The Fénix Software Architecture (v2)

João Cachopo
It all started back in 2003...
Problems with Fénix

- Performance
- Data inconsistencies
- Slow development
A New Architecture for Fenix

João Cachopo

October 26, 2004
The Fénix system is a rather successful application!

- It started small but grew very rapidly
- It features a significant amount of functionality
- It is in a stable point of development
- It has a well defined architecture

and, most importantly, it has a great team.
Make it even better!

Propose a new architecture that:

- Simplifies the programming model
- Improves performance
- Makes the system more modular

This presentation will sketch it, but it is not finished yet...
Implementing architecture v2
Implemented directly in the Fénix code base
First deploy on September 2005
Code implementing the new architecture extracted in 2008

The Fénix Framework
Fénix Framework
Strict Serializability is Harmless!

Domain Modelling Language
Model the structure of your domain entities with a lightweight Java-like domain-specific language and FF automatically generates all the support code.

Strong Consistency
Use a Software Transactional Memory to provide Strict Serializability for Enterprise Applications.

Transparent Persistence
Application state is automatically transferred to/from the underlying storage system.

Welcome to the Fénix Framework project

Fénix Framework allows the development of Java-based applications that need a transactional and persistent domain model. Even though it was originally created to support the development of web applications, it may be used to develop any other kind of application that needs to keep a persistent set of data.

One of the major design goals underlying the development of the Fénix Framework is that it should provide a natural programming model to programmers used to develop plain Java programs without persistence. Unfortunately, the addition of persistence, typically backed up by a relational database management system, interferes with the normal coding patterns of object-oriented programming, because
Continues to evolve...
Strict Serializability is Harmless: A New Architecture for Enterprise Applications

Sérgio Fernandes and João Cachopo

October 25, 2011, Portland, Oregon, USA
Fénix is a typical web-application, with a 3-tier architecture: Presentation, Business and Persistence.
Not so long ago
Not so long ago

User Interface

Client

Application logic
Transactions
Persistence

RDBMS
More clients

User Interface

Client

Application logic
Transactions
Persistence

RDBMS
More clients

User Interface

Client

Application logic

Transactions

Persistence

RDBMS
With the web

User Interface

Client

Client

Application logic

App Server

Transactions Persistence

RDBMS
Fénix is a typical web-application, with a 3-tier architecture: Presentation, Business and Persistence.
Fénix is a typical web-application, with a 3-tier architecture: Presentation, Business and Persistence.

The Business Tier has a layered architecture:
- The Presentation Logic Layer (Struts)
- The Service Layer
- The Domain Layer
- The Data Access Layer (DAOs + OJB)
<table>
<thead>
<tr>
<th>Presentation Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Services</td>
</tr>
<tr>
<td>OJB + DAOs</td>
</tr>
</tbody>
</table>
First problem:
Slow development
Developing new functionality

- New domain entity:
  - New table in the relational database
  - New domain class
  - Mapping between table and class
  - New DAO (interface + concrete class)

- New service:
  - New service class
  - Configure Berserk
  - New DTOs

- New action and JSP
Developing new functionality

- New domain entity:
  - New table in the relational database
  - New domain class
  - Mapping between table and class
  - New DAO (interface + concrete class)

- New service:
  - New service class
  - Configure Berserk
  - New DTOs

- New action and JSP
Developing new functionality

- New domain entity:
  - New table in the relational database
  - New domain class
  - Mapping between table and class
  - New DAO (interface + concrete class)

- New service:
  - New service class
  - Configure Berserk
  - New DTOs

- New action and JSP
The current architecture is well defined and has been taught to students.

But...

- There is a lot to write, even for simple things
- Simple services are hard to read and write
- Dealing with object locks is error-prone
- A service that writes cannot read
- We can’t reuse services easily
- Much of the business logic is on the DAOs
Take home message:
Lots of work!
The Anemic Domain Model

- Domain objects don’t exhibit behavior.
- All behavior is at the service level.
- Services are the result of a functional decomposition, not an object-oriented decomposition.
- Services deal with several objects.
- Object-specific behavior is replicated among services.

