PAUL SCHERRER INSTITUT

SWISS LIGHT SOURCE

MAKING A STATEMENT WITH CORBA

Michael Böge, Jan Chrin

Paul Scherrer Institute

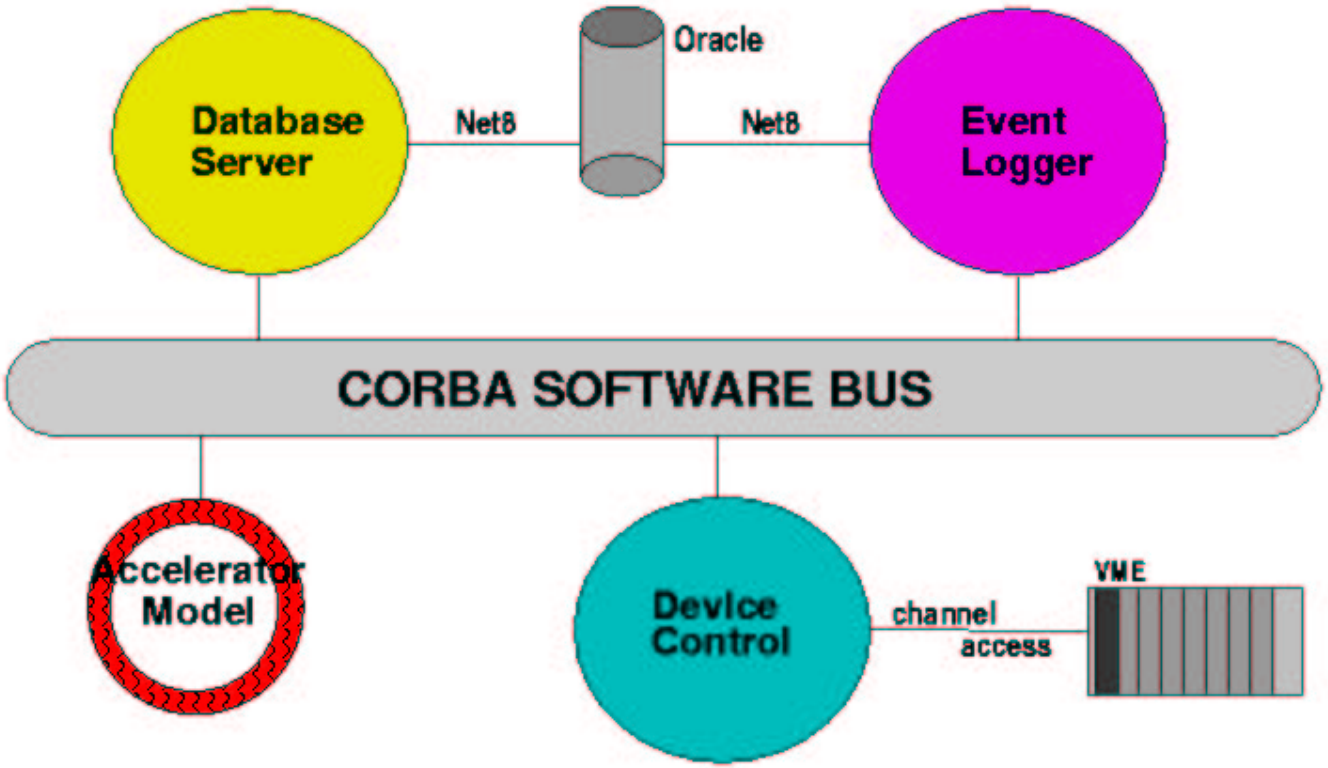
"SELECT * FROM DRINKS WHERE COFFEE=’CAPPUCCINO’"

Outline

- Introduction
  - Beam Dynamics Applications @ SLS
- CORBA Fundamentals
  - scalability of the CORBA database server
- Database Application Objects
  - database API
  - mapping of native database types to CORBA IDL types
  - object −relational mapping schemes
  - performance versus ease of use
- Present Direction & Summary

Introduction

Components Serving Beam Dynamics Applications



Introduction (cont.)

