Temporal Issues in a Rich Domain Model

Erwin Vervaet

ervacon

Presentation goal

An introduction to temporal issues in rich domain models

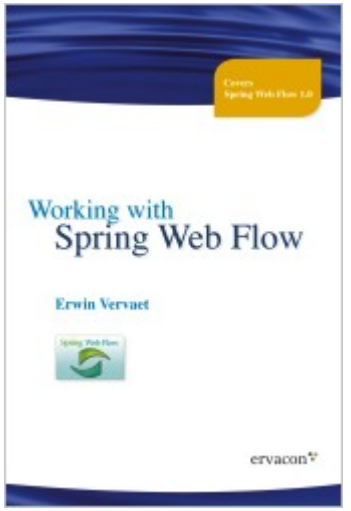
and

Illustrate an elegant and practical implementation using a standard RDBMS, Java 5, and Hibernate

ervacon

About the speaker

- Erwin Vervaet
  - Senior consultant at Ervacon
  - Extensive experience using JSE and JEE
  - Spring Framework project member
  - Spring Web Flow project founder
  - Author of the book
    Working with Spring Web Flow
    http://www.ervacon.com/products/swfbook



ervacon

Overview

- Tracking time
  - non-temporal
  - single-temporal
  - bi-temporal
- Implementation techniques
- Show me the code!
  - An elegant and practical implementation using
    - Standard RDBMS
    - Java 5
    - Hibernate

ervacon

Overview

- **Tracking time**
  - non-temporal
  - single-temporal
  - bi-temporal
- Implementation techniques
- Show me the code!
  - An elegant and practical implementation using
    - Standard RDBMS
    - Java 5
    - Hibernate

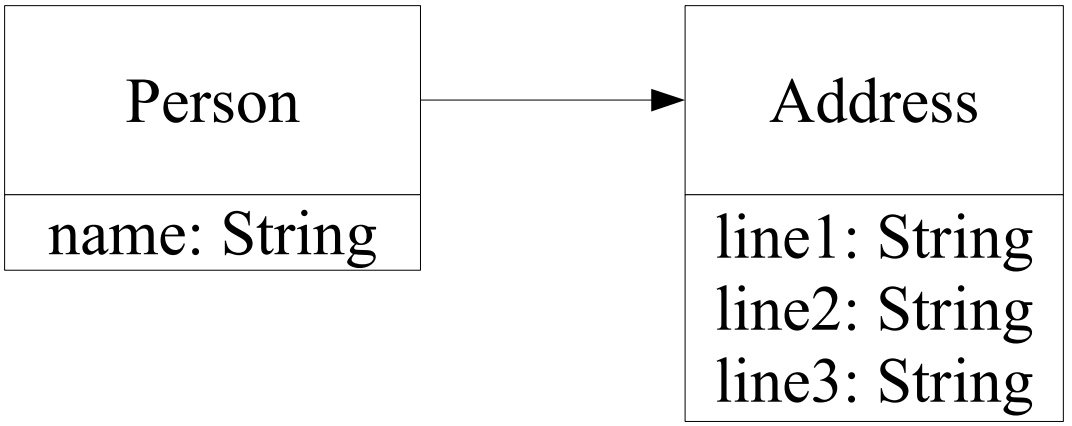
ervacon

Tracking Time

- Very common concern in domain models
- Hard problem to tackle!

A simple example

- John Doe's residence
  - Where does he live?



Non-temporal

- No time tracking
- We can only answer questions about the current situation
  - *Where does John Doe (currently) live?*
- This is what you get for free
  - A classic RDBMS is non-temporal
  - Directly supported by all ORMs
- Easy!

ervacon

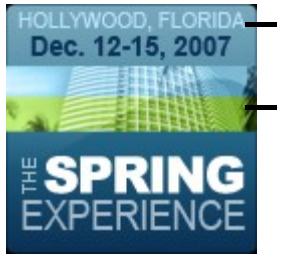
Single-temporal
Variant 1: actual-temporal

