Hummingbird

An Open Source Ground Segment for Small Satellites

A true story of doing things different

GSAW 2012

Gert Villemos (Villemos Solutions)

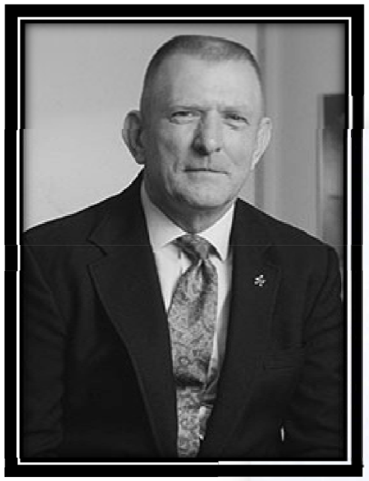
Mark Doyle (Logica) and Johannes Klug (Logica)

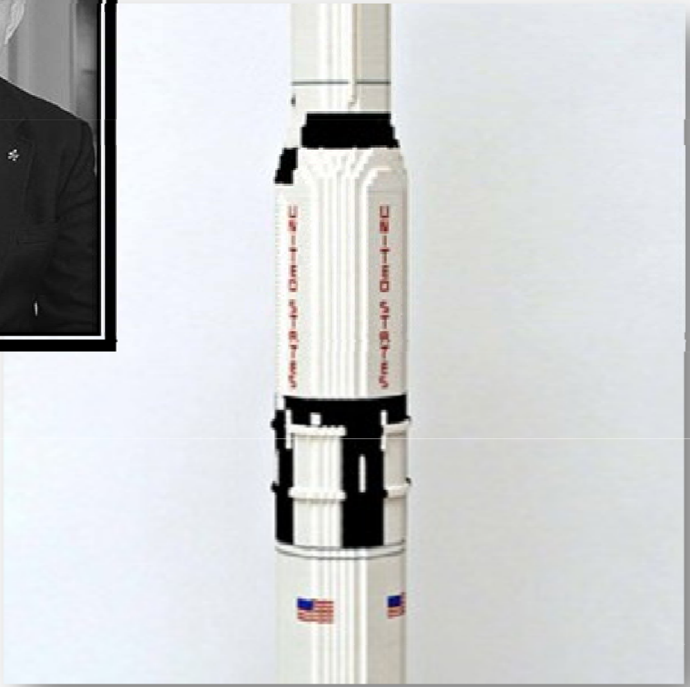
What is ‚Hummingbird‘

- An open source software framework (Apache License)
- For building ground segment systems for small satellites
- A ‘back to basics’ approach
- Using simplicity as a design principle
- Pushing as much functionality as possible to existing technologies
- www.hbird.org and facebook

