapılarına sahip dosya formatlarında verileri tutarlardı. Bu dosyalar yazılımcının ihtiyacına uygun hazırlanmış formattaydılar. Farklı uygulamaların, bu dosyaları kullanabilmesi için dosya formatları ve tutulan verinin yapısı hakkında bilgi sahibi olmaları gerekmekteydi. Farklı uygulamalar tarafından erişim kontrolü ve tutarlılığın sağlanması da üstesinden gelinmesi gereken bir başka problemdi.

Veritabanların ortaya çıktığı zamandan bu yana hızlı ve köklü bir evrim sürecinden geçti. Hiyerarşik veritabanı modeli ise yukarda bahsedilen problemlerinin üstesinden gelmek için geliştirildi. Ağ veri tabanı modeli, hiyerarşik veritabanındaki veri tekrarı problemini ortadan kaldırmak için icat edildi. İlişkisel veri modeli, herhangi bir uygulama yazılımı tarafından kayıtlı verilerin sorgulanması için standart bir yöntem sağlandı. İlişkisel veritabanı modelinde veriler tablolar halinde tutulmaktadır. Bu şekilde veriler yapılandırılmış bir şekilde tutularak esnek bir şekilde sorgulama imkânı sağlanır. Veritabanında işlemler yapılması için yapısal sorgulama dili (Structured Query Language -SQL) geliştirildi. Bu şekilde standart bir yapıda saklanan veriler üzerinden işlem yapacak standart bir dil kullanılmış oldu. SQL dili ile tabloların yapısal tasarım, değişiklik ve silmesi gerçekleştirilirken, verilerin eklenmesi, silinmesi, güncellenmesi ve sorgulanması yapılabildi. Matematiksel bir forma sahip bu dil farklı programlama ortamları ile birlikte kullanılarak tüm işlemleri standart hale getirdi [1].

Günümüze kadar, nesne yönelimli programlama yazılım geliştirmede baskın teknoloji olduğu gibi, veritabanı pazarında da ilişkisel model baskın teknolojidir. Yazılım sistemlerinin çoğu, özellikle kurumsal bilgi sistemleri, bu iki teknolojiyi kullanıyor. Bu çalışmada da bu iki ortamın birbirinden soyutlanmasını sağlayacak ve yazılım geliştiricilerin Veritabanı modelinden ve sorgulama dilinden bağımsız OOP yapısı ile veri işlemlerini gerçekleştireceği bir ara katman mimarisi geliştirilecektir.

Literatürde, C++ dili için birçok ORM aracı bulunmaktadır. Ancak bu C++ araçları, C++'ın diğer yüksek seviyeli programlama dilleri gibi bazı yeteneklere sahip olmadığından dolayı, kullanım ve özellik açısından diğer dillerin araçlarına göre bazı sınırlamalara sahiptir. Bu sınırlamalar dört temel probleme aittir:

1- Teknoloji yığını bağımlılığı (technology stack dependency).

2- Kullanıcı kodu değiştirtmesi (intrusion).

3- Makro kullanımı (code manipulation).

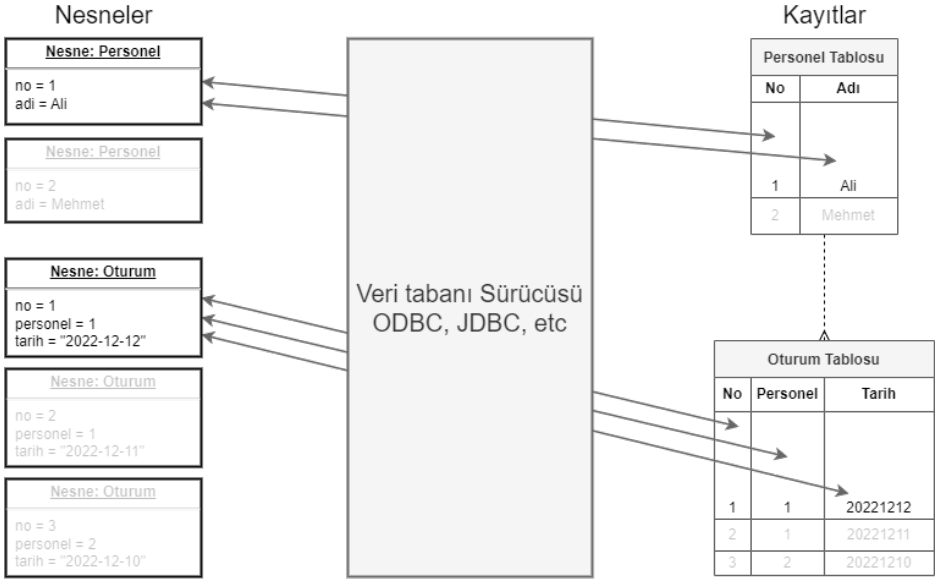
4- Kod analizi ve üretmesi (code parsing and generation).