- Adds a *validity interval*: when is something valid in actual time?
- Allows us to answer questions about the situation on a certain moment
  - *Where did John Doe live on September 1, 1994?*
- The application will have to track a validity interval
- Intermediate complexity
  - Custom coding needed

ervacon

Single-temporal
Variant 2: record-temporal

- Adds a *record interval*: when was something known (recorded)?
- Allows us to answer questions about what we knew at a certain point in time
  - *On October 1, 1994, what did we know about John Doe's residence?*
- The application will have to track a record interval
- Intermediate complexity
  - Custom coding needed
  - SVN, CVS, ...



ervacon

Bi-temporal

- Combines actual-temporal and record-temporal change tracking
- Allows us to answer questions about what we knew at a certain point in time about the situation at another moment
  - *On October 1, 1994, what did we know about John Doe's residence on September 1, 1994?*
- The application will have to track **two** time intervals!
- Hard!
  - Custom coding needed

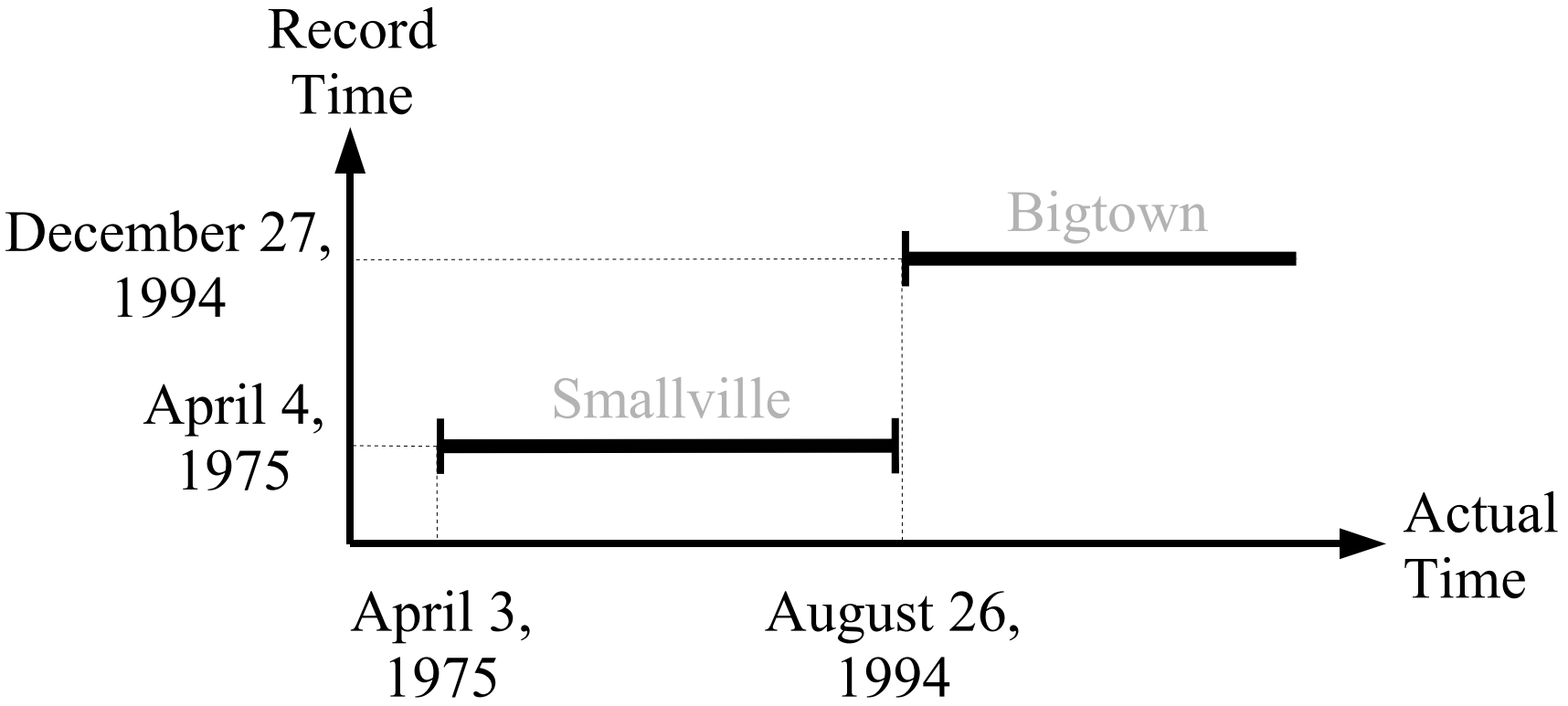
ervacon

Example

- “John Doe is born on April 3, 1975. His father registers him as a new Smallville resident the next day.”
- “After graduation, John Doe moves from Smallville to Bigtown. Although he moved out on August 26, 1994, he only registers the new address officially on December 27, 1994”

ervacon

Graphically



- Let's answer the questions posed earlier!