Some statistics: 230 domain classes versus 700 services...
Second problem: Performance
Performance problems

On moderate demand, the system gets really slow.

Why?
- Accessing the DAOs (in each service) hits the database
Presentation Logic

<table>
<thead>
<tr>
<th>Services</th>
<th>Views</th>
</tr>
</thead>
<tbody>
<tr>
<td>OJB + DAOs</td>
<td>Anemic Domain</td>
</tr>
</tbody>
</table>
OID: 324
name: John
contact: ...
Application logic

User Interface

Client

Client

App Server

Transactions Persistence

RDBMS
User Interface

Client

Client

Application logic

Simple

App Server

Transactions

Persistence

RDBMS
User Interface

Client

Client

Application logic

Simple

App Server

Transactions Persistence

RDBMS

1 request

10 round-trips
The diagram shows a system architecture with the following components:

- **User Interface**: Client
- **Application Logic**: App Server
- **Transactions Persistence**: RDBMS

The diagram also highlights the following:

- **Latency**: From User Interface to App Server, and from App Server to RDBMS.
Application logic

User Interface
- Client
- Client

App Server
- Complex

Transactions Persistence
- RDBMS
User Interface

Client

Client

Application logic

Complex

App Server

1 request

1000 round-trips

Transactions

Persistence

RDBMS
User Interface

Client

Client

Application logic

App Server

Transactions Persistence

RDBMS

Latency

Complex

Latency
How to solve this?
Caching

User Interface

Client

Client

Application logic

App Server

Cache

RDBMS

Transactions Persistence
“[cache-based] application server implementations are incorrect when used with databases providing snapshot isolation.”

-- Perez-Sorrosal, Middleware’07
Performance problems

On moderate demand, the system gets really slow.

Why?
- Accessing the DAOs (in each service) hits the database
Performance problems

On moderate demand, the system gets really slow.

Why?

- Accessing the DAOs (in each service) hits the database
- To guarantee the isolation object-level locks are used
- Each request needs to copy data back and forth via the DTOs
- Many services are poorly written, performance-wise, because:
  - Generate lots of garbage
  - Misuse tools like Collections.intersection()
  - Don’t make use of indexing on the data
Third problem: Data inconsistencies
Performance problems

On moderate demand, the system gets really slow.

Why?

- To guarantee the isolation object-level locks are used
Outline

1 Introduction

2 Goal

3 The Current Architecture

4 The New Architecture
   - The Premises
   - The Main Ideas
   - Implementing it

5 Future Work
Web-applications are special

Some assumptions regarding Fénix:

- There are much more reads than writes (10 to 1, 100 to 1, more?)
- Most of the data, if not all, fits in memory
- It is easier and better to program in the OO-model
- We have resources to record all the changes made to the system
How to improve development

To improve development:

- Simplify the model, by providing support for essential tasks
- Reduce the code to write
- Make the business logic reusable
- Eliminate bookkeeping code
How to control the performance

To control the performance:
- Try to do all of the work in memory
- Avoid generating unnecessary garbage
- Make the reads as fast as possible (avoid synchronization)
- Don’t bother with the write performance
The new architecture is not yet completely specified. Yet, I have some ideas on what to do:

- Eliminate all DAOs (but one)
- Make the Domain Layer **thicker** and the Service Layer **thinner**
- Use Versioned Objects to model the domain
- Change the Concurrency Control Mechanism
- Eliminate the DTOs
- Specify the access control at the object (or property) level
- Give each domain object one or more presentation type
The new architecture is not yet completely specified.

Yet, I have some ideas on what to do:

- Eliminate all DAOs (but one)
- Make the Domain Layer **thicker** and the Service Layer **thinner**
- Use Versioned Objects to model the domain
- Change the Concurrency Control Mechanism
- Eliminate the DTOs
- Specify the access control at the object (or property) level
- Give each domain object one or more presentation type
The new architecture is not yet completely specified.