QxOrm [7] ile TreeFrog [10], Qt çerçevesine bağlı olduğundan dolayı birinci sınırlamaya sahiptir. Bu nedenle, bu araçlar bağlı oldukları teknoloji yığını kullanmayan yazılımlarda kullanılamamaktadır. Zhang [5], Oat++ [8], Wt::Dbo[9], TreeFrog [10], Lithium [11], ve Hiberlite [12] ikinci sınırlamaya sahiptir. Dolayısıyla, bu araçlar mevcut olan yazılım yapılarını değiştirtmeden (inheritance veya aggregation yoluyla) çalışmamaktır. QxOrm [7], Oat++ [8], Hiberlite [12], ve Bun [13] üçüncü sınırlamaya sahiptir. Yalnız makrolar, daha sonraki bölümlerde anlatılacağı gibi birçok sebepten ötürü programlama, derleme ve debug süreçlerini zor kılmaktadır. ODB [6], TreeFrog [10], ve Lithium [11] ise dördüncü sınırlamaya sahiptir. Fakat özel derleyici veya harici kod üretme araçları projenin build sistemini yavaşlatıp ve daha karışık etmektedir. Bu farklılıklara rağmen, tüm bu ORM araçları temel özelliklere sahiptir (nesne-ilişkisel haritalama, nesne kalıcılığını sağlama ve CRUD işlemlerini işleme).

Önerilen yaklaşımda, yukarıdaki bahsedilen sınırlamaların hiçbiri söz konusu değil.

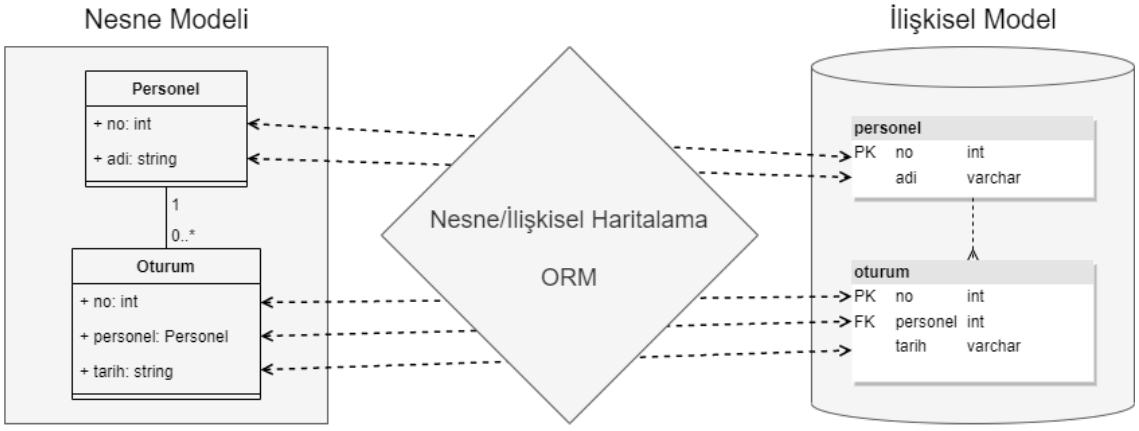
2. NESNE/İLİŞKİSEL HARİTALAMA (OBJECT/RELATIONAL MAPPING)

Nesne yönelimli dillerde oluşturulan nesneler hakkında bilgiye ve nesnenin kendisine ulaşılabilmesine nesne kalıcılığı (object persistence) denir. Çalışma zamanında kalıcı duruma getirilen nesnelerin bellekte silinse bile ulaşılabilir olması nesne yönelimli programlama dilleri için gereklidir. Veritabanı etkileşimi bulunan yazılımlar için de tablolardaki bilgilerin kalıcı duruma getirilmesi için tabloların sütunlarının sınıflardaki özellikler ile eşleştirilmesi ve bu sınıfların kalıcı duruma getirilmesi ile mümkündür. Şekil 1' de kalıcı duruma getirilen bir nesnenin ilgili tabloya bir satır olarak kaydedilebileceğini göstermektedir.