Where does John Doe live?

- Non-temporal
  - In Bigtown
- Actual temporal
  - In Bigtown
- Record temporal
  - In Bigtown
- Bi-temporal
  - In Bigtown

ervacon

Where did John Doe live on September 1, 1994?

- Non-temporal
  - *Unknown -> Bigtown*
- Actual temporal
  - Bigtown
- Record temporal
  - *Unknown -> Bigtown*
- Bi-temporal
  - Bigtown

ervacon

On October 1, 1994, what did we know about John residence?

- Non-temporal
  - *Unknown -> Bigtown*
- Actual-temporal
  - *Unknown -> Bigtown*
- Record-temporal
  - Smallville
- Bi-temporal
  - Smallville

ervacon

On October 1, 1994, what did we know about John's residence on September 1, 1994?

- Non-temporal
  - *Unknown -> Bigtown*
- Actual-temporal
  - *Unknown ->Bigtown*
- Record-temporal
  - *Unknown -> Smallville*
- Bi-temporal
  - Smallville

ervacon

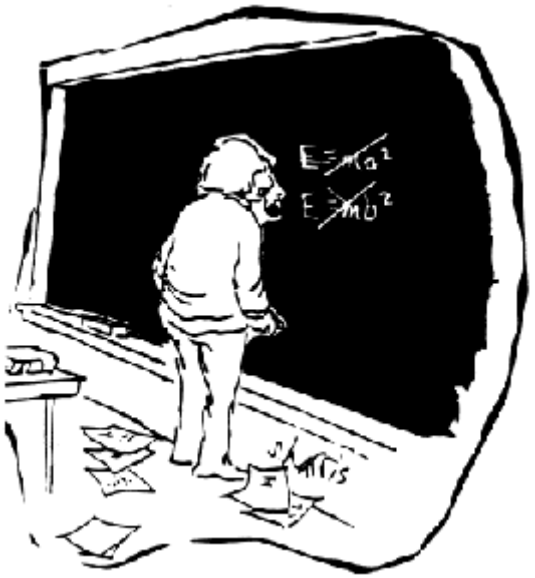
Overview

- Tracking time
  - non-temporal
  - single temporal
  - bi-temporal
- **Implementation techniques**
- Show me the code!
  - An elegant and practical implementation using
    - Standard RDBMS
    - Java 5
    - Hibernate

ervacon

Implementing bi-temporality

- Two options currently
  - Temporal database
  - Custom code



Temporal databases

- Active research area
  - Adding temporality to SQL: SQL/Temporal
- Implementations – still immature
  - TimeDB (http://www.timeconsult.com)
    - A bi-temporal query language on top of a classic RDBMS
  - Oracle FlashBack
    - Record temporal
  - ...

ervacon

Adding time information to a database

- General implementation technique
  - 4 extra columns:
    - [validityFrom, validityTo[
    - [recordFrom, recordTo[

ADDRESS

ID	PERSON_ID	LINE1	LINE2	LINE3	VF	VT	RF	RT

ervacon

Manipulating temporal data

- John Doe is Born on April 3, 1975 and registered as a Smallville resident the next day.