The world

5 years…





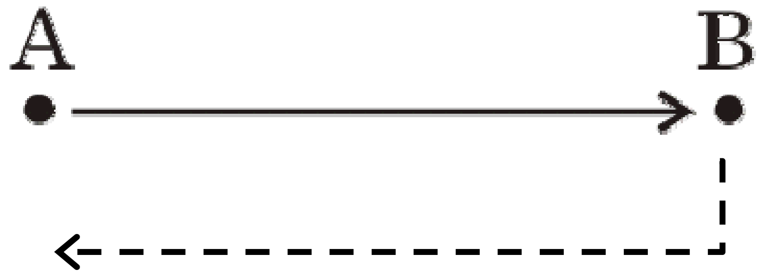
8 years…

The world we live in

Business Plan

2.0

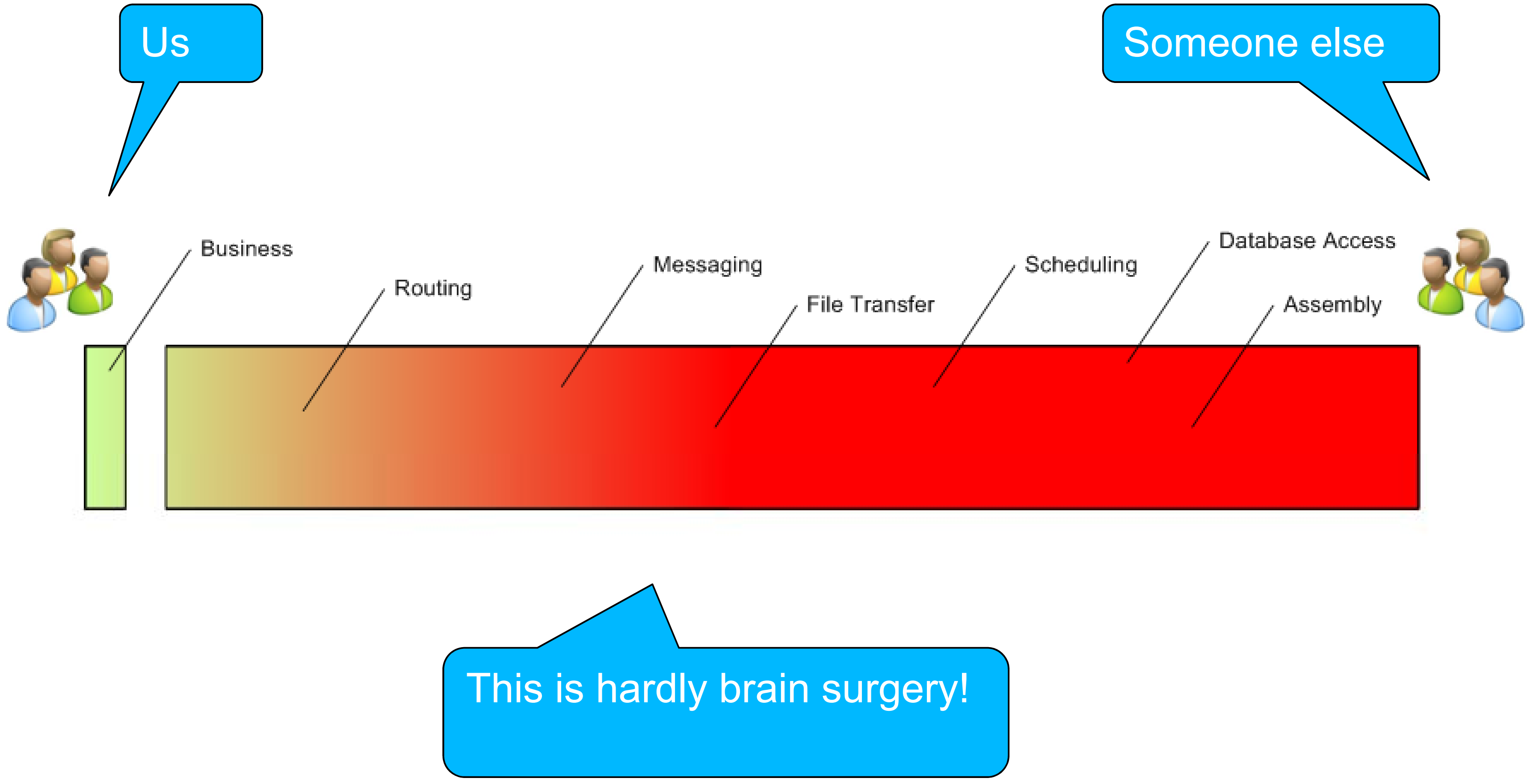
Back to Basics



Thesis

- Ground segment systems for satellites is no longer special; we move data from A to B
- Modern network technologies can be used and are better. Lets stop reinventing the wheel
- Complexity is inherited (... and often cultural) and propagates through our systems. It is the root of all evil
- Find the root course and remove it. Ground segment system can be really simple

‚New‘ Implementation Concept



Technology Stack (fantastic four)