Şekil 1. Uygulama nesneleri ilgili tablolardaki kayıtlar ile eşleştirilmesi

Nesne/ilişkisel haritalama (Object-Relational Mapping - ORM), kaynak kod ortamında bulunan sınıfın veritabanındaki tablo ile eşleştirilmesidir. Bu eşleştirmede tablonun sütunları sınıfın kalıcı değişkenlerine karşılık gelecek şekilde kodlama yapılmaktadır. Eşleştirme ile sınıf üzerinde gerçekleştirilecek işlemler tabloları etkileyecek şekilde kodlama yapılabilmektedir. Sınıf üzerinde gerçekleştirilecek verilerin kaydedilmesi, değiştirilmesi, silinmesi ve sorgulanması işlemlerini veritabanına yansıtarak gerçekleştirebilmektedir. Yazılım tarafından bu işlemlerin veritabanına uygulanması için veritabanı erişim bileşenleri kullanılır. Bu bileşenler ara sürücüler kullanılarak iki ortam arasında bağlantı kurmaktadır. ORM veritabanına erişim katmanı ile sunum katmanı arasında veritabanına erişim süreçlerini soyutlayan bir ara katman olarak görev yapmaktadır. Şekil 2, nesnelerin kalıcılığını, bir ORM aracı kullanarak, otomatikleştirilebildiğini göstermektedir.



Şekil 2. Nesne/İlişkisel haritalama otomatikleştirmesi

İlişkisel veritabanları ile nesne yönelimli diller arasında bazı kullanım farklılıkları bulunmaktadır. Bu uyumsuzluklara yol açan farklılıklar nesne yapıları ile veritabanı tablolarının kullanım farklılığından kaynaklanmaktadır. Nesneler veri ile birlikte fonksiyonları birlikte sarmalanmış olarak tutmaktadır. Ancak tablolarda sadece veriler bulunmaktadır. Nesneler arası iletişim nesnelerin adresleri ile gerçekleşirken tablolar birbirleri ile tablolar arasında bağlantıyı sağlayan anahtar ve yabancı sütunlar ile gerçekleşmektedir. Nesnelerde miras alma ve çok şekillilik (polymophism) ile tip dönüşümleri gerçekleşmektedir. Ancak bu özellikler Veritabanı tablolarında bulunmamaktadır. Bu farklılıklara nesne/ilişkisel uyumsuzluğu (Object/Relational Impedance Mismatch - ORIM) denir. ORM yapılarında bu tip durumlar dikkate alınarak üstesinden gelebilecek kısıtlamalar kullanılmalıdır.

ORIM durumlarının göz önüne alınarak giderilmesi gerekmektedir. Bunun için farklı stratejiler geliştirilmiştir. Bunlardan en bilineni Gömülü (embedded) SQL kullanımı ile veritabanı sorgularının kod tarafında hazırlanarak uygulanması olacaktır. Bunun için JDBC ve ODBC gibi bileşenler kullanılmaktadır. Bu uygulama ORM mantığının tersine tüm veritabanı yükünü programcıya bırakmaktadır. Veritabanı ile nesneler arası farklılığı ve veritabanları arası farklılıkları programcı bilerek aradaki uyumu kod ile sağlaması gerekmektedir. Bu durumun programcıdan bağımsız olarak üstesinden gelmek ve otomatik uyum problemini ortadan kaldırmak için pek çok nesne/ilişkisel haritalama (Object/Relational Mapping - ORM) çerçevesi geliştirildi.