Yet, I have some ideas on what to do:

- Eliminate all DAOs (but one)
- Make the Domain Layer **thicker** and the Service Layer **thinner**
- Use Versioned Objects to model the domain
- Change the Concurrency Control Mechanism
- Eliminate the DTOs
- Specify the access control at the object (or property) level
- Give each domain object one or more presentation type
The new architecture is not yet completely specified.

Yet, I have some ideas on what to do:

- Eliminate all DAOs (but one)
- Make the Domain Layer **thicker** and the Service Layer **thinner**
- Use **Versioned Objects** to model the domain
- Change the Concurrency Control Mechanism
- Eliminate the DTOs
- Specify the access control at the object (or property) level
- Give each domain object one or more presentation type
The new architecture is not yet completely specified. Yet, I have some ideas on what to do:

- Eliminate all DAOs (but one)
- Make the Domain Layer **thicker** and the Service Layer **thinner**
- Use Versioned Objects to model the domain
- Change the **Concurrency Control Mechanism**
- Eliminate the DTOs
- Specify the access control at the object (or property) level
- Give each domain object one or more presentation type
The new architecture is not yet completely specified.

Yet, I have some ideas on what to do:

- Eliminate all DAOs (but one)
- Make the Domain Layer **thicker** and the Service Layer **thinner**
- Use Versioned Objects to model the domain
- Change the Concurrency Control Mechanism
- Eliminate the DTOs
- Specify the access control at the object (or property) level
- Give each domain object one or more presentation type
The new architecture is not yet completely specified.

Yet, I have some ideas on what to do:

- Eliminate all DAOs (but one)
- Make the Domain Layer **thicker** and the Service Layer **thinner**
- Use Versioned Objects to model the domain
- Change the Concurrency Control Mechanism
- Eliminate the DTOs
- Specify the access control at the object (or property) level
- Give each domain object one or more presentation type
The architectural changes

The new architecture is not yet completely specified.

Yet, I have some ideas on what to do:

- Eliminate all DAOs (but one)
- Make the Domain Layer **thicker** and the Service Layer **thinner**
- Use Versioned Objects to model the domain
- Change the Concurrency Control Mechanism
- Eliminate the DTOs
- Specify the access control at the object (or property) level
- Give each domain object one or more **presentation type**
Outline

1. Introduction
2. Goal
3. The Current Architecture
   - Overview
   - Current Code Samples
   - Evaluation
4. The New Architecture
   - The Premises
   - The Main Ideas
   - Implementing it
5. Future Work
   - Finishing the architecture
   - Implementing the architecture
The new architecture is not yet completely specified.

Yet, I have some ideas on what to do:

- Eliminate all DAOs (but one)
- Make the Domain Layer **thicker** and the Service Layer **thinner**
- Use Versioned Objects to model the domain
- Change the Concurrency Control Mechanism
- Eliminate the DTOs
- Specify the access control at the object (or property) level
- Give each domain object one or more presentation type
The new architecture is not yet completely specified.

Yet, I have some ideas on what to do:

- Eliminate all DAOs (but one)
- Make the Domain Layer **thicker** and the Service Layer **thinner**
- Use Versioned Objects to model the domain
- Change the Concurrency Control Mechanism
- Eliminate the DTOs
The new architecture is not yet completely specified. Yet, I have some ideas on what to do:

- Eliminate all DAOs (but one)
- Eliminate the DTOs
Presentation Logic

Services          Views

OJB + DAOs    Anemic Domain
Presentation Logic

Services

_views

OJB + DAOs

Anemic Domain
The new architecture is not yet completely specified.
Yet, I have some ideas on what to do:

- Make the Domain Layer **thicker** and the Service Layer **thinner**
Presentation Logic

Services

OJB + DAOs

Anemic Domain

Views
Presentation Logic

Rich Domain Model
The new architecture is not yet completely specified.