- ➢ Spring
- ➢ Camel
- ➢ ActiveMQ
- ➢ Cometd

... but components in other languages can be integrated

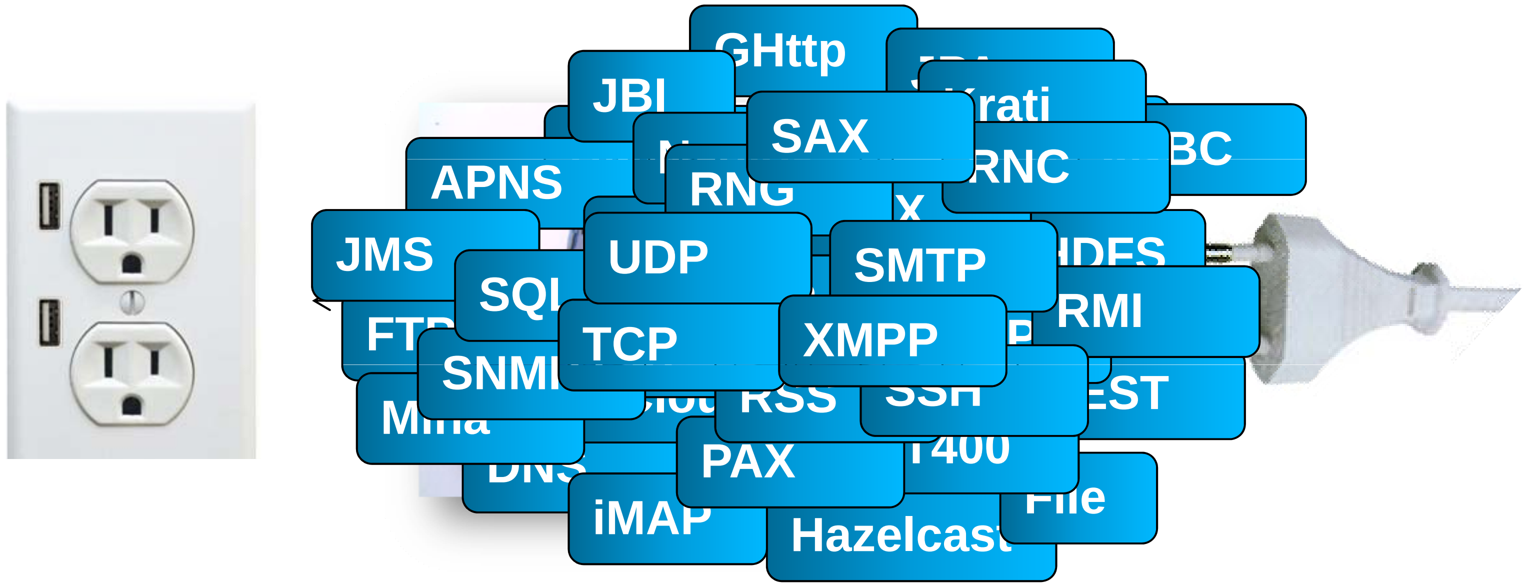
- ➢ ... and yes, its all Java

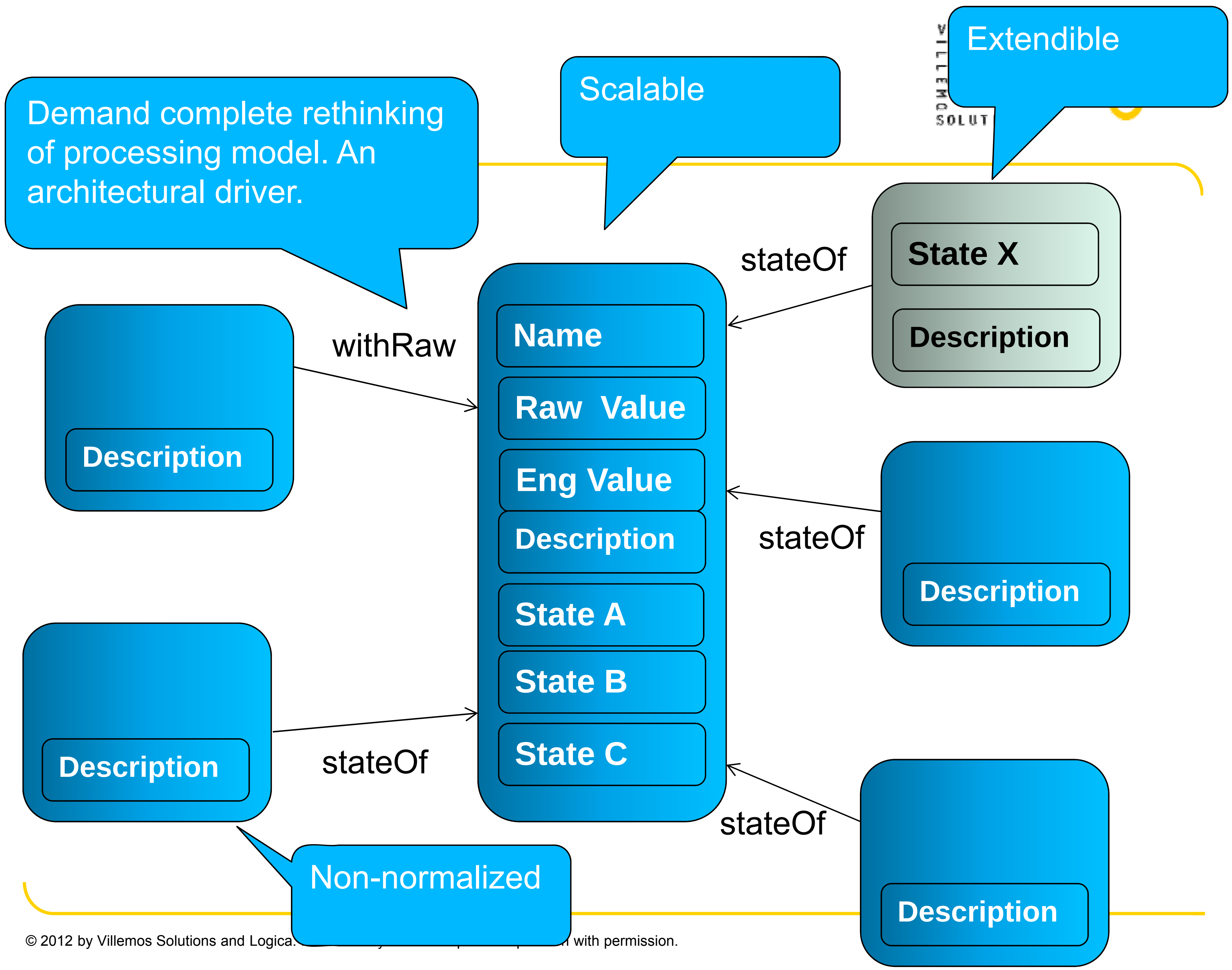
Evolution

- Hummingbird 0.1
  - Classical separation into tiers (transport, business, presentation)
  - CCSDS stack Frames → Packets → Parameters