Bu çerçeveler iki farklı ortam arasında dönüştürücü rolünü üstlenmiştir. ORM'ler nesneler ile tablolar arası eşleştirme görevi görerek dönüşümü otomatikleştirmekte ve böylece yazılacak kod miktarını azaltmaktadır. Kod içerisinde yazılmış SQL komutları veya saklı yordamlardan daha az kod yazılarak geliştirme ve bakım süresini azaltmaktadır. Bununla birlikte nesne kalıcılığı özelliği ile veritabanı iletişiminde kod testini ve bakımını kolaylaştırmaktadır. ORM katmanı sayesinde veritabanı modeli geliştirici tarafından daha kolay anlaşılabilir ve veritabanı sunucusu ile daha az iletişim kurulmasında daha performanslı uygulamalar geliştirilebilmektedir [2] [3] [4].

OOP dillerinde nesneler ile ilişkisel veritabanı tablolarını eşleştirebilecek açık kodlu ve ticari araçlar geliştirilmiştir. Ticari olarak ilk başarılı ORM çerçeve Smalltalk ürünü olan TOPLink idi, daha sonra Java'ya taşındı. Birkaç yıl sonra Hibernate, Java'da açık kaynaklı bir proje olarak ORM yaklaşımını gerçekleştirmeye başladı. Bugün, her popüler geliştirme platformu için çeşitli ORM çerçeveleri mevcuttur. Java platformu birçok popüler ORM çözümüne sahiptir; JPA standardı uygulamalarına ek olarak TOPLink, MyBatis, Cayenne; Hibernate, EclipseLink ve OpenJPA. Python platformu iyi bilinen SQLAlchemy çerçevesine sahiptir. Ruby platformu, ana ORM çözümü olarak ActiveRecord'a sahiptir. PHP ise Doctrine'e sahiptir, Bookshelf, Node.js platformu içindir. Öte yandan C++, olgun ORM çözümü olarak yalnızca ODB'ye [6] sahiptir. Ancak ODB bir çerçeve değil, gerekli ilişkilendirme kodunu oluşturmak için bir derleyici eklentisine bağlı olan bir kütüphanedir. C++ için başka ORM çözümleri de mevcuttur, ancak yaygın olarak kullanılmalarını engelleyen birkaç sınırlama vardır. Bunlar;

- QxOrm [7], Qt çerçevesini temel alan bir ORM kütüphanesidir. Qt kullanmayan C++ projeleri için uyumlu değildir. Ayrıca, geliştiricinin, sınıf değişkenlerini/metotlarını kaydetmeye ek olarak, kalıcı olması için sınıfın kaynak dosyalarını (.hpp ve .cpp) manuel olarak kaydetmesini gerektirir.
- Oat++ [8], ORM bileşenine sahip açık kaynaklı bir web çerçevesidir. Çok müdahaleci ve yalnızca sınıfın açık (public) değişkenleriyle çalışabilir.
- Wt [9], web uygulamalarını geliştirmek için kullanılan bir kütüphanedir. Kendisinden bağımsız olarak kullanılabilen bir ORM bileşeni olan Wt::Dbo'yu içerir. Oat++ ile aynı sınırlamalara sahiptir.
- TreeFrog [10], Web uygulamaları geliştirmek için tam yığın (full stack) bir C++ çerçevesidir. Qt'ye bağlıdır ve QxOrm ile aynı sınırlamalara sahip bir ORM bileşenine sahiptir, ayrıca müdahalecidir ve yalnızca sınıfın açık (public) değişkenleriyle çalışabilir.
- Lithium [11], bir web çerçevesidir ayrıca kendi ORM çözümüne de sahiptir. Oat++ ile aynı sınırlamalara sahiptir.
- Hiberlite [12], aktif kayıt modelini (Active Record Pattern) izleyen bir ORM çerçevesidir. Wt ile aynı sınırlamalara sahiptir.

Bun [13], bir ORM kütüphanesidir. Müdahaleci değildir, ancak sınıf değişkenlerini kaydetmek için makrolar kullanır ve yalnızca sınıfın açık (public) değişkenleriyle çalışabilir.