Yet, I have some ideas on what to do:

- Use Versioned Objects to model the domain
- Change the Concurrency Control Mechanism
The Versioned Objects Model

The most important change is in the Domain Model:

- Domain objects have versions
- A change in an object makes a new version of that object
- All versions are kept
- All changes within the same transaction are for the same new version
- Each write transaction advances the world state one version, modifying possibly many objects
- Objects are read in a particular world state

Result

There is no shared mutable state
Implementing Versioned Objects in Java requires special care on the definition of domain classes.
Versioned Objects

class Person {
    String name;
    Address contact;

    String getName() { return this.name; }
    void setName(String name) { this.name = name; }
    ...
}

class Person {
    final VBox<String> name = new VBox<String>();
    final VBox<Address> contact = new VBox<Address>();

    String getName() { return this.name.get(); }
    void setName(String name) { this.name.put(name); }

    ...
}

Relations are much more complex...
Implementing Versioned Objects in Java requires special care on the definition of domain classes.
Implementing Versioned Objects in Java requires special care on the definition of domain classes.

To simplify the task and minimize errors, use a Domain Modelling Language:

- Allow the specification of entities
- Allow the specification of relations
- A compiler generates all the Java code
Transactional Domain Objects

```java
class Person {
    final VBox<String> name = new VBox<String>();
    final VBox<Address> contact = new VBox<Address>();

    String getName() { return this.name.get(); }
    void setName(String name) { this.name.put(name); }
    ...
}
```

```java
class Person {
    String name;
}
```

```java
relation PersonHasAddress {
    Address playsRole contact;
    Person playsRole people { multiplicity *; }
}
```
Domain Model

Structure + Behavior + Constraints

DML
Java + Atomic actions
Consistency predicates
Presentation Logic

Transactional + Persistent
Rich Domain Model
New Architecture

User Interface

Client

Client

Application logic

Transactions

App Server

Persistence

RDBMS
Persistence completely transparent

No RDBMS dependencies
### Transactional Domain Objects

<table>
<thead>
<tr>
<th>Person</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OID:</strong> 324</td>
</tr>
<tr>
<td><strong>name:</strong> John</td>
</tr>
<tr>
<td><strong>contact:</strong> ...</td>
</tr>
</tbody>
</table>
Transactional Domain Objects

Person

- OID: 324
- name: John
- contact: ...

Tx1
Transactional Domain Objects

Person

- OID: 324
- name: John
- contact: ...

Person

- OID: 324
- name: John
- contact: ...

Tx1 Tx2
Transactional Domain Objects

STM protected

Person
- OID: 324
- name: John
- contact: ...

Person
- OID: 324
- name: John
- contact: ...

Tx1 Tx2
Transactional Domain Objects

STM protected

Person

<table>
<thead>
<tr>
<th>OID: 324</th>
</tr>
</thead>
<tbody>
<tr>
<td>name: John</td>
</tr>
<tr>
<td>contact: ...</td>
</tr>
</tbody>
</table>

Tx1 Tx2
Transactional Domain Objects

STM protected

Person

<table>
<thead>
<tr>
<th>OID: 324</th>
</tr>
</thead>
<tbody>
<tr>
<td>name: John</td>
</tr>
<tr>
<td>contact: ...</td>
</tr>
</tbody>
</table>

Tx1 Tx2
Domain Object Cache

Person

OID: 324
name: John
contact: ...

Tx1 Tx2
Domain Object Cache

Cache

Person

OID: 324
name: John
contact: ...

Tx1 Tx2
Domain Object Cache

• Domain objects have identity

Cache
324 SoftRef()

Person
OID: 324
name: John
contact: ...

