EINE OBJEKTORIENTIERTE SCHNITTSTELLE FÜR EIN ECHTZEIT-PROZEßDATENHALTUNGSSYSTEM

Prof. Dr. rer. nat. U. Brinkschulte*, Dipl. Inform H. Vogelsang
Universität Karlsruhe, Institut für Mikrorechner und Automation

Abstract

To aid the engineering of computer based automation systems, a relational real-time database system called Merlin has been developed and was successfully used in several industrial and scientific applications. In the last few years, object oriented design principles have been strongly introduced in the design of automation systems. So the idea was born to extend Merlin to allow object oriented real-time database management.
The basic idea to realize this extension is to create an interface to store objects in the existing relational real-time database system. This has two main advantages. Firstly, the database system itself must not be modified. Secondly, the interface can be used with other database systems as well. The task of this interface is to create images of objects in an application program and to store or load these images in or from the database. By this, persistent objects are created. As implementation language, C++ was selected.
To store objects in a relational database, objects's data structures must be stored in according database relations. Object's code sections do not need to be stored in the database, because they are already stored in the application program. But there are some problems to consider, such as: unique object identification, object references, multiple referenced objects and objects with complex data structures. To reflect this, a relational database scheme is created by the interface, which contains two administrative relations and one relation for each stored object class.
The interface itself consists of several C++ classes. They are used as basic classes for the persistent objects and provide database specific methods.
The described system has been tested in several application projects. It has proven to be very useful in object oriented programming. The real-time features of the underlying relational database system could be easily adapted to object oriented database management.
At the moment, the object oriented interface is under construction again. Several extensions are planed and partially realized, such as: a precompiler to ease the definition of persistent objects, object selection and object synchronization in distributed environment.

1. Einleitung

Die Entwicklung mikrorechnergestützter Automatisierungssysteme ist eines der Hauptforschungsgebiete des Instituts für Mikrorechner und Automation (IMA) an der Universität Karlsruhe. Um diesen Entwicklungsprozeß zu unterstützen, wurden am IMA eine Anzahl Grundkomponenten und Dienste entworfen und realisiert. Eine dieser Grundkomponenten ist das Echtzeit-Prozeßdatenhaltungssystem Merlin [1], welches eine einfache und effiziente Datenhaltung auch in zeitkritischen Automatisierungsaufgaben ermöglicht. Dieses System basiert auf relationalen Datenstrukturen und wurde bis heute mehrfach erfolgreich wissenschaftlich wie industriell eingesetzt (z.B. [2],[3]).
Da in letzter Zeit objektorientierte Entwurfs- und Entwicklungsmethoden im Bereich der Automatisierungssysteme verstärkt zum Einsatz kommen, wurde Merlin nun für objektorientierte Echtzeit-Datenhaltung erweitert.

2. Systemarchitektur

Grundlegende Idee für diese Erweiterung ist die Entwicklung einer Schnittstelle, welche es erlaubt, Objekte in dem vorhandenen relationalen Echtzeit-Prozeßdatenhaltungssystem zu speichern. Dies hat zwei Vorteile:

- Das Prozeßdatenhaltungssystem Merlin selbst muß nicht verändert werden.
- Die entwickelte Schnittstelle kann auch zusammen mit anderen Datenhaltungs- oder Datenbanksystemen verwendet werden.