3. SERİLEŞTİRME VE İÇGÖZLEM (SERIALIZATION & INTROSPECTION)

Veritabanı, SQL sayesinde veri alışverişini seri halde (serialized) gerçekleştirebilir. Böylece, nesneleri seri hale getirebildiğimiz takdirde (örneğin ikili biçimde) nesne/ilişkisel haritalamayı da gerçekleştirebileceğimiz anlamına gelir. Bu nedenle, öncelikle nesne modelini veritabanının beklediği ikili biçime ve ikili biçimden seri hale getirebilmemiz gerekir. C++'da yansıma yeteneklerinin olmaması nedeniyle, sınıf üyeleri hakkındaki bilgiler derleme zamanında ve hatta çalışma zamanında mevcut değildir. Bu nedenle, sınıfın içyapısı hakkında bir şekilde bilgi edinmeye ihtiyaç vardır. ODB [6], bu sorunu çözmek için bir derleyici eklentisi geliştirmiştir. Bazı ORM sağlayıcıları [7, 8, 12, ve 13] ön işlemeci (Preprocessor) yönergeleri (directives) kullanmıştır. Ancak, makrolar birçok sebepten ötürü (basit metin ikamelerini tanımladıkları için) semantik olarak geçerliliği valide edilmesi mümkün değil ve kaçınılması mümkünse modern C++ kodunda kullanılmaları önerilmez. Diğer bazı ORM sağlayıcıları, nesne kalıcılığı mekanizmalarına erişim sağlamak için sınıf yapısına bazı yönergeler/metotlar eklemeyi tercih etti [5, 8, 9, 10, 11, ve 12]. Ancak bu teknik, kullanıcının koduna müdahale ve sağlayıcının koduna özgü olur. Birkaç kütüphane, sınıf yapısı ayrıntılarını harici bir nesnede saklamak için meta-nesne yaklaşımını izledi. Ancak, sarmalama (encapsulation) sınırlamaları ile karşı karşıya kaldılar. Bu yazıda, bu yaklaşımdan yola çıkarak, makro kullanmadan ve müdahaleci olmayan bir şekilde saf C++'da belirli bir sınıf yapısı hakkındaki tüm ayrıntıları depolamak için C++'in şablon (template) ve işaretçi (pointer) özelliklerini kullanarak nesne/ilişkisel haritalama problemi için yeni yöntem geliştireceğiz.

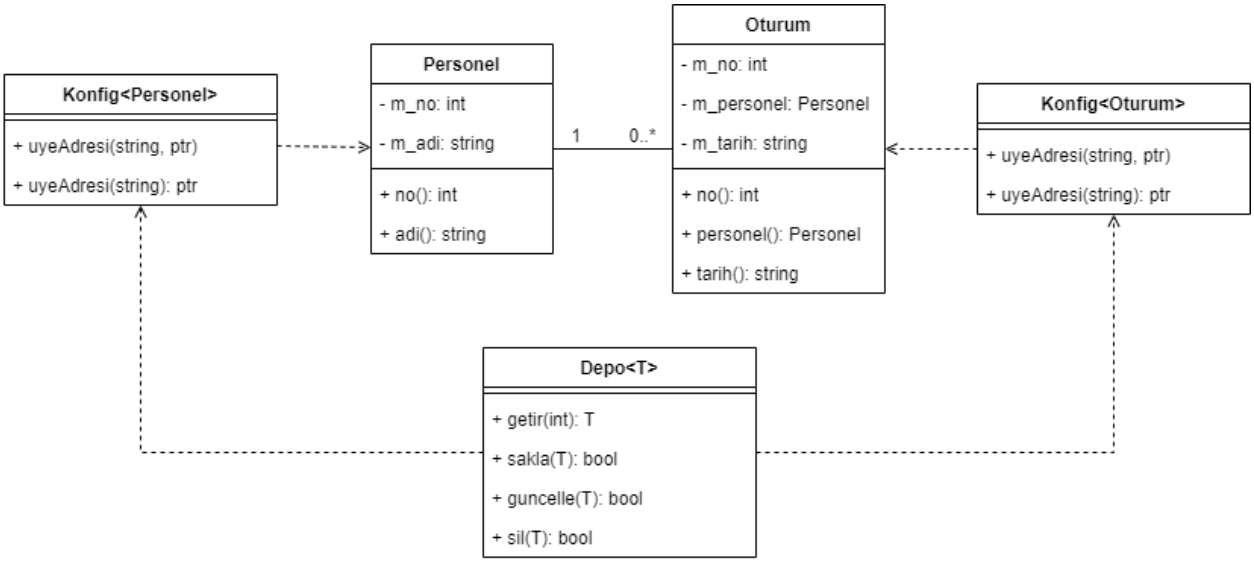
4. ÖNERİLEN YAKLAŞIM (PROPOSED APPROACH)

