ESB pilot project at the FMI

EGOWS 2008

Pekka Rantala

Finnish Meteorological Institute

Contents

What is an ESB?

- **Enterprise Service Bus**
- **From Mule glossary: *"An architecture that allows different applications to communicate with each other by acting as a transit system for carrying data between applications within or outside your intranet."***
  - Typically the term "ESB" is used to denote the software infrastructure that enables such an architecture
- **A messaging integration solution**
  - Canonical messages → decoupling of disparate parts of a system so that everything only needs to communicate with the bus
  - (Mostly) asynchronous communication: programming model considerations
- **Most ESB implementations offer a wide selection of adapters or "end-points" (JMS, file, ftp, socket, ...) out-of-the-box for applications' use**

FMI production environment

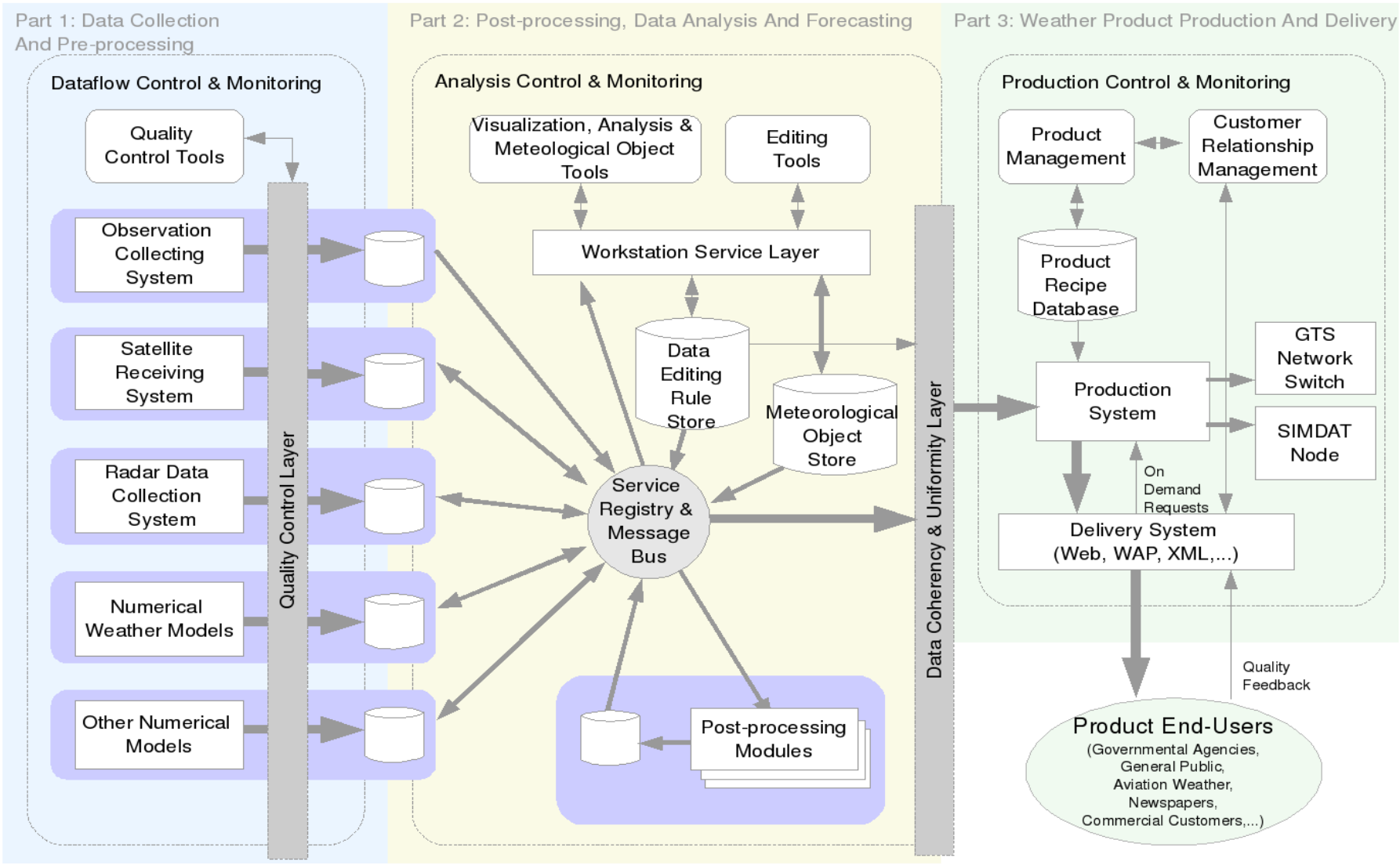
- **A large number of systems taking part in the production**
  - Various platforms: Unix, Linux, Windows
  - Numerous technologies, languages etc.
- **Some form of integration needed; would an ESB help?**