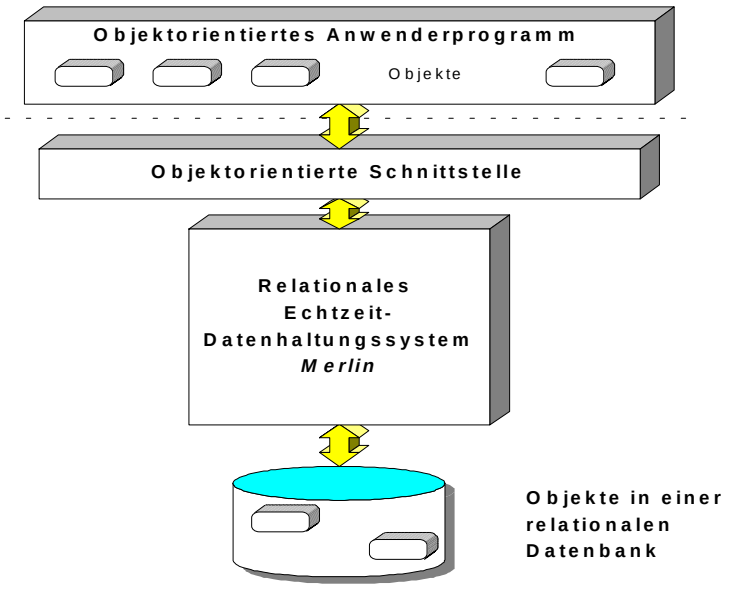

Bild 1: Systemarchitektur

Aufgabe der Schnittstelle ist es, die Objekte des Anwenderprogramms auf eine relationale Datenbankstruktur abzubilden. Hierdurch werden zum einen persistente Objekte ermöglicht, welche die Laufzeit des Anwenderprogramms überdauern. Zum anderen können die gespeicherten Objekte nach verschiedensten Suchkriterien wieder aus der Datenbank geladen werden.

Als objektorientierte Programmiersprache zur Einbettung und Realisierung der Schnittstelle wurde C++ verwendet. Dies hat zwei Gründe: Zum einen wurde Merlin selbst in C entwickelt. Zum anderen ist C++ eine sehr portable und hochverfügbare Sprache.

3. Speicherung von Objekten in einer relationalen Datenbasis

Um Objekte in einer relationalen Datenbankstruktur zu speichern, müssen die Datenstrukturen der Objekte in entsprechenden Relationen der Datenbank gespeichert werden. Die Objektmethoden müssen hingegen nicht gespeichert werden, da sie sich bereits dauerhaft im Anwenderprogramm befinden. Um ein Objekt wieder zu laden, müssen nur seine Datenstrukturen wieder geladen werden.
Es sind jedoch einige Probleme zu berücksichtigen (vgl. [4]):

- Objektidentifikation
  In C++ wird ein Objekt nur durch seine Speicheradresse identifiziert. Diese ist jedoch nur zur Laufzeit gültig. Um ein Objekt dauerhaft und eindeutig in der Datenbank zu identifizieren, muß die Schnittstelle deshalb einen eigenen eindeutigen Identifikationscode für jedes gespeicherte Objekt erzeugen.

- Objektverweise
  Objekte können mittels Zeigern auf andere Objekte verweisen. Diese Verweise müssen auch in der Datenbank aufrecht erhalten werden. Deshalb bildet die Schnittstelle solche Verweise auf den eindeutigen Identifikationscode der Objekte ab.

- Mehrfachreferenzen
  Jedes Objekt darf nur einmal aus der Datenbank geladen werden, auch wenn es mehrfach referenziert wird. Werden beispielsweise die Objekte A und B aus der Datenbank geholt, und beide Objekte referenzieren dasselbe Objekt C, so darf C nur einmal geladen werden. Um dies sicherzustellen, verwaltet die Schnittstelle deshalb bereits geladene Objekte in einer Hashtabelle.

- Komplexe Datenstrukturen
  Objekte können komplexe verschachtelte Datenstrukturen enthalten, welche nicht in einfachen normalisierten Relationen ablegbar sind. Um auch solche Objekte speichern zu können, werden die in Merlin vorhandenen unformatierten Langfelder (Binary Large Objects (Blobs) bis 4 Gbyte) benutzt.

Eine von der Schnittstelle erzeugte Objektdatenbank besitzt deshalb die in Bild 2 dargestellte Struktur.
Die Relation ‘Class Administration’ verwaltet die gespeicherten Objektklassen. Sie enthält die C++ Klassennamen (Classname) und deren interne eindeutige Identifikationen (Class Id). Eine solche Identifikation besteht aus der Nummer derjenigen Relation, in der alle Objekte der Klasse gespeichert sind. Weiterhin sind Typinformationen über die gespeicherten Klassen vorgesehen (Type Information), um eventuelle Änderungen der Klassendefinitionen im Anwenderprogramm zu entdecken.
Die Relation ‘Unused Object Id’s’ dient dazu, eindeutige Objektidentifikationen zu vergeben. Sie enthält für jede Klasse die z.B. durch Löschen freigegebenen Objektidentifikationen (Object Id). Diese bestehen aus eindeutigen Nummern. Soll ein neues Objekt gespeichert werden, so werden zunächst die dort gespeicherten freien Objektidentifikationen verwendet.