  - Distribution of predefined types
  - Centered on centralized ‘System Model’ (think runtime XTCE)

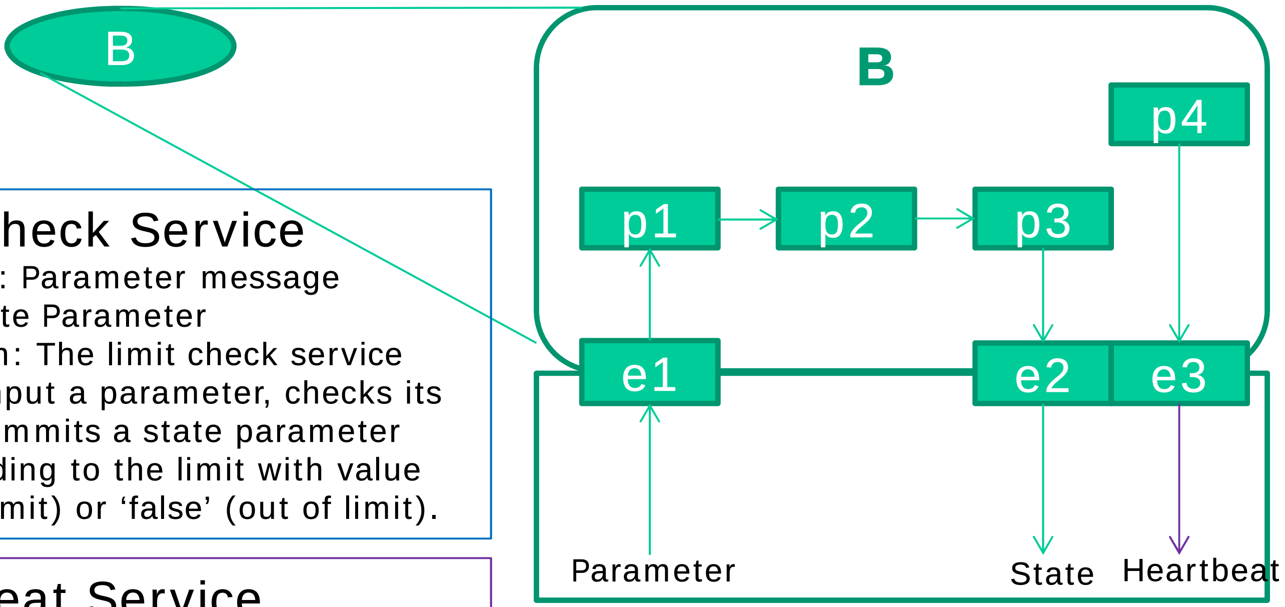
- Hummingbird 0.2
  - CCSDS stack ‘banned’ to transport tier, fully encapsulated
  - True asynchronous processing
  - Semantic information model, non-relational databases
  - Distribution of ‘what-ever’ with plugable services