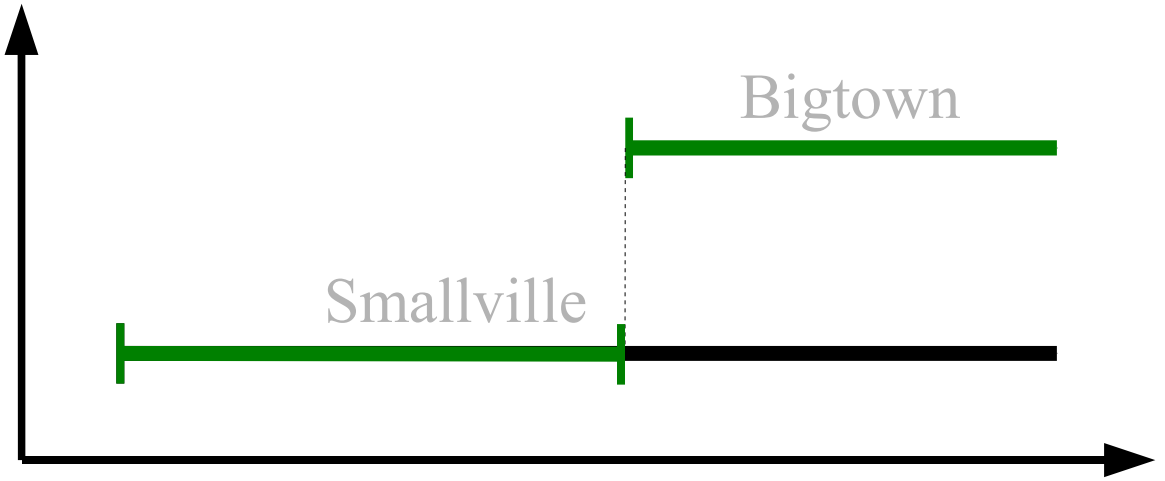
Name	Address	VF	VT	RF	RT	
John Doe	Smallville	1975/4/3		1975/4/4		Insert

Manipulating temporal data

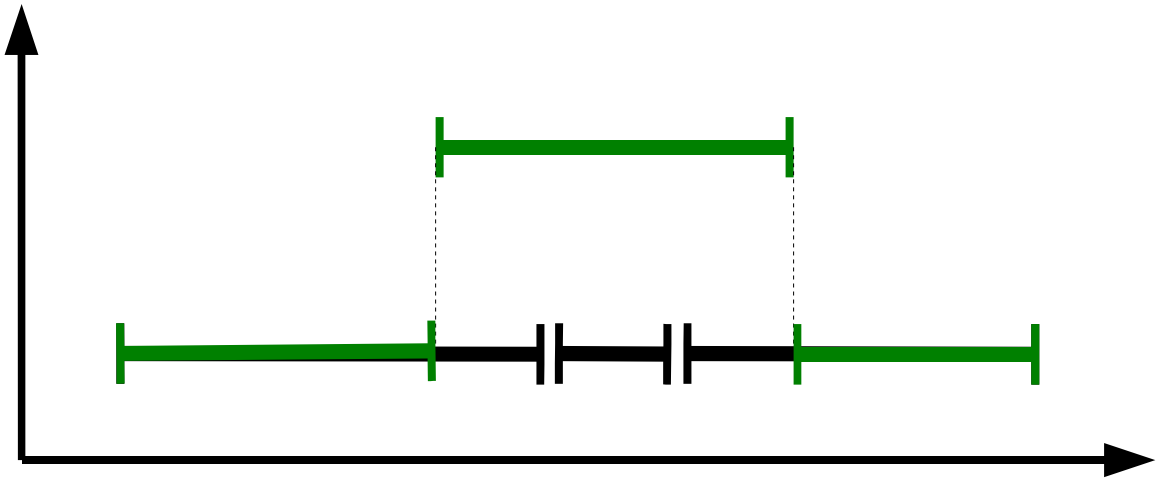
- John Doe moves from Smallville to Bigtown on August 26, 1994, but he only registers the new address officially on December 27, 1994.

Name	Address	VF	VT	RF	RT	
John Doe	Smallville	1975/4/3		1975/4/4	1994/12/27	Update
John Doe	Smallville	1975/4/3	1994/8/26	1994/12/27		Insert
John Doe	Bigtown	1994/8/26		1994/12/27		Insert

Data manipulation algorithm



Data manipulation algorithm
A more complex case



Querying temporal data

- What do we currently know about the history of John Doe's residence?