Erst wenn diese aufgebraucht sind, werden neue Identifikationen in Form der jeweils nächst größeren freien Nummer erzeugt.
Die eigentlichen Objekte werden in den darauf folgenden Relationen gespeichert. Hierzu erzeugt die Schnittstelle für jede in der Datenbank zu verwaltende Objektklasse eine eigene Relation, in der alle Objekte dieser Klasse abgelegt werden. Hierbei wird die Merlin-Eigenschaft genutzt, bis zu 2^{16} Relationen in einer Datenbank dynamisch in Echtzeit verwalten zu können. Jede dieser Relationen hat die Struktur der in Bild 2 dargestellten Relation 'Classes'. Sie enthält für jedes Objekt einer Klasse dessen eindeutige Objektidentifikation und dessen Datenstrukturen, die in einem unformatierten Langfeld (Data Structure) gespeichert werden. Zusätzlich können verschiedene Merkmale einer Klasse als optionale Suchattribute für assoziativen Objektzugriff ausgewählt und gespeichert werden (Optional Search Attributes).

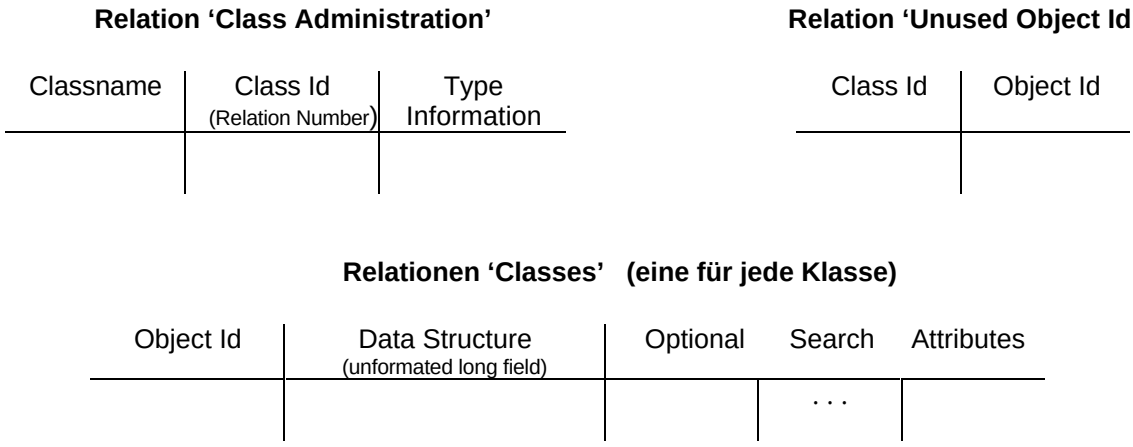

Bild 2: Struktur einer Objektdatenbank

4. Schnittstellenarchitektur

Die Schnittstelle selbst besteht aus einer Reihe von C++ Klassen. Diese dienen als Basisklassen für die zu speichernden Objekte und stellen die datenbankspezifischen Methoden zur Verfügung. Auf diese Weise wird eine einfache und natürliche Einbettung in die Sprache C++ erreicht. Bild 3 zeigt die Architektur der Schnittstelle.

Vier Basisklassen stehen dem Anwenderprogramm zur Verfügung: 'MetaTypeServer', 'Persistent', 'PersistentPtr' und 'AutoPersistentPtr'.
Die Klasse 'MetaTypeServer' dient dazu, die Schnittstelle über die Datenstrukturen und Datentypen der in der Datenbank zu speichernden Klassen zu informieren. Jede Klasse, von der Objekte in der Datenbank abgelegt werden sollen, muß über 'MetaTypeServer' durch folgende Informationen beschrieben werden:

- Klassenname;
- Datentyp und Größe jeder Klassenkomponente;
- Position jeder Klassenkomponente relativ zur Startadresse der Klasse.

