adaptTo()

APACHE SLING & FRIENDS TECH MEETUP
BERLIN, 28-30 SEPTEMBER 2015

Conflict handling with Oak

Michael Dürig, Adobe Research

Agenda

- Why?
- What?
- How?
- Conclusion

https://www.flickr.com/photos/35168673@N03/3792471453/

All you need to know about JCR

- Hierarchical database
- Oak implements JCR
- Plugins for customisation

Why?

adaptTo()

Collaboration causes conflicts

- Collaborative applications
- Internet scale applications
  - Scalability
  - Weak consistency

adaptTo()

What?

What is a conflict?

- Conflict semantics
  - Back-end
  - Application
- Incompatible, concurrent updates

Resource in JCR

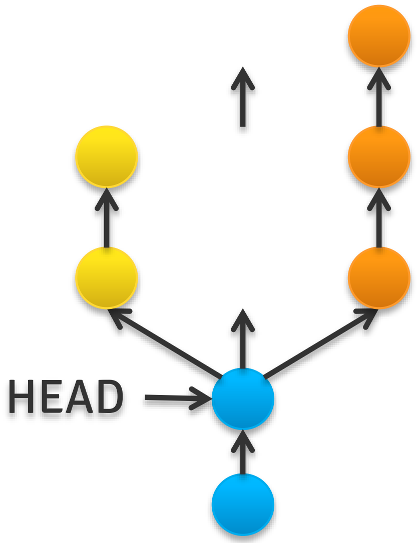
- Identified by path
- No conflicts between
  - Nodes and properties
  - Items with different names

Incompatible updates

	Change	Add	Remove
Remove	✗	NA	✓
Add	NA	✓ ✗	
Change	✓ ✗		

Saving changes

- Rebase
- Handle conflicts
  - Built-in
  - Custom
- Persist or fail
  - Run commit hooks



How?

Conflict handling strategies

Lock	Retry	Resolve	Avoid
Pessimistic	Optimistic	Proactive	Anticipatory
Consensus	Brute force	Resolving	Contention free
Wasteful	Wasteful	Economical	Economical
Serial			Parallel
Transparent	Not transparent	Transparent	Not transparent
	Constrained	Constrained	

Atomic counter

- Use cases
  - Rating
  - Booking
- Building blocks
  - Up/down votes
  - Average

Naïve implementation

```
void increment(long delta) {
    while (true) {
        Property counter = node.getProperty("count");
        long count = counter.getLong();
        counter.setValue(count + delta);
        try {
            counter.getSession().save();
            break;
        } catch (RepositoryException ignore) { }
    }
}
```

Avoiding conflicts

- Data structure
    - Avoid application conflicts
    - Leverage back-end

Share nothing

- Counter per process
- Sum of counters
- Accumulate via commit hook

Usage

```
    void increment(long delta) {
        session.getNode("/counter")
            .setProperty("oak:increment", delta);
        session.save();
    }

    long get() {
        session.getNode("/counter")
            .getProperty("oak:counter").getLong();
    }
```

Accumulate

```
public class AtomicCountEditor extends DefaultEditor {

    @Override
    public void propertyAdded(PropertyState after) {
        ...
    }

    @Override
    public void leave(NodeState before, NodeState after) {
        ...
    }
}
```

Accumulate

```
public class AtomicCountEditor extends DefaultEditor {
    private final NodeBuilder builder;
    private long total;

    public AtomicCountEditor(NodeBuilder builder) {
        this.builder = builder;
        total = builder.getProperty("oak:counter").getValue(LONG);
    }

     ...
}
```

Accumulate

```
public void propertyAdded(PropertyState after) {
    if ("oak:increment".equals(after.getName()) {
        total += after.getValue(LONG);
        builder.removeProperty("oak:increment");
    }
}

public void leave(NodeState before, NodeState after) {
    builder.setProperty("oak:counter", total);
}
```

Demo

https://www.flickr.com/photos/ruiwen/3260095534/

From here...

- Bidding
  - Maximum instead of addition
- Shopping
  - Set union instead of addition
- Signal
  - Logical or instead of addition

Resolving conflicts

- Conflict handler
  - Close to source
  - No retry

Multi-value register

- Store conflicting values
  - Materialise conflict
  - Delegate resolution
  - Additional round trip

Usage

```
session1.getNode("/mv").setProperty("value", 1);
session2.getNode("/mv").setProperty("value", 2);
```

```
session1.save();
session2.save();
```

```
session3.getProperty("/mv/value");          // Multi valued [1, 2]
```

Conflict handler

```
public interface PartialConflictHandler {
    Resolution addExistingProperty(...);
    Resoluti    chan                  .);
    Resolution                        .);
    Resolution de                     .);
    Resolution del                    .);
    Resolution add
    Resolution cha
    Resolution deleteChangedNode(...);
    Resolution deleteDeletedNode(...);
}
```

```
enum Resolution {
    OURS,
    THEIRS,
    MERGED
}
```

Conflict handler

```
@Override
public Resolution changeChangedProperty(
        NodeBuilder parent,
        PropertyState ours,
        PropertyState theirs) {

    parent.setProperty(
        ours.getName(),
        union(getValues(ours), getValues(theirs)), LONGS);
        return Resolution.MERGED;
}
```

Conflict handler

```
@Override
public Resolution changeDeletedProperty(...) {
     return Resolution.OURS;
}

@Override
public Resolution deleteChangedProperty(...) {
     return Resolution.THEIRS;
}

@Override
public Resolution deleteDeletedProperty(...) {
     return Resolution.MERGED;
}
```

Demo

```
[illegible]ments.callee.u,c=arguments.
[illegible]hasOwnProperty(p)&&c(d[p],
[illegible]" + hr + ")break;}" ;
[illegible]" + lr + ")return [0];";
[illegible]s._dirty=true;" +
[illegible]delete d[r[i]];return [r.lengt
[illegible]hasOwnProperty(p)&&c(d[p]
```

https://www.flickr.com/photos/ruiwen/3260095534/

Conclusion

Conclusion

- Avoid conflicts
  - Private copy
  - Accumulate
- Resolve conflicts
  - Custom handler
  - Prevent retry

Thank you

- https://github.com/mduerig/oak-crdt