C++, veri tipi konusunda, sert kurallara sahip derlenmiş bir dildir. Yapıların türüyle ilgili bilgiler kaynak kodun içine yerleştirilmiştir. Derleme tamamlandığında, türlerle ilgili tüm bu ayrıntılar sonsuza kadar yok olur. C++, bu sınırlamanın üstesinden gelip kendisi için bir ORM çözümü uygulamamıza izin verecek ölçüde kullanılabilecek iki özelliğe sahiptir; şablonlar (templates) ve işaretçiler (pointers).

Şablonlar, C++ dilinde genel programlamanın temelini oluşturur. Derlenen dillerde, geliştirici değişkenlerin veri tiplerini açık bir şekilde belirlemelidir. Ancak, bazı veri yapıları ve algoritmalar parametre olarak gelen verinin türünden bağımsız olarak aynı işlevi yerine getirirler. Şablonların kullanılmadığı durumlarda aşırı yükleme ile bu işlem gerçekleştiriliri. Şablonlar, sadece sınıf veya işlevi veri tipi belirtilmeden, tipten soyutlanmış olarak tanımlamamızı sağlar. Bu sınıf veya işlevi kullanacak yazılımcı hangi veri türünde işlem yapacak ise o durumda veri tipine göre uyarlanmış olur. Bu şekilde türden bağımsız kod yazılabilmektedir. Herhangi bir türden bağımsız bir şekilde kod yazmak yararlı olduğu gibi, o tür hakkında bilgi depolamak da çok yararlıdır.

İşaretçi, değeri aynı türden başka bir değişkenin adresi olan bir değişkendir. İşaretçiler, ister küresel (global) ister bir sınıf içinde olsun, değişkenlere (attributes) ve benzer işlevlere (functions) işaret edebilir. Yani, temel olarak C++'da dört tür işaretçimiz var; değişken işaretçisi, işlev işaretçisi, sınıf değişkeni işaretçisi ve sınıf işlev işaretçisi.

Bunlar C++'da normal veri türleridir, bunları koleksiyonlarda saklayabilir, işlevlere iletebilirsiniz, vb. Ancak sınıf değişkeni işaretçisi türünün ilginç yönü, normal bir işaretçinin yaptığı gibi tam (absolute) adresi tutmamasıdır. Ancak değişkenin/işlevin sınıf düzeninde olduğu yerin göreli (relative) adresini tutar. Bu nedenle, bu tür işaretçiler için gerçek adresi, göreli adresi ile belirli bir sınıf nesnesinin başlangıç adresinden oluşur (yalnızca çalışma zamanı (run-time) ile nesnenin oluşturulduğunda). Bu nedenle sözdizimi, bir sınıf değişkeni/işlevi işaretçisini isterken, bir nesne seçmemizi de gerektirir. Dolayısıyla, C++ dili, sınıf üyesi işaretçisi için özel işleçler (operators) tanımlamıştır. .* ve ->* operatör işlevi, bir ifadenin sol tarafında temsil ettiği nesne için belirli sınıf üyesi değerleri döndürürken, sağ tarafı bir sınıf üyesini belirtir. Böylece, herhangi bir sınıf nesnesinin açık ara yüzünü (public interface), depolanmış üye işaretçileri aracılığıyla yönetebiliriz. Her sınıf için üye işaretçilerini depolamak için de şablonlar kullanırız. Şekil 3, bu yaklaşımın bir örneği için Class Diagram'ını göstermektedir.