- CORBA interface to Oracle

CORBA API translates −
object constructs to relational constructs and communicates with the database using SQL

CORBA API provides −
fns for creating, retrieving, modifying objects and their attribute values in the relational db

e.g.

GetObjects(<Class>,<Attribute>,{<Result Set>});

DB Table(s) Column Name Object returned to caller

Advantage: db operations using oo methods; no SQL syntax required

CORBA Architecture

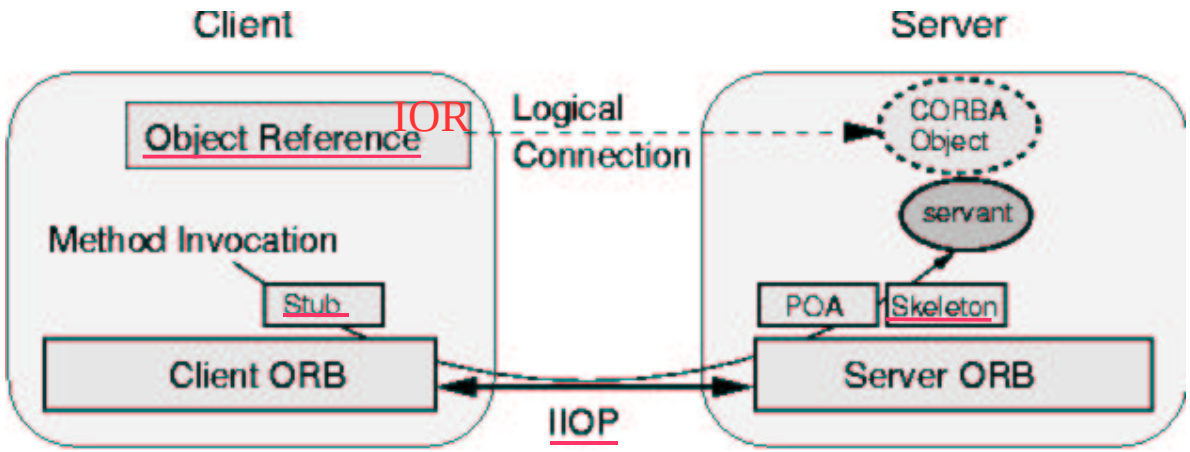
ORB Object Request Broker – facilitates communication between objects

IOR Interoperable Object Reference – reference to target object

IDL Interface Definition Language – specifies interface to CORBA object

IDL to C++, Java, Tcl

IIOP Internet Inter–ORB Protocol –communications protocol for interoperability



Portable Object Adapter

POA component visible to server only

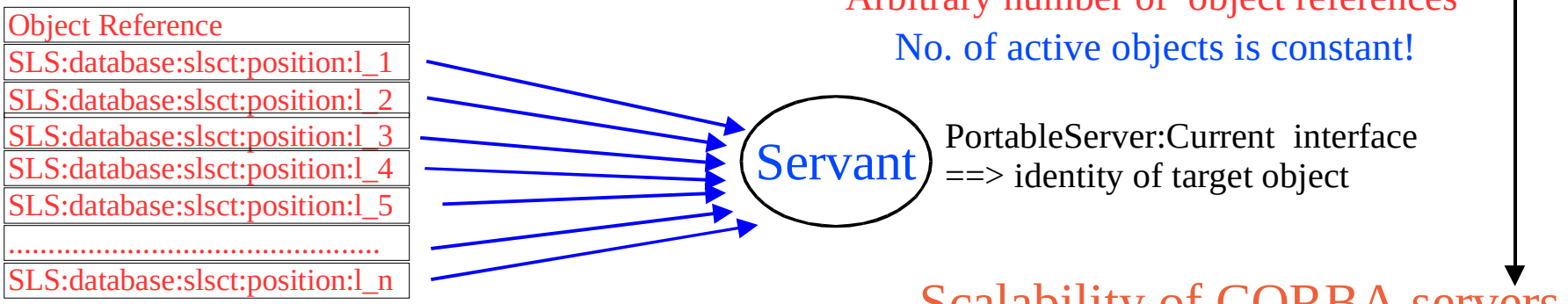
- generate/interpret object references
- demultiplex requests to map object references to servants

POA characteristics defined by POA policies, e.g. ...