Name	Address	VF	VT	RF	RT	
John Doe	Smallville	1975/4/3		1975/4/4	1994/12/27	
John Doe	Smallville	1975/4/3	1994/8/26	1994/12/27		
John Doe	Bigtown	1994/8/26		1994/12/27		

Key facets

- *Key facets* of temporal change tracking are well understood
  - The identity issue
  - Immutability
  - Validity & recording time details
  - Controlling time

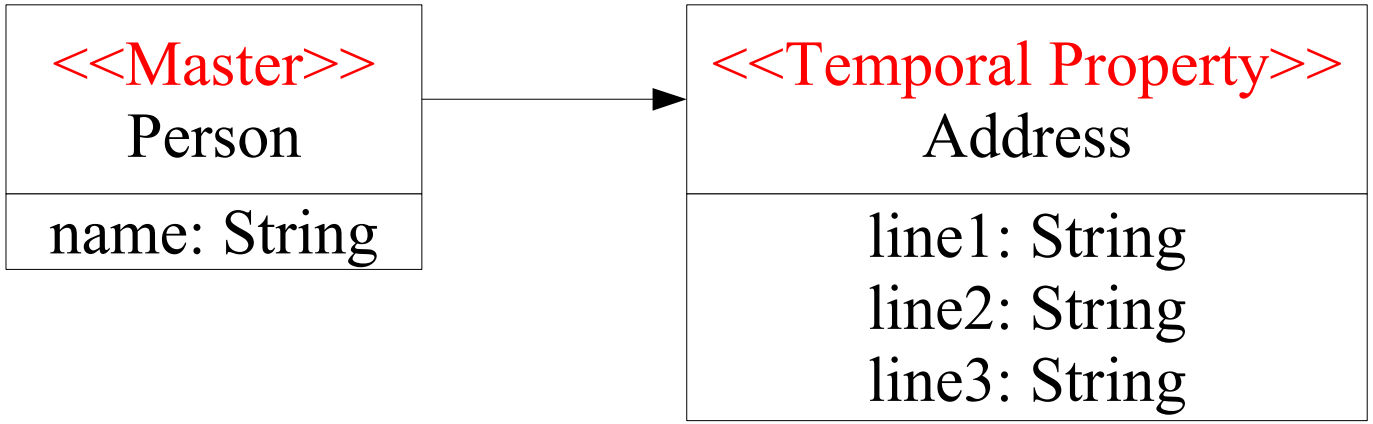
ervacon

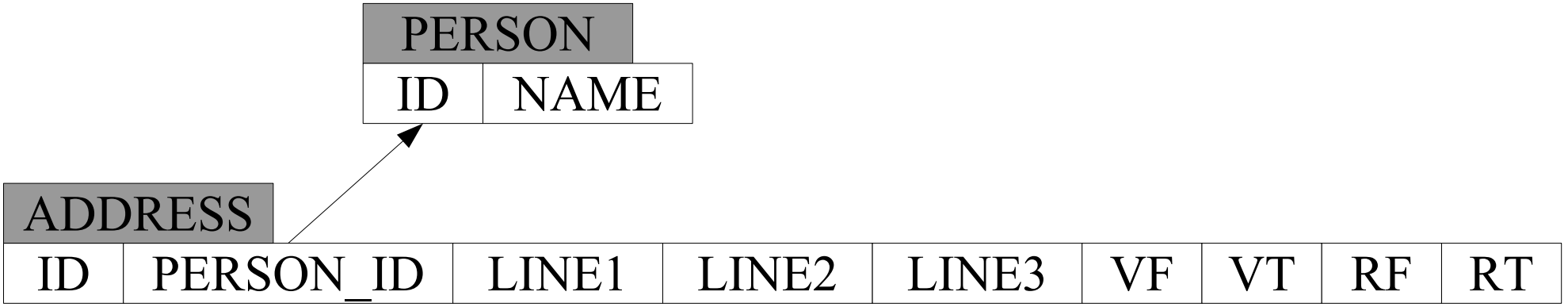
The identity issue

- Since there can be multiple *versions* of temporally tracked data, there is an identify issue
- You always have a **master** with **temporal properties**
  - Master data is either immutable or non-temporal
    - *Entities*
  - Temporal properties linked to the master and temporally tracked
    - *Values*

ervacon

The identity issue (contd.)





ervacon

Immutability

- Temporal properties are *technically immutable*
  - Values!
  - Changing the value results in a new value being added to the properties history
    - Otherwise we would loose information!