Tx1 Tx2
• Domain objects have identity
• DB round-trips reduced
JVSTM
Software Transactional Memory as a Java library
First multi-version STM

Designed for very large transactions and high read/write ratio
Key features

- Read-only transactions never conflict
- Commit-time locking (for R/W only)
- Mechanisms for reducing conflicts
Welcome to the JVSTM project

The JVSTM (Java Versioned STM) is a Java library implementing our approach to STM (Software Transactional Memory), which introduces the concept of versioned boxes.

The Java Versioned Software Transactional Memory (JVSTM) is a pure Java library implementing an STM. JVSTM introduces the concept of versioned boxes, which are transactional locations that may be read and written during transactions, much in the same way of other STMs, except that they keep the history of values written to them by any committed transaction.


Fénix was the first real-world application of STMs
Extracting the Fénix Framework
<table>
<thead>
<tr>
<th>Presentation Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transactional + Persistent</td>
</tr>
<tr>
<td>Rich Domain Model</td>
</tr>
</tbody>
</table>
Presentation Logic

DML + Java + JVSTM
<table>
<thead>
<tr>
<th>Presentation Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>DML + Java</td>
</tr>
<tr>
<td>DML Compiler + JVSTM</td>
</tr>
</tbody>
</table>
Fénix Framework

Presentation Logic

DML + Java

Fénix Framework
Presentation Logic

Application Domain

Fénix Framework
Fénix Application

Fénix Framework
Other Applications

Fénix Framework
Projectos / Fénix - Sistema de Informação Académica

O projecto tem 84 sugestões

**Suporte a Flash Video** Novo

#59 | Por: João Manuel Brisson Lopes em: 30/09/2009 | 2 comentários
Existem já bastantes disciplinas com elementos de estudo / demonstração em vídeo. O formato flash permite que esses vídeos sejam de pequeno tamanho e o seu streaming, o que é mais eficiente do q ...

**Pautas de Avaliação** Novo

#43 | Por: Alexandre Jorge Salgueiro Chambel em: 08/07/2009 | 6 comentários
As páginas com as pautas de avaliação das cadeiras só deveriam estar disponíveis para pessoas pertencentes ao IST, ou seja, deveriam pedir autenticação.

**Votar Online para a AEIST (e outras eleições)** Novo

#45 | Por: Pedro Manuel Romão Ferreira Pereira da Silva em: 21/07/2009 | 6 comentários
Parece-me ser uma boa ideia que se pudesse votar no fénix nas listas da AEIST, em vez de se exigir exclusivamente o voto presencial. Seria semelhante do que se já faz (e bem!) para eleger c d ...

**Histórico do Aluno** Novo

#71 | Por: Diogo Alexandre dos Santos Nunes em: 12/12/2009 | 5 comentários
No seguimento desta ideia (https://fears.ist.utl.pt/#Project3&viewFeature55) aqui vai outra: Implementar no Fénix, uma opção que permitisse consultar o histórico/aproveitamento de um aluno (...
### Resumo

- **Adicionar**

<table>
<thead>
<tr>
<th>Pesquisas Favoritas</th>
<th>Regime Simplificado (Pendentes)</th>
<th>Processos em Acesso Exclusivo</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Processos com Comentários por Ler</th>
<th>Reembolsos (Pendentes)</th>
<th>Acesso Rápido</th>
</tr>
</thead>
<tbody>
<tr>
<td>Não existem processos com novos comentários.</td>
<td>Não existem processos onde pode efetuar operações.</td>
<td>Id. do Processo: [ ] [Ver]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Os Meus Processos</th>
<th>Notificação por Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ainda não iniciou processos.</td>
<td>Pretendo receber notificações diárias por email dos processos pendentes</td>
</tr>
</tbody>
</table>

- **Sim** - [ ]

©2010 Instituto Superior Técnico
Fénix Framework: Key Features

- STM-based
- Transactional Domain Objects
- Domain Object Cache
- Transparent Persistence
- Distributed
Evaluation

• Real-world applications
  • Fénix EDU
  • dot.IST
  • FeaRS
  • ...
• Benchmark
  • TPC-W
Fénix EDU