- Lifespan Policy: TRANSIENT or PERSISTENT
- Request Processing Policy: SERVANT_MANAGER or DEFAULT SERVER

activates servants as they are required implementations for many objects

Arbitrary number of object references
No. of active objects is constant!



Database Application Objects

- Database API
- Mapping of −native database to IDL − datatypes
- Object to relational database mapping
- Performance

Database API

C++:
Oracle Template Library (OTL) + Oracle Call Interface (OCI)

http://otl.sourceforge.net/

OTL C++ classes:
- database connections and transaction management
- SQL statements and stored procedure execution
- exception handling
- operations with BLOBs and CLOBs

OTL templates are expanded into direct database API function calls
- high performance and reliability

Database Retrieval Timing Tests

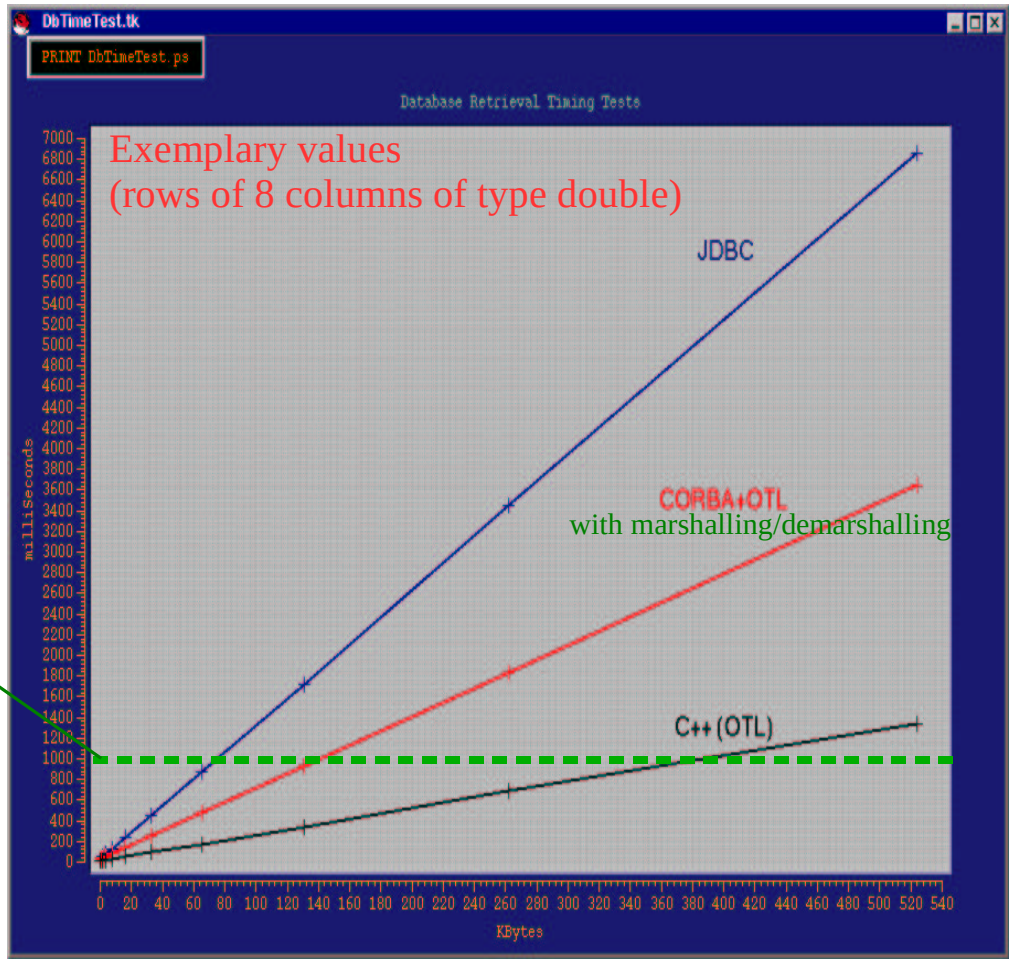
Throughput

C++(OTL): 400 KBytes/s

CORBA+OTL: 140 KBytes/s

JDBC: 70 KBytes/s

t(CORBA+OTL) = 0.5 (JDBC)