Şekil 3. Uygulama sınıfları için meta-sınıf yaklaşımı

Şekil 3'deki yaklaşımın kod örneği ise, şu şekilde gösterilebilir:

```
Personel p = Depo<Personel>::getir(1);  // Seri numarası 1 olan personelin nesnesi veritabanındaki verilerle doldurulup getirilir
p.adi("Ahmet");  // Personelin adı değiştirilir
Depo<Personel>::guncelle(p);  // Nesnenin verileri veritabana yansıtılır
```

Verilen kodda verilen veritabanındaki *Personel* tablosundaki kayıt eşleştirilen sınıf objesine aktarılmaktadır. Sonraki satırda ilgili kaydın alanında güncelleme yapılarak *güncelle* fonksiyonu ile veritabanı değiştirilmiştir.

4. KULLANIM DURUMLARI (USE CASES)

4.1 Eski Yazılımlar

Bilgisayar teknolojisinin birkaç on yıl boyunca yaygın kullanımı, bazı büyük, karmaşık sistemlerin daha fazla değişiklik ve güncelleme sorunları ile karşılaşmasıyla sonuçlanmıştır. Bu eski bilgi sistemleri önemli sorunlar (kırılganlık, esneklik, yalıtım, genişletile bilirlik, açıklık eksikliği vb.) oluşturmasına rağmen, görev açısından kritik de olabilirler. Bu sistemlerden herhangi biri çalışmamaya başlarsa iş durma noktasına gelebilir. Bu nedenle birçok kuruluş için sistemin hizmetten çıkarılması bir seçenek değildir. Bu tür sistemler veri yapılarına her hangi bir değişiklik yapmadan, yukarıdaki önerdiğimiz yaklaşım ile sistemin veri katmanını oluşturabilir bunun yanında onu revize etme imkânı da meydana gelir. Böylece, müdahaleci olmayan bir ORM aracı sayesinde, sistemin ömrünü uzatmış oluruz, sistem sürdürülebilir olur. Örneğin, bir eski kurumsal bilgi sisteminin verilerini bir CSV dosyaya kayıt ediyor. Bu sistemin başka sistem ile entegrasyonu istendiğinde, önerdiğimiz ORM kütüphanesi ile sistemin veri yapı sınıflarını değiştirmeden diğer sistem ile veri ilişkilendirmeleri yapılıp uyum sağlanmış olur. Böylelikle, eski sistemin esnekliği artırılıp yapıları değişmeden güncellenmiş olur.

4.2 Çerçeveler

Yukarıda bahsedilen yöntem sadece ORM araçları için geçerli değil, farklı çerçeveler için de geçerlidir. Örneğin, bir UI çerçevesi için, veritabanındaki tablonun detayları nesne ortamında meta-nesne olarak kayıtlı olduğu için, generic tablo görüntüleyicisi geliştirmemize imkân verir. Ayrıca, öyle bir meta-nesne sistemi, bu tür çerçevelerde kullanıldığı takdirde, nesne/veri dönüşümü (serialization & deserialization) otomatik bir şekilde desteklenmiş olur. Örneğin, Person gibi bir sınıfın detaylarını ORM kütüphanemizde sakladığımızda, TableViewer<Person> gibi bir generic tablo görüntüleyicisi (template class), ve serialize<Person>() gibi bir generic serileştirme işlemi (template function) tanımlanabilir. Böylelikle, daha az kod ile birçok özellik elde edilir. Bunun ile birlikte, kodun testini ve bakımını daha kolay olur.

4.3 Web Uygulamaları

C++, diğer diller ile kıyas ettiğimizde, çok fazla web çerçeveleri yoktur. En temel nedenlerin arasında, standart C++ kütüphanelerinde yansıma özeliği bulunmamasıdır. Önerdiğimiz yöntem, bir web çerçevesinin sadece veri erişim katmanını değil, birçok hususta (serileştirme, görüntüleme, vb.) işlevselliğini daha kolay ve performanslı gerçekleştirmemize olanak sağlar. Geliştirilen yöntem veritabanı ile C++ objeleri arasında eşleştirme yaptığından arka plan kodlamayı kolaylaştırmaktadır. Böylelikle, önerilen yöntem bir web çerçevenin ekosistemini oluşturmaktadır. Aksi halde, web çerçevenin her bir katmanını (data, business, presentation) gerçekleştirmek için ayrı ayrı yardımcı yapılara ihtiyaç gerektirmektedir.