System Integration




Demand complete rethinking of processing model. An architectural driver.
Scalable
Extendible
withRaw
Description
Name
Raw Value
Eng Value
Description
State A
State B
State C
stateOf
State X
Description
stateOf
Description
stateOf
Description
stateOf
Description
Non-normalized

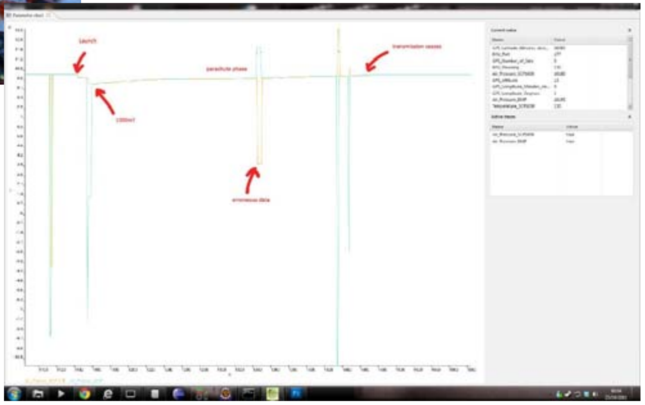
Service



Limit Check Service
Consumes: Parameter message
Emits: State Parameter
Description: The limit check service takes as input a parameter, checks its limit and emmits a state parameter corresponding to the limit with value 'true' (in limit) or 'false' (out of limit).

Heartbeat Service
Consumes: None
Emits: Heartbeat
Description: Issues a ‚alive' message at intervals.

First Flight

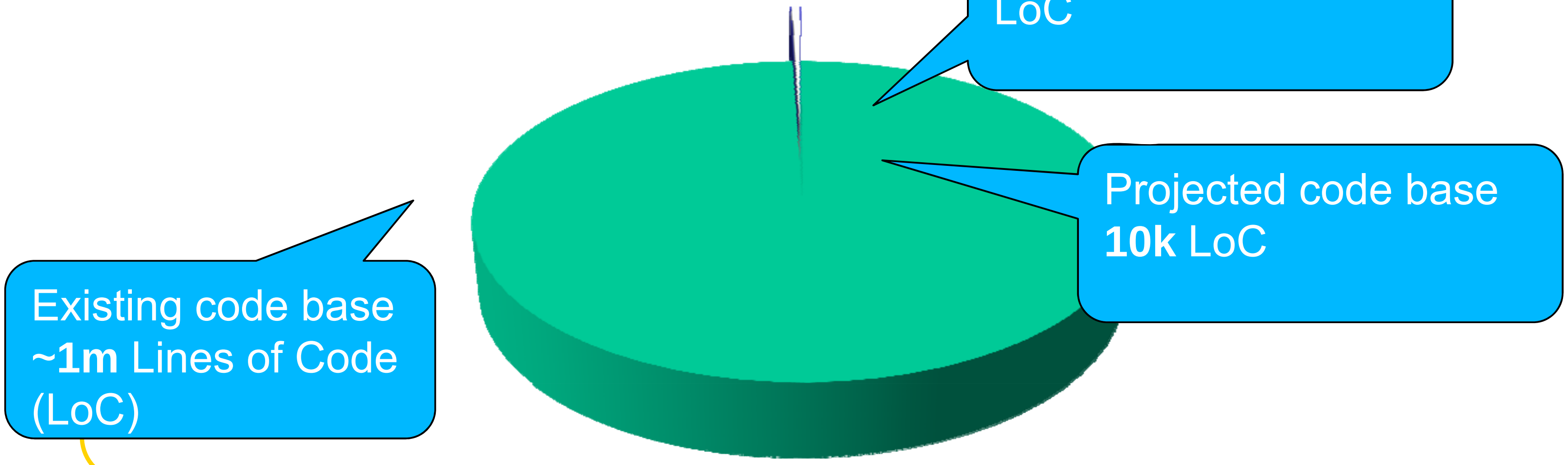


- Comming up
  - EstCube (University of Estland)
  - Strand (Surrey Satellites)
  - TechDemoSat-1 (UK Space Agency)

Initial Results

Having implemented guestimated 10% of the functionality of similar existing systems