Mapping OTL datatypes to IDL basic types

- OTL defines datatypes that map to Oracle8 types

Recommended Conversions

OTL_VAR_	IDL_
CHAR	char
FLOAT	float
DOUBLE	double
INT	long
UNSIGNED INT	unsigned long
SHORT	short
LONG_INT	long
VARCHAR_LONG	string
BLOB	

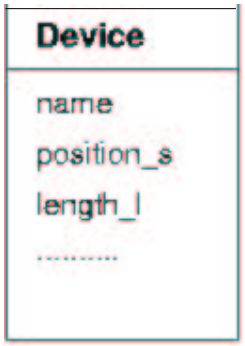
Column Data retrieved using native OTL type and converted to recommended (+other) IDL type

Metadata supplied to client

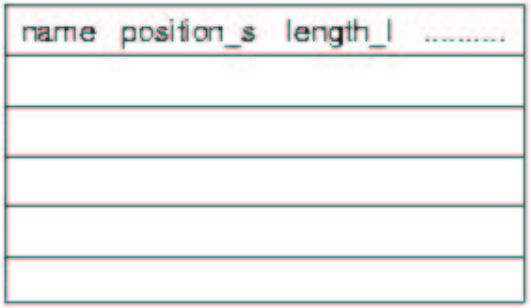
Data arrays (IDL sequences) mediated thru OTL types that map onto BLOBS

BLOBS are decomposed into sequences of the recommended IDL types

Basic Object Mapping Schemes



one object –> one table



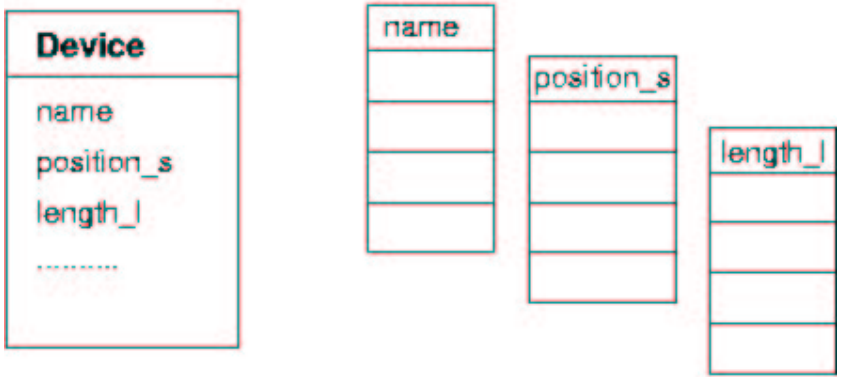
→ Attributes of one object map to columns of one table

Advantage: Simple + Best Performance

When objects retrieved from database − no joins required

When objects stored in database − one write operation/object

one object –> many tables



→ Attributes of one object map to multiple tables

When data for one object scattered across many tables

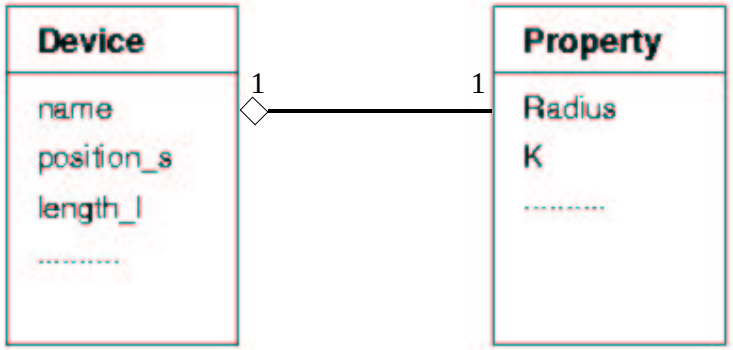
Advantage: Additional "columns" easily added as separate *new* tables