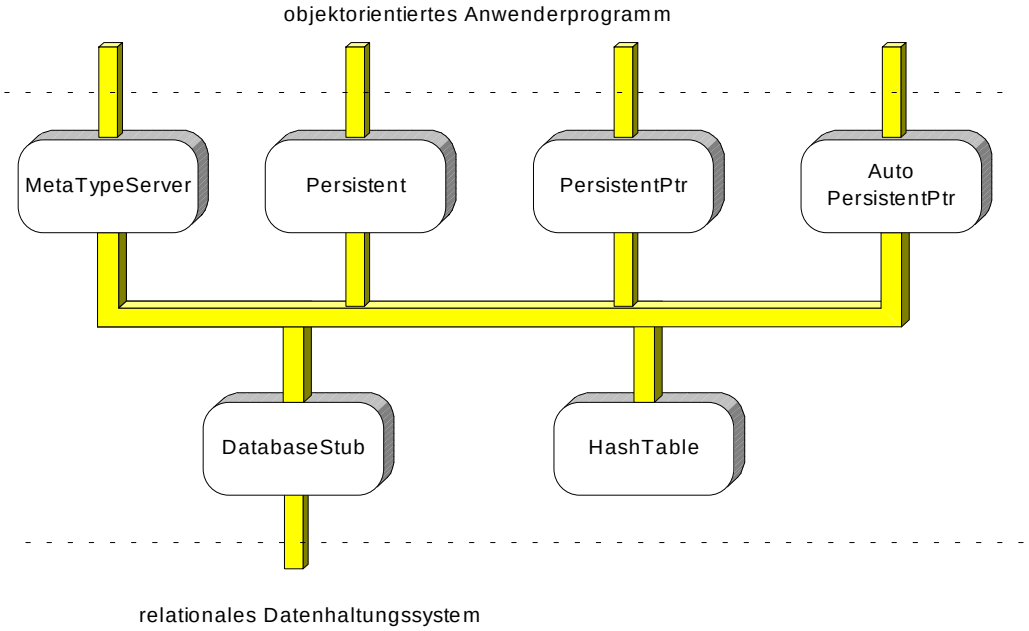

Bild 3: Schnittstellenarchitektur

Die Klasse ‘Persistent’ ist die Grundklasse der Schnittstelle. Jedes Objekt, welches in der Datenbank gespeichert werden soll, muß von dieser Klasse abgeleitet werden. Sie stellt Methoden zur Verfügung zum:

- Speichern von Objekten in der Datenbasis;
- Laden von Objekten aus der Datenbasis;
- Löschen von Objekten aus der Datenbasis;
- Holen der eindeutigen Objektidentifikation.

Um Referenzen zwischen Objekten auch in der Datenbasis zu wahren, stellt die Schnittstelle eine spezielle Zeigerklasse namens ‘PersistentPtr’ zur Verfügung. Zusätzlich zu Standard-Zeigern, die nur im Hauptspeicher Gültigkeit besitzen, verwaltet diese Klasse auch die eindeutige Objektidentifikation des referenzierten Objekts. Sie ist als Template Klasse realisiert. Durch die Überladung von Operatoren existieren keine syntaktischen Unterschiede zwischen Standard-Zeigern und den persistenten Zeigern. Es wurden lediglich einige Methoden hinzugefügt, um

- referenzierte Objekte aus der Datenbasis zu laden oder zumindest den Zeiger zu aktualisieren, falls das betreffende Objekt bereits geladen ist;
- referenzierte Objekte aus dem Hauptspeicher zu entfernen;
- Objektidentifikation und Hauptspeicheradresse eines referenzierten Objekts zu holen;
- den persistenten Zeiger auf ein anderes Objekt zu setzen.

Die Klasse ‘AutoPersistentPtr’ ist eine Erweiterung von ‘PersistentPtr’. Sie ist funktional identisch mit der einzigen Ausnahme, daß das referenzierte Objekt durch den Klassenkonstruktor automatisch aus der Datenbasis geladen wird.

Die verbleibenden zwei Klassen sind für das Anwenderprogramm unsichtbar. ‘HashTable’ enthält die bereits aus der Datenbasis geladenen Objekte, um mehrfaches Laden eines Objektes zu verhindern (siehe Abschnitt 3). Die Klasse ‘DataBaseStub’ schließlich stellt die Verbindung zwischen der Schnittstelle mit dem Datenhaltungssystem Merlin her.

5. Echtzeit-Prozeßdatenhaltungssystem

Das Prozeßdatenhaltungssystem Merlin wurde als einfaches und effizientes System zur Datenhaltung für Automatisierungsaufgaben entwickelt. Es besitzt im wesentlichen folgende Eigenschaften:

- relationale Datenstrukturen;
- eine einfache, aber leistungsfähige satzorientierte Schnittstelle zum Anwenderprogramm;
- Echtzeiteigenschaften wie hohe Verarbeitungsgeschwindigkeit, unterbrechungsfreier 24 Stunden Betrieb, bedingte zeitliche Vorhersagbarkeit;
- konfigurierbare Datensicherheit zum Schutz vor Datenverlust;
- lokaler Betrieb und verteilter Betrieb im Netzwerk;
- kompakter Aufbau (für den Einsatz auch auf kleinen Rechnersystemen);
- Portabilität und Plattformunabhängigkeit.