2001: Started as a typical web application

2005: New architecture
Code Base

1.2 million LOC
12 full-time engineers
3,600 web pages
8,000 classes (1,200 domain)
NOT ‘Hello World!’
User Base

6,000 BSc
4,000 MSc & PhD
1,100 faculty members
Daily @ Feb 2011

Number of transactions vs. Day of month
Daily @ Feb 2011

Rate vs Day of month

- Write Tx
- Conflicts
Some Fénix statistics
Normal week
No way we could get this before

17th to 19th
September 3rd
Enrollment counts
Fénix transactional workload

Writes \( \sim 1\% \)

Conflicts \( \sim 1\% \)

Some txs \( > 10,000,000 \) reads

\[ \frac{reads}{writes} > 1.000 \]
<table>
<thead>
<tr>
<th></th>
<th>Reads</th>
<th>Writes</th>
<th>Conflicts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal day</td>
<td>898 802</td>
<td>15 330</td>
<td>9</td>
</tr>
<tr>
<td>17th February</td>
<td>3 306 600</td>
<td>116 651</td>
<td>745</td>
</tr>
</tbody>
</table>
Hardware

2 nodes (fault-tolerance)

2 quad-core CPU

32GB RAM (6GB used)

Underused!
2008: Already new architecture

Fewer users (no students)

No public pages
TPC-W

- Web commerce application: Bookstore
- JDBC vs. Fénix Framework
- Web Interactions Per Second (WIPS)
- Workloads:
  - Read-only
  - Browsing (5% writes)
  - Shopping (20% writes)
- Up to 8 app servers
Single server

- Java 6
- MySQL 5.1
- Tomcat 6
- 48-core: 10 clients
- 128GB RAM: <15GB used
Single server

![Graph showing Web Interactions Per Second (WIPS) versus Number of items for different combinations of FF and JDBC read-only, browsing, and shopping. The graph indicates performance under single server conditions.]
Single server

![Graph showing speed-up for FF/JDBC with different numbers of items: 1,000, 10,000, 100,000. The graph compares Read-only, Browsing, and Shopping activities.]
Single server

![Graph showing speed-up FF/JDBC versus number of items for different tasks: Read-only, Browsing, and Shopping. The graph indicates a 3.7x speed-up for Browsing at 10,000 items.]
## TPC-W specs

<table>
<thead>
<tr>
<th>#Emulated Clients</th>
<th>#DB Customers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2,880</td>
</tr>
<tr>
<td>10</td>
<td>28,880</td>
</tr>
<tr>
<td>60</td>
<td>172,800</td>
</tr>
</tbody>
</table>
Single server

![Graph showing speed-up FF/JDBC vs number of clients for different activities: Read-only, Browsing, Shopping. The graph shows that speed-up increases with the number of clients for Browsing and Shopping, while it remains constant for Read-only.]
Single server

Number of clients

Speed-up FF/JDBC

Read-only  Browsing  Shopping

22x
Clustered server

- 10 machines (client + 8 x server + DB)
- 40 clients evenly split
- LAN
- 8 cores
- 8GB RAM
Clustered server

![Bar chart showing WIPS for different numbers of Application Servers. The chart compares FF Read-only and JDBC Read-only.]

- Number of Application Servers: 2, 4, 6, 8
- WIPS: 0, 1000, 2000, ..., 9000
- FF Read-only
- JDBC Read-only

Legend:
- FF Read-only
- JDBC Read-only
- JDBC Read-only (Green)
Clustered server

![Clustered server diagram](image)

- Number of Application Servers:
  - 2
  - 4
  - 6
  - 8

- WIPS:
  - FF Browsing
  - JDBC Browsing
  - FF Shopping
  - JDBC Shopping
Conclusions

• Current Tx away from App Logic
  • DB access is expensive
• Recent hardware + STM
• Strict Serializability & Performance
• Best with high read/write ratio