- Commanding (scheduling, pre release validation, release, verification and history)
- Monitoring (parameter creation, limit check, consistency check, calibration and history)
- Orbit and contact prediction



Value

- Highly motivated staff, learning by doing
- Concepts feedback into winning ‘normal’ work
- Door opener to new, and frequently unexpected, markets
- Great PR
- Neither predictable nor quantifiable but very real

Business Model

- Open Source Core

- Enterprise Edition
  - Liability
  - Tested
  - Proven
  - Documented
  - Evolution plan
  - Ahead of the curve

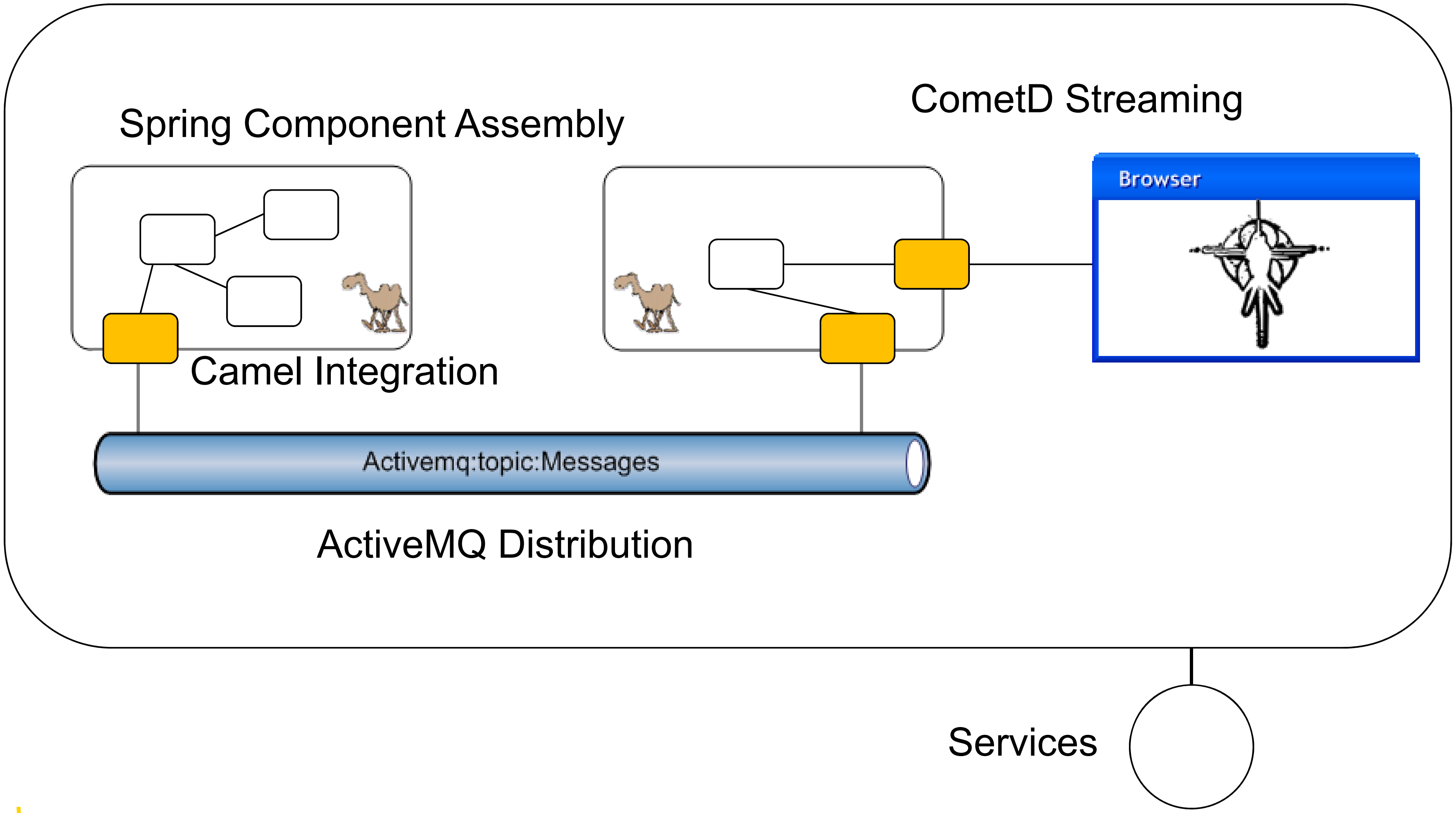
- Services and added value products

Conclusions

- Nothing is more convincing than to stop talking and start doing
- Standard technologies (of course) work for space data systems
- System integration can be very simple, we make it complex
- ‚Managing‘ an open source community is not simple

- Read more at: www.hbird.org and facebook

Business Tier



WARNING: Source code ahead!