Validity time

- The application has free control over validity time
  - **Additive** changes: more information added to the end the history
    - *Starting now, John Doe's lives in Bigtown*
  - **Retroactive** changes: changing values in the past
    - *Last year, John Doe lived in New York*
  - **Proactive** changes: changing values in the future

ervacon

Recording time

- The time you record something is always **now**
  - Doing otherwise would lead to corruption
    - You can't asses the impact of your changes since later events have already occurred
      - Ex.: What would happen to today if you went back in time and prevented the JFK assassination?
  - Recording time is **strictly increasing**
- You only need to worry about record time when querying

ervacon

Controlling time

- Don't use wall clock time! (sysdate)
- Systems that temporally track information need more control over *time*
  - When processing data
  - When manipulating data
    - Record time can't evolve during a single transaction
- Simulating an infinitely fast computer

ervacon

How do I implement this?

- No direct support in commercial RDBMSs
- No direct support in ORM tools

➔ Custom code needed

Props to
Christophe Vanfleteren

Temporal patterns

- Martin Fowler's PoEAA describes several temporal patterns
  - **Audit Log**
    - *A simple log of changes, easy to write and non-intrusive*
  - **Effectivity**
    - *Add a time period to an object to show when it is effective (valid)*
    - Actual-temporal

ervacon

Temporal patterns (contd.)

- **Temporal property**
  - *A property that changes over time*
- **Temporal object**
  - *An object that changes over time*
- **Snapshot**
  - *A view of an object at a point in time*

- These patterns are very *code centric*
  - Leave persistence question open

ervacon

Overview

- Tracking time
  - non-temporal
  - single-temporal
  - bi-temporal
- Implementation techniques
- **Show me the code!**
  - An elegant and practical implementation using
    - Standard RDBMS
    - Java 5
    - Hibernate

ervacon

Implementation goals

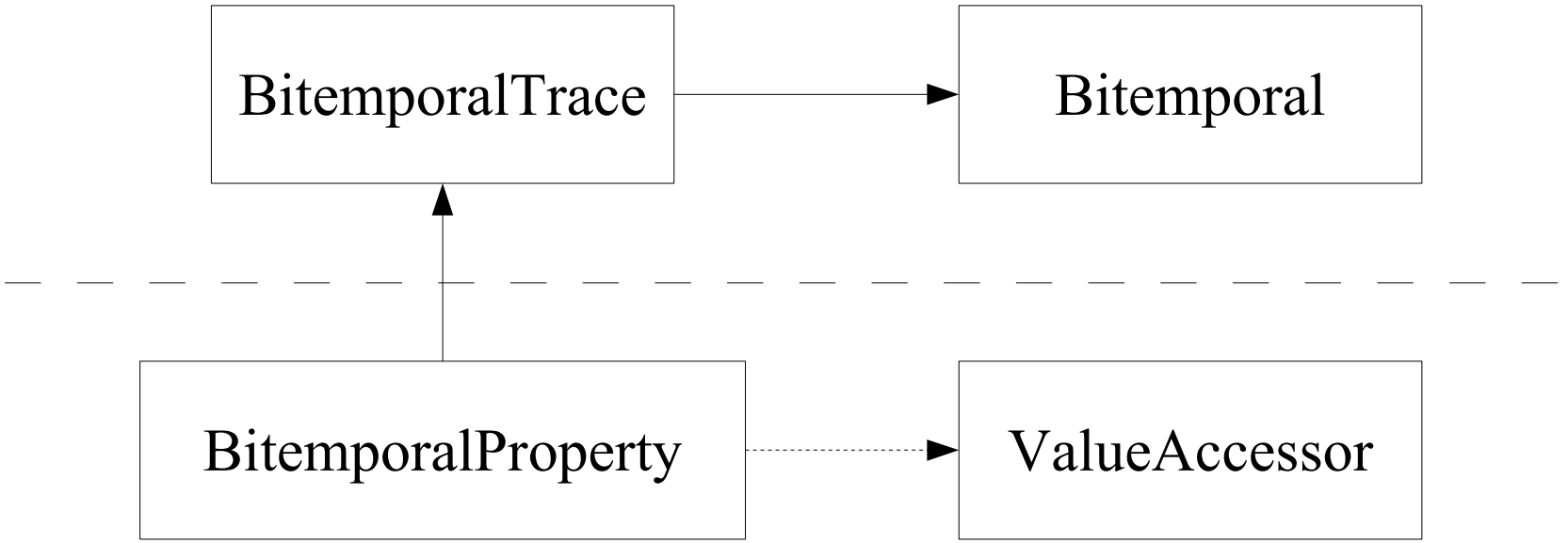
- Simple and elegant API
  - Hide temporality when not needed
  - Give access to it when needed
- Persistable
  - Using standard RDBMS
    - Oracle, ...
  - Using standard ORMs
    - Hibernate, ...

