Interprocedural Query Extraction for Transparent Persistence

Ben Wiedermann, Ali Ibrahim, and William R. Cook
The University of Texas at Austin
Enterprise Applications

Business Logic + Data Access
Enterprise Applications

Object-Oriented Programming Language + Relational Database Management System
Impedance Mismatch
Impedance Mismatch

Imperative vs Declarative
Impedance Mismatch

Object Graph vs Tables
Impedance Mismatch

Field Traversal vs Joins
Design Tradeoff

Abstraction vs Performance
Query Extraction

Abstraction + Performance
Example: Transparent Persistence

```java
void printOverLimit(int limit) {

    for (Employee emp : db.employees)
        if (emp.salary > limit)
            printEmployee(emp);

}

void printEmployee(Employee emp) {
    System.out.println(emp.name + ": " + emp.dept.title);
}
```
Example: Transparent Persistence

```java
void printOverLimit(int limit) {
    for (Employee emp : db.employees)
        if (emp.salary > limit)
            printEmployee(emp);
}

void printEmployee(Employee emp) {
    System.out.println(emp.name + " : " + emp.dept.title);
}
```
void printOverLimit(int limit) {
    for (Employee emp : db.employees)
        if (emp.salary > limit)
            printEmployee(emp);
}

void printEmployee(Employee emp) {
    System.out.println(emp.name + 
                        " : " + emp.dept.title);
}
Example: Transparent Persistence

```java
void printOverLimit(int limit) {
    for (Employee emp : db.employees)
        if (emp.salary > limit)
            printEmployee(emp);
}

void printEmployee(Employee emp) {
    System.out.println(emp.name + ": " + emp.dept.title);
}
```
Example: Transparent Persistence

```java
void printOverLimit(int limit) {
    for (Employee emp : db.employees)
        if (emp.salary > limit)
            printEmployee(emp);
}

void printEmployee(Employee emp) {
    System.out.println(emp.name + " : " + emp.dept.title);
}
```
Example: Transparent Persistence

```java
void printOverLimit(int limit) {
    for (Employee emp : db.employees) {
        if (emp.salary > limit) {
            printEmployee(emp);
        }
    }
}
```

```java
void printEmployee(Employee emp) {
    System.out.println(emp.name + "": " + emp.dept.title);
}
```
Example: Language-Integrated Queries

```csharp
void printOverLimit(DataContext db, int limit) {

    DataLoadOptions dlo = new DataLoadOptions();
    dlo.LoadWith<Employee>(e => e.dept);         // prefetch
    db.LoadOptions = dlo;

    var employees = from e in db.Employee
                     where e.salary > limit
                     select e;

    foreach (Employee emp in employees)        // query execution
        printEmployee(emp);
}

void printEmployee(Employee emp) {
    Console.WriteLine(emp.name +
                      " : " + emp.dept.title);
}
```
Example: Language-Integrated Queries

```csharp
void printOverLimit(DataContext db, int limit) {

    DataLoadOptions dlo = new DataLoadOptions();
    dlo.LoadWith<Employee>(e => e.dept);
    db.LoadOptions = dlo;

    var employees = from e in db.Employee
                     where e.salary > limit
                     select e;

    foreach (Employee emp in employees)
        printEmployee(emp);
}

void printEmployee(Employee emp) {
    Console.WriteLine(emp.name +": " + emp.dept.title);
}
```
Example: Language-Integrated Queries

```csharp
void printOverLimit(DataContext db, int limit) {
    DataLoadOptions dlo = new DataLoadOptions();
    dlo.LoadWith<Employee>(e => e.dept);
    db.LoadOptions = dlo;

    var employees = from e in db.Employee
                    where e.salary > limit
                    select e;

    foreach (Employee emp in employees)
        printEmployee(emp);
}

void printEmployee(Employee emp) {
    Console.WriteLine(emp.name +
        " : " + emp.dept.title);
}
```