… the message is not in the code itself, but in the changes to the code

The Modern Code Base

```
public class Manager implements IManager {

  protected IWorker worker = new Worker();

  public void manage() {
    worker.work("1:2:3");
  }
}
```

```
public class Worker implements IWorker {

  protected IPublisher publisher = new Publisher();

  public void work(String values) {
  String[] elements = values.split(":");
    for (String element : elements) {
      long value = Long.parseLong(element);
     publisher.display(value);
    }
  }
}
```

```
public class Publisher implements IPublisher {

  public void display(long value) {
    System.out.println("Test value: " + value);
  }
}
```

The Modern Code Base

```
public class Manager implements IManager {

  protected IWorker worker = new Worker();

  public void manage() {
    worker.work("1:2:3");
  }
}
```

```
public class Worker implements IWorker {

  protected IPublisher publisher = new Publisher();

  public void work(String values) {
   String[] elements = values.split(":");
    for (String element : elements) {
      long value = Long.parseLong(element);
     publisher.display(value);
    }
  }
}
```

```
public class Publisher implements IPublisher {

  public void display(long value) {
   System.out.println("Test value: " + value);
  }
}
```

Only three lines of **business logic** here...

The Modern Code Base

```
public class Manager ~~implements IManager~~ {

  ~~protected IWorker worker = new Worker();~~

  public void manage() {
    worker.work("1:2:3");
  }
}
```

```
public class Worker ~~implements IWorker~~ {

  ~~protected IPublisher publisher = new Publisher();~~

  public void work(String values) {
  String[] elements = values.split(":");
    for (String element : elements) {
      long value = Long.parseLong(element);
     ~~publisher.display(value);~~
    }
  }
}
```

```
public class Publisher ~~implements IPublisher~~ {

  public void display(long value) {
    System.out.println("Test value: " + value);
  }
}
```

```
<bean id="manager" class="foo.Manager"/>

<bean id="worker" class="foo.Worker"/>

<bean id="publisher" class="foo.Publisher"/>

<route>
        <from uri="timer://foo?period=60000"/>
        <to uri=bean:manager" />
        <to uri="bean:worker"/>
        <to uri=„bean:publisher"/>
</route>
```

The Modern Code Base

```
public class Manager ~~implements IManager~~ {

~~  protected IWorker worker = new Worker();~~

  public void manage() {
    worker.work("1:2:3");
  }
}
```

```
public class ~~Worker implements IWorker~~ {

  ~~protected IPublisher publisher = new Publisher();~~

  public void work(String values) {
  String[] elements = values.split(":");
~~    for (String element : elements) {~~
~~      long value = Long.parseLong(element);~~
~~     publisher.display(value);~~
~~   }~~
  }
}
```

```
public class Publisher ~~implements IPublisher~~ {

  public void display(long value) {
    System.out.println("Test value: " + value);
  }
}
```

```
<bean id="manager" class="foo.Manager"/>

<bean id="worker" class="foo.Worker"/>

<bean id="publisher" class="foo.Publisher"/>

<route>
        <from uri="timer://foo?period=60000"/>
        <to uri=bean:manager" />
        <split>
                <method bean="worker"/>
                <to uri=„bean:publisher"/>
        </split>
</route>
```

The Modern Code Base

```
public class Manager {

  public String manage() {
    return "1:2:3";
  }
}
```

```
public class Worker {

  public String[] work(String values) {
    return values.split(":");
  }
}
```

```
public class Publisher {

  public void display(long value) {
    System.out.println("Test value: " + value);
  }
}
```

```
<bean id=“manager” class=“foo.Manager”/>

<bean id=“worker” class=“foo.Worker”/>

<bean id=“publisher” class=“foo.Publisher”/>

<route>
        <from uri=“timer://foo?period=60000”/>
        <to uri=bean:manager" />
        <split>
                <method bean=“worker”/>
                <to uri=„bean:publisher"/>
        </split>
</route>
```

Massive code base reduction

Only business logic in the code, routing in the configuration

Fantastic code metrics (no coupling)

100% test coverage easy to reach