5. SONUÇ (CONCLUSION)

Bu çalışmada C++ programlama dili için veritabanı tabloları ile nesneler arası senkronizasyonu sağlayacak ORM kütüphanesi geliştirilmiştir. Kütüphanenin geliştirilmesi için C++' da kullanılan şablon programlama araçlarından faydalanılmıştır. Geliştirilen kütüphane ile veritabanı işlemleri için yazılması gereken kod miktarı oldukça azaltılabilecektir. Bununla birlikte yazılımcının veritabanı yönetim sistemleri farklılıkları ile ilgilenmesi ve SQL kullanım ihtiyacı ortadan kaldırılmaktadır. Bununla birlikte veritabanı uygulamaları için çalışma sürecinde test ve debug işlemlerini mümkün kılacak şekilde nesnelerin bellekte kalıcılığı sağlanabilecektir.

Önerilen ORM kütüphanesi, mevcut olan C++ ORM araçları gibi otomatik haritalama yeteneğine sahip olup nesnelerin veritabanında kalıcılığını ile CRUD işlemlerini yapabilir olması birlikte, teknoloji yığını bağımlılığı, kullanıcı yapılarını değiştirtme zorunluğu, makro kullanımı zorunluğu, ve kod üretme aracı kullanım zorunluğu sınırlamaların hiçbirine sahip değildir. Bununla birlikte daha iyi kullanıcı dostluğu, kod testi, kod bakımı ve performans sağlayabilir.

Bir sınıfın kalıcılığını desteklemek için yapılandırma esnasında üyelerinin detayları tekrarlanarak kayıt edilmektedir. Bu tekrar problemi mimarinin tek zayıf yönü sayılabilir.

ÇIKAR ÇATIŞMASI (CONFLICT OF INTEREST)

Makale yazarları aralarında herhangi bir çıkar çatışması bulunmamaktadır..

KAYNAKLAR (REFERENCES)

[1] S. Karacabey, "Veritabanı modelleri ve hiyerarşik veritabanından ilişkisel veritabanına dönüşüm", Yüksek Lisans Tezi, Yıldız Teknik Üniversitesi (1997).

[2] C. Ireland, D. Bowers, M. Newton, K. Waugh, "Understanding object-relational mapping: A framework based approach", International Journal on Advances in Software, vol 2 no 2&3, (2009).

[3] M. K. Awang, "Transforming object oriented data model to relational data model", International Journal of New Computer Architectures and their Applications, 3(3):403-410 (2012).

[4] C. Xia, G. Yu, M. Tang,."Efficient Implement of ORM (Object/Relational Mapping) Use in J2EE Framework: Hibernate", 1 - 3. 10.1109/CISE.2009.5365905 (2010).

[5] X. Zhang, "A framework for object-relational mapping with an example in C++", Masters thesis, Concordia University (2004).

[6] Code Synthesis, "ODB - C++ Object-Relational Mapping (ORM)", https://www.codesynthesis.com/products/odb/ (Accessed 2022-11-06).

[7] L. Marty, "QxOrm : C++ Qt ORM Object Relational Mapping database library - QxEntityEditor : C++ Qt entities graphic editor (data model designer and source code generator)", http://www.qxorm.com/qxorm_en/home.html/ (Accessed 2022-11-06).

[8] L. Stryzhevskyi, “Oat++”, https://oatpp.io/ (Accessed 2022-11-06).

[9] Emweb, “Wt, C++ Web Toolkit — Emweb”, https://www.webtoolkit.eu/wt/ (Accessed 2022-11-06).

[10] TreeFrog Framework Project, “TreeFrog Framework | High-speed C++ MVC Framework for Web Application”, https://www.treefrogframework.org/ (Accessed 2022-11-06).

[11] M. Garrigues, “Lithium C++ High Performance HTTP server”, https://matt-42.github.io/lithium/ (Accessed 2022-11-06).

[12] P. Korzhyk, “GitHub - paulftw/hiberlite: C++ ORM for SQLite”, https://github.com/paulftw/hiberlite/ (Accessed 2022-11-06).

[13] BrainlessLabs, “GitHub - BrainlessLabs/bun: Bun is a simple to use C++ Object Database, Object Relational Mapper (ORM) and key-value library”, https://github.com/BrainlessLabs/bun/ (Accessed 2022-11-06).