Query optimization & Data prefetch
Example: Language-Integrated Queries

```csharp
void printOverLimit(DataContext db, int limit) {

    DataLoadOptions dlo = new DataLoadOptions();
    dlo.LoadWith<Employee>(e => e.dept);
    db.LoadOptions = dlo;

    var employees = from e in db.Employee
                     where e.salary > limit
                     select e;

    foreach (Employee emp in employees)
        printEmployee(emp);
}

void printEmployee(Employee emp) {
    Console.WriteLine(emp.name +
                      " : " + emp.dept.title);
}
```
Example: Language-Integrated Queries

```csharp
void printOverLimit(DataContext db, int limit) {

    DataLoadOptions dlo = new DataLoadOptions();
    dlo.LoadWith<Employee>(e => e.dept);
    db.LoadOptions = dlo;

    var employees = from e in db.Employee
                     where e.salary > limit
                     select e;

    foreach (Employee emp in employees)
        printEmployee(emp);
}

void printEmployee(Employee emp) {
    Console.WriteLine(emp.name + " : " + emp.dept.title);
}
```
Example: Language-Integrated Queries

```csharp
void printOverLimit(DataContext db, int limit) {
    DataLoadOptions dlo = new DataLoadOptions();
    dlo.LoadWith<Employee>(e => e.dept);
    db.LoadOptions = dlo;
    var employees = from e in db.Employee
                     where e.salary > limit
                     select e;
    foreach (Employee emp in employees)
        printEmployee(emp);
}
void printEmployee(Employee emp) {
    Console.WriteLine(emp.name + " : " + emp.dept.title);
}
```
transparent persistence

language-integrated queries

JDBC
transparent persistence

compiler

language-integrated queries

JDBC
transparent persistence

compiler

query extraction

language-integrated queries

JDBC

performance

abstraction
Java Program w/ Transparent Persistence

Static Analysis

Traversal Summary

Query Creation & Program Transformation

Object-Relational Mapping

Java Program w/ Explicit Queries
Java Program w/ Transparent Persistence → Static Analysis → Traversal Summary → Query Creation & Program Transformation → Java Program w/ Explicit Queries
void printOverLimit(int limit) {
    for (Employee emp : db.Employees)
        if (emp.salary > limit)
            System.out.println(emp.name + " : " + emp.dept.title);
}
void printOverLimit(int limit) {
    for (Employee emp : db.Employees)
        if (emp.salary > limit)
            System.out.println(emp.name + "": " + emp.dept.title);
}
Traversal Summary: Filtering

```java
void printOverLimit(int limit) {
    for (Employee emp : db.Employees)
        if (emp.salary > limit)
            System.out.println(emp.name + " : " + emp.dept.title);
}
```
Traversal Summary as Query

```java
void printOverLimit(int limit) {
    for (Employee emp : db'.Employees)
        if (emp.salary > limit)
            System.out.println(emp.name + " : " + emp.dept.title);
}
```
Interprocedural Analysis (g as consumer)

```c
void f() {
    o.g();
}

T g() {
    o.
}
```
void f( ) {
    o.g();
}

T g( ) {
}

Caller Prefetches Callee’s Data

• Analysis statically composes summaries, performs prefetch.
Interprocedural Analysis
(virtual g as consumer)

void f( ) {
    o.g();
}

T g( ) {
}

T g( ) {
}
Interprocedural Analysis (virtual \textit{g} as consumer)

```c
void f() {
    o.g();
}
```

Callee Loads Its Data

- Caller does not load callee’s data.
- Callee executes dynamic query, using parameter as root.

```c
T g() {
}
```
void f( ) {
  o = o.g( );
}

T g( ) {
  return o;
}
Interprocedural Analysis (g as producer)

void f( ) {
    o.g( );
}

Callee Prefetches Caller’s Data

1) Analysis statically modifies callsite & method, to include caller’s summary.

TS summary

T g( ) {
    return ;
}
Interprocedural Analysis (g as producer)

void f( ) {
   o = o.g();
}

T g( ) {
   return o;
}

Callee Prefetches Caller’s Data

1) Analysis statically modifies callsite & method, to include caller’s summary.

2) Callee dynamically composes summaries, performs prefetch.
Interprocedural Analysis (g as producer)

void f( ) {
    = o.g( );
}

Callee Prefetches Caller’s Data

1) Analysis statically modifies callsite & method, to include caller’s summary.

2) Callee dynamically composes summaries, performs prefetch.
Handled

• Flow-sensitive
• Read-only queries
• User-created collections
• Unnecessary null checks
• Recursive traversals

Not Handled

• (Partially) Context-insensitive
• Database updates
• Escaped database values
• Aggregation / grouping / sorting
• Dynamic class loading / reflection
### Implementation
- JastAdd
- Java + Hibernate

### Server
- 2.4 GHz Intel Pentium 4
- 1 GB RAM
- Linux 2.6.22
- PostgreSQL 8.2.6

### Client
- 2x 3.0 GHz Intel Pentium-D
- 2 GB RAM
- Linux 2.6.22
- Sun HotSpot JVM 1.5.0

### Benchmarks

<table>
<thead>
<tr>
<th>TORPEDO</th>
<th>OO7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>Use Cases</td>
</tr>
<tr>
<td>40 objects</td>
<td>6 read-only queries</td>
</tr>
<tr>
<td>41,000 objects</td>
<td>7 read-only queries</td>
</tr>
<tr>
<td></td>
<td>4 graph traversals</td>
</tr>
</tbody>
</table>

- **TORPEDO**
  - 40 objects
  - 6 read-only queries
  - Hand-Optimized
  - Transparent
  - Query Extracted

- **OO7**
  - 41,000 objects
  - 7 read-only queries
  - Hand-Optimized
  - Transparent
  - Query Extracted
Can Static Analysis Extract Queries? Yes.