Disadvantage: database reads require join operations

− writes require separate insert/update operations

Basic Object Mapping Schemes (cont.)

many objects –> one table



→ multiple objects map to same row in table

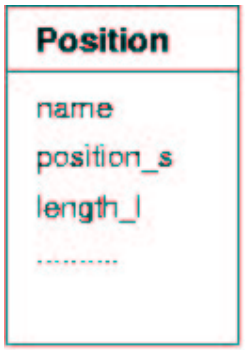
Scenario:
- dependent objects contained within primary object

Disadvantage: more object modelling required

DevicePropertyTable

name	position_s	length_l	.....	Radius	K	.....

Basic Object Mapping Schemes (cont.)



one object –> one table

Position Table

name	position_s	length_l	.....

→ Attributes of one object map to columns of one table

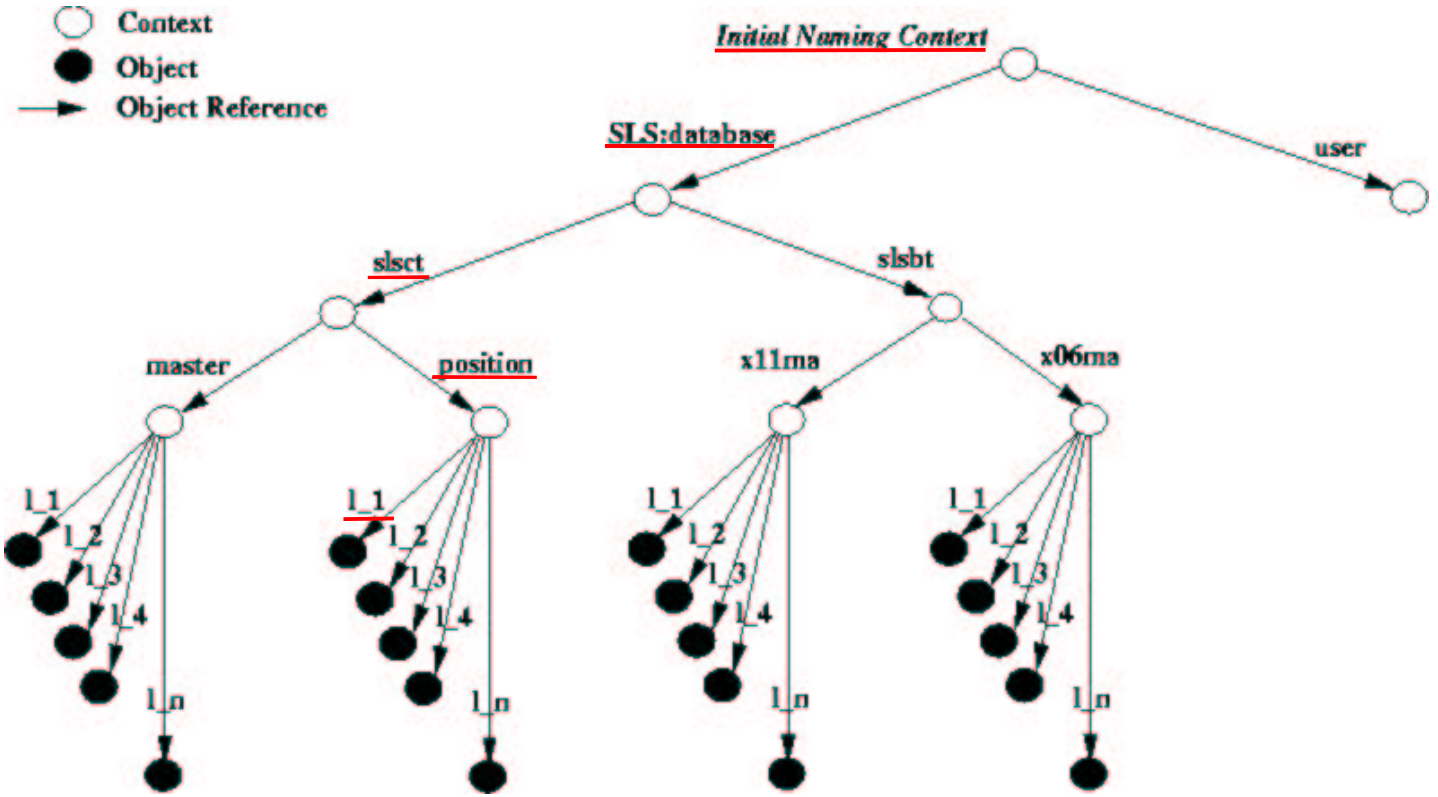
Table ⇄ Class

Row ⇄ Instant

Table Row/Line No.	Object Reference
Row_1	SLS:database:slsct:position:l_1
Row_2	SLS:database:slsct:position:l_2
Row_3	SLS:database:slsct:position:l_3
Row_4	SLS:database:slsct:position:l_4
....	....
Row_n	SLS:database:slsct:position:l_6

Naming Service

Naming Graph – hierarchy of bindings – name to reference association
contexts – objects that store bindings



Database Retrieval Operation

Server : Database application object contains the SQL to:
establish database connection, execute a SQL statement, retrieve data

Each table row is indexed using the primary key

Client: Database table/row retrieval thru RMI

Tcl:

Object Reference Operation Output (i.e. Column data)

```
$ObjectRefToTable($LineNo)  GetRow  rowData
```

Procedure to unfold Tcl list ($rowData)

```
$DisplayData  $rowData
```

Advantage: Simple & Generic

Usage: configuration + offline

Performance Considerations

Access to each single table row requires a RMI (not as fast as a local fn call!)

RMI limited by
call latency i.e. Message overhead (1 ms)
marshalling rate i.e. Rate at which ORB transmits/receives data (~500 KBytes/s)

Default Servant
Trade Off: Time required to look −up target object identity (PortableServer::Current)