ervacon

Assumptions

- *Slowly* changing data
  - Address
  - Name
  - Social status
  - ....
- Don't use it for a stock ticker...

API structure



Show me the code!

HOLLYWOOD, FLORIDA
Dec. 12-15, 2007
THE SPRING EXPERIENCE

ervacon

API usage – setup

```
public class Person {

    private Long id;
    private String name;
    private Collection<BitemporalWrapper<Address>> address =
        new LinkedList<BitemporalWrapper<Address>>();


    ...

    public WrappedBitemporalProperty<Address> address() {
        return new WrappedBitemporalProperty<Address>(address);
    }

    ...

}
```

ervacon

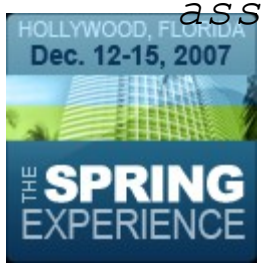
Simple API usage

- Non temporal

```
Person pete = new Person("Pete");
pete.address().set(new Address("Foostreet", "Bartown", "USA"));
assertTrue(pete.address().hasValue());
assertEquals("Foostreet", pete.address().now().getLine1());
```

- Temporal

```
Person pete = new Person("Pete");
pete.address().set(new Address("Foostreet", "Bartown", "USA"),
        TimeUtils.from(TimeUtils.day(1, 1, 2000)));
assertTrue(pete.address().hasValueOn(TimeUtils.day(1, 1, 2001)));
assertEquals("Foostreet", pete.address().on(
        TimeUtils.day(1, 1, 2001)).getLine1());
assertEquals(
        TimeUtils.from(TimeUtils.day(1, 1, 2000)),
        pete.address().get().getValidityInterval());
```



ervacon

API usage – example

- John Doe is Born on April 3, 1975 and registered as a Smallville resident the next day.

```
TimeUtils.setReference(TimeUtils.day(4, 4, 1975));
Person johnDoe = new Person("John Doe");
johnDoe.address().set(
        new Address("Some Street 8", "Smallville", "FL, USA"),
        TimeUtils.from(TimeUtils.day(3, 4, 1975)));
```

ervacon

API usage – example (contd.)

- John Doe moves from Smallville to Bigtown on August 26, 1994, but he only registers the new address officially on December 27, 1994.

```
TimeUtils.setReference(TimeUtils.day(27, 12, 1994));
johnDoe.address().set(
        new Address("Some Avenue 773", "Bigtown", "FL, USA"),
        TimeUtils.from(TimeUtils.day(26, 8, 1994)));
```

ervacon

API usage – example (contd.)

- What do we currently know about the history of John Doe's residence?

```
List<BitemporalWrapper<Address>> history =
        johnDoe.address().getHistory();
```

- On October 1, 1994, what did we know about John's residence on September 1, 1994?

```
johnDoe.address().on(
        TimeUtils.day(1, 9, 1994),
        TimeUtils.day(1, 10, 1994)).getLine2(); // Smallville
```

ervacon

Persistence

- `BitemporalTrace` manipulates a `Collection<Bitemporal>`
- Client code accesses the data in this collection trough a `BitemporalProperty`
- Hibernate persists the raw collection

```
<hibernate-mapping default-access="field">
    <class name="com.ervacon.bitemporal.Person">

        ...

        <bag name="address" cascade="all-delete-orphan">
            <key column="person_id" not-null="true" update="false"/>
            <one-to-many entity-name="Address"/>
        </bag>
```

ervacon

Concluding

- Bi-temporality is a challenging and interesting problem domain
- With some clever coding you can hide most of the complexity from client code while still leveraging proven technology

ervacon

Q & A