**TORPEDO**

<table>
<thead>
<tr>
<th>Query Type</th>
<th>Hand-Optimized</th>
<th>Query Extracted</th>
<th>Transparent</th>
</tr>
</thead>
<tbody>
<tr>
<td>List Auction Twice Without Transaction</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>List Auction Twice With Transaction</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>List Partial Auction</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>List Auction</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Find High Bids</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Find All Auctions</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**OO7 Queries**

<table>
<thead>
<tr>
<th>Query</th>
<th>Hand-Optimized</th>
<th>Query Extracted</th>
<th>Transparent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Query 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Query 3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Query 4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Query 5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Query 7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Query 8</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Can Static Analysis Extract Queries?

Yes.

TORPEDO
# of Queries Executed

List Auction Twice Without Transaction
List Auction Twice With Transaction
List Partial Auction
List Auction
Find High Bids
Find All Auctions

OO7 Queries
# of Queries Executed

Query 1
Query 2
Query 3
Query 4
Query 5
Query 7
Query 8

Ad-hoc Join

Hand-Optimized
Query Extracted
Transparent
Does Prefetch Help Performance?

Sometimes.

---

**TORPEDO Execution Time (s)**

- List Auction Twice Without Transaction
- List Auction Twice With Transaction
- List Partial Auction
- List Auction
- Find High Bids
- Find All Auctions

**OO7 Queries Execution Time (ms)**

- Query 1
- Query 2
- Query 3
- Query 4
- Query 5
- Query 6
- Query 7
- Query 8
Does Prefetch Help Performance?

Sometimes.

**TORPEDO Execution Time (s)**

- List Auction Twice Without Transaction
- List Auction Twice With Transaction
- List Partial Auction
- List Auction
- Find High Bids
- Find All Auctions

**OO7 Queries Execution Time (ms)**

- Query 1
- Query 2
- Query 3
- Query 4
- Query 5
- Query 7
- Query 8

Legend:
- Hand-Optimized
- Query Extracted
- Transparent
Does Prefetch Help Performance? 

Sometimes.

**Context Sensitive**
- List Auction Twice Without Transaction
- List Auction Twice With Transaction
- List Partial Auction
- List Auction
- Find High Bids
- Find All Auctions

**Aggregation**
- TORPEDO Execution Time (s)

**Ad-hoc Join**
- OO7 Queries Execution Time (ms)

**Query 1**
- Hand-Optimized
- Query Extracted
- Transparent

**Query 2**
- Hand-Optimized
- Query Extracted
- Transparent

**Query 3**
- Hand-Optimized
- Query Extracted
- Transparent

**Query 4**
- Hand-Optimized
- Query Extracted
- Transparent

**Query 5**
- Hand-Optimized
- Query Extracted
- Transparent

**Query 7**
- Hand-Optimized
- Query Extracted
- Transparent

**Query 8**
- Hand-Optimized
- Query Extracted
- Transparent

*Wiedermann, Ibrahim, Cook - OOPSLA 2008*
Related Work

Orthogonal Persistence
- Atkinson & Morrison [1995]
- Moss & Hosking [1991]

Analysis for Queries
- Type-checking SQL
  - Wassermann et al. [2007]; Tatlock et al. [2008] (QUAIL)
- Dynamic prefetching for persistence
  - Bernstein, Pal, Shutt [1999]; Han, Moon, Whang [2003] (PrefetchGuide); Ibrahim & Cook [2006] (AUTOFETCH)

First-Class Queries
- Wong [2000] (Kleisli)
- Bierman, Meijer, Schulte [2005] (C#, LINQ)
- Willis, Pearce, Noble [2006] (JQL)
- Cooper, et al [2006] (LINKS)
Query extraction is a novel static analysis for transparent persistence that supports recursive traversals.

Evaluation shows that query extraction can improve the performance of transparent persistence by at least an order of magnitude in many cases.

Results can guide ongoing programming language design.