Für die Entwicklung der objektorientierte Schnittstelle waren darüber hinaus einige weitere Eigenschaften nützlich: Relationen einer Merlin-Datenbasis können lange unformatierte Felder (Blobs) bis zu 4 Gbyte Größe enthalten. Dadurch können auch Objekte mit sehr großen Datenstrukturen gespeichert werden. Weiterhin erlaubt Merlin die dynamische Erzeugung und Entfernung von Relationen zur Laufzeit. Eine Datenbasis kann bis zu 2^{16} Relationen enthalten. Die Verwaltung dieser Relationen erfolgt in Echtzeit ohne die Notwendigkeit einer Datenbasis-Reorganisation. Dies ermöglicht die Speicherung einer großen Anzahl verschiedener Klassen und Objekte.
Weitergehende Informationen über Eigenschaften, Schnittstellen und interne Realisierung von Merlin können z.B. in [1] und [5] nachgelesen werden.

6. Erfahrungen und künftige Erweiterungen

Die objektorientierte Schnittstelle für Merlin wurde bereits in einigen Anwendungen erprobt, z.B. in einem fahrerlosen Transportsystem oder einem objektorientierten Mensch-Machine-Dienst. Sie hat sich als sehr nützlich für die objektorientierten Programmierung erwiesen. Der Entwicklungs- und Wartungsaufwand konnte merklich reduziert werden. Die Wiederverwendbarkeit persistenter Objekte für andere Projekte ist ebenfalls leicht möglich. Daneben hat sich gezeigt, daß Merlin-Eigenschaften wie hohe Verarbeitungsgeschwindigkeit, unterbrechungsfreier 24h Betrieb, Datensicherheit, Zugriffsrechte, u.s.w. durch die entwickelte Schnittstelle einfach auf objektorientierte Datenhaltung übertragbar sind. Tabelle 1 zeigt z.B. die Ergebnisse einiger exemplarischer Zeitmessungen. Hierbei wurden jeweils 50000 Objekte (ca. 50 Byte Daten/Objekt) auf verschiedenen Hard- und Softwareplattformen in einer Datenbasis gespeichert, geladen oder gesucht.

Operation	Pentium PC, 90 Mhz MSDOS 6.22	Pentium PC, 90 Mhz LINUX 1.3.43	SUN Sparc 10 SunOS 4.1.3
Store	724	2941	1315
Load/Search	2173	3333	1851

Tabelle 1: Zeitmessungen von Merlin mit objektorientierter Schnittstelle, Speichern und Laden/Suchen von 50000 Objekten, Meßwerte in Operationen/Sekunde

Daneben sind einige Erweiterung der Schnittstelle in Arbeit:

- Objekt Selektion nach verschiedenen Kriterien
  Die Objekt Selektion, d.h. Suche nach verschiedenen Kriterien ist in der Datenbasisstruktur bereits realisiert (Optional Search Attributes), jedoch noch nicht in der Schnittstelle.

- Precomiler zur automatischen Generierung von MetaTypen
  Ein Precompiler, der die MetaTypen der persistenten Objekte automatisch aus dem Quellcode erzeugt, erleichtert erheblich die Definition persistenter Objekte und erkennt bzw. vermeidet Definitionsfehler.

- Synchronisation von Objekten in verteilten Anwendungen
  In verteilten Anwendungen ist ein Synchronisationsmechanismus für gemeinsam genutzte Objekte wichtig. Geplant ist hier, den in Merlin für Relationen vorhandenen Synchronisationsmechanismus über die Schnittstelle auf Objekte abzubilden.

Literatur

[1] U. Brinkschulte,"MERLIN-Ein Prozeßdatenhaltungssystem für Echtzeitanwendungen", In: Conference Proceedings, Echtzeit 93, Karlsruhe, 1993

[2] U. Brinkschulte, R. Schaeffer, "Nachrichten- und Archivsysteme - Innovation eines kleineren mittelständischen Unternehmens auf Basis der Informatik Systemtechnik" In: Conference Proceedings, 2. Beckmannkolloqium, Wismar, 1994

[3] U. Brinkschulte, M. Siormanolakis, H. Vogelsang, "Visualization and manipulation of structured information", In: Conference Proceedings, Visual 96, Melbourne, Australia 1996

[4] Artikelserie über persistente Objekte, In: Journal of object oriented programming : JOOP 95, SIGS Publ. New York, 1995

[5] U. Brinkschulte, "Architektur eines Datenhaltungsdienstes", In: Conference Proceedings, 39. Wissenschaftliches Kolloqium, TU Illmenau, 1994