versus space required for multiple servants

Call latency + target object ID lookup + database access

e.g. Table of 3300 rows − each row with 3 columns − of datatype union => 18s

Improved performance if entire Table data is returned in one call:

Object Reference Operation Output (i.e. Sequence of Rows)

Tcl:
$ObjectRefTo($Table) GetRows rowDataSeq => <2s

Direct SQL Statements

Tcl: $ObjectRef($DbInstant) SQLSelect "select * from myTable where ..." rowDataSeq

(Object Reference → $ObjectRef($DbInstant); Operation → SQLSelect; Input ("SQL Statement") → "select * from myTable where ..."; Output (i.e. Sequence of Rows) → rowDataSeq)

Tcl: $ObjectRef($DbInstant) SQL_Direct "insert into myTable values (...)"

(Object Reference → $ObjectRef($DbInstant); Operation → SQL_Direct; Input ("SQL Statement") → "insert into myTable values (...)")

DML Commands: INSERT, UPDATE, DELETE

DDL Commands: CREATE, DROP

---

usage: acquisition and analysis of data online

Purpose built application objects

For clients with particular needs

Methods perfom specific database operations
only acquiring data relevant to the application

C++

```
BdDbMagnet * magnetData;
Char * magnetType="bo";
```

```
DbRef−>getMagnetParameters(magnetType,  Data)
```

Present Direction

> 10^4 rows made available as CORBA objects

all tables can be made available − through small−scale code generation

Object by Value semantics

Value types declare both state members and operations/attributes

→ "Value objects"

CORBA interface return "result set" as in JDBC

Summary

- CORBA database application objects
  - translate object constructs to relational constructs
  - C++ Oracle Template Library
- Default Servants
  - support arbitrary number of objects in a fixed amount of memory
  - scalability of the database server
- Client database operations using object oriented methodology
  - configuration
  - acquisition and analysis of data online