Some architectural plans

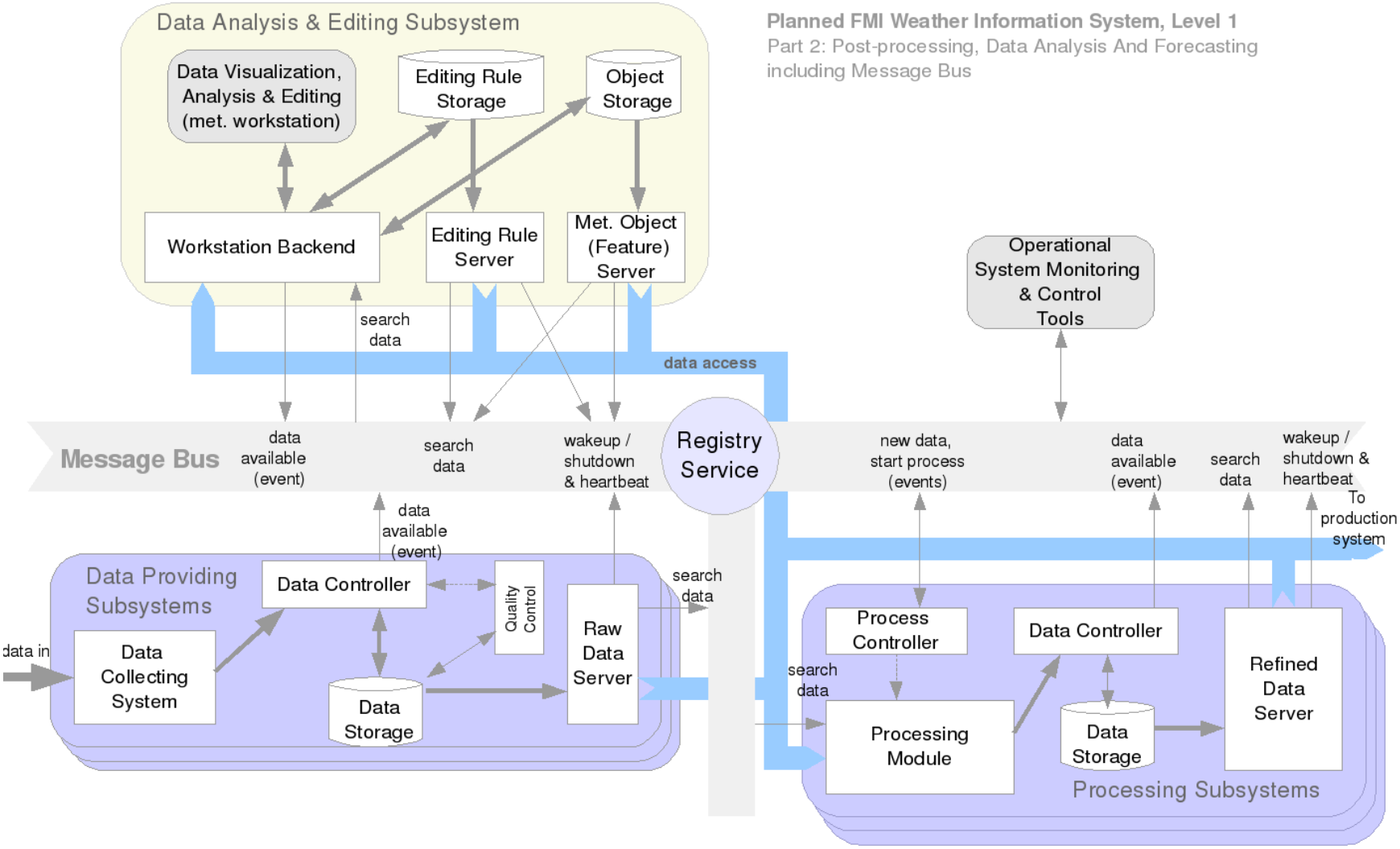
→

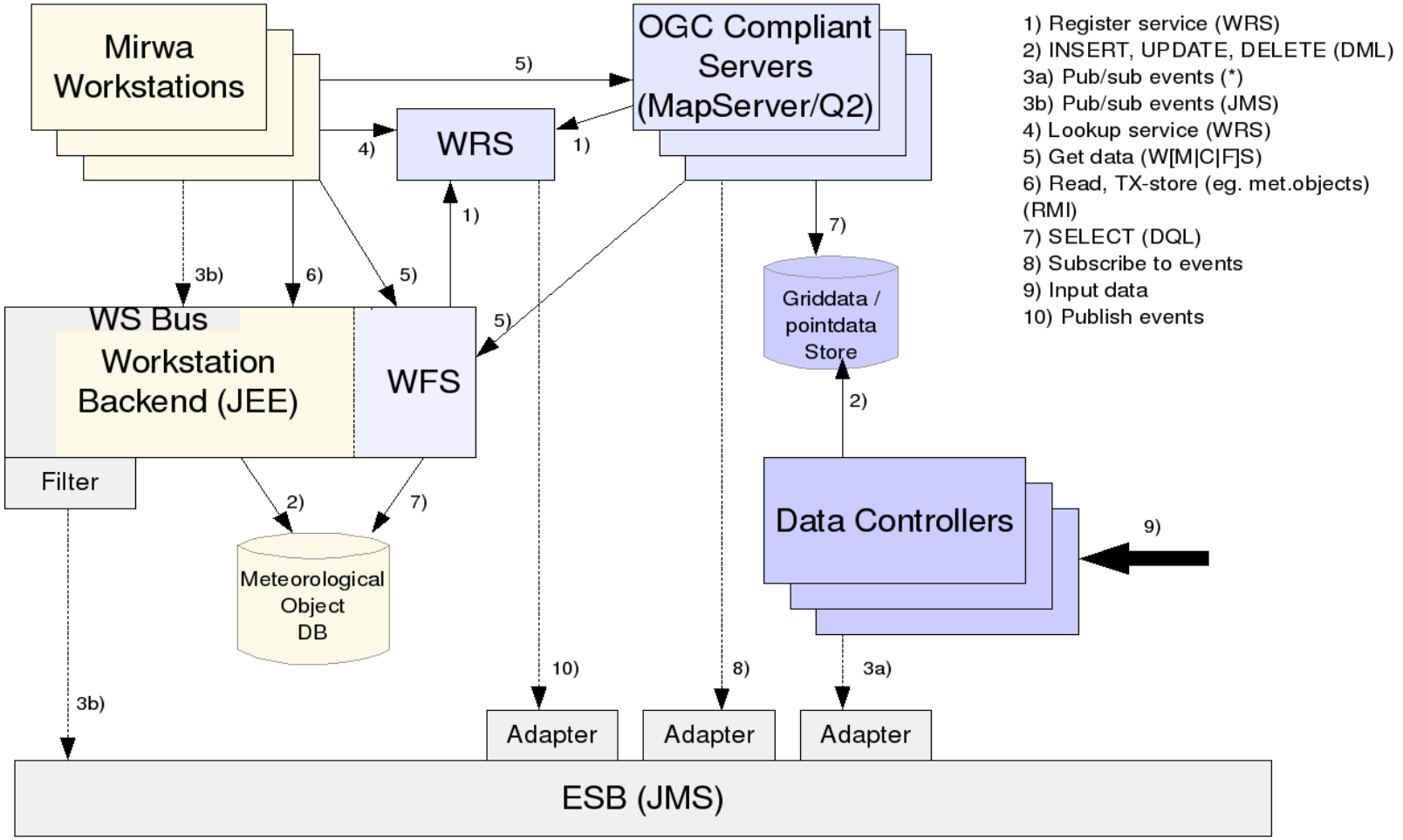
ILMATIETEEN LAITOS
METEOROLOGISKA INSTITUTET
FINNISH METEOROLOGICAL INSTITUTE

Planned FMI Weather Information System, Level 0



Planned FMI Weather Information System, Level 1
Part 2: Post-processing, Data Analysis And Forecasting including Message Bus





1) Register service (WRS)
2) INSERT, UPDATE, DELETE (DML)
3a) Pub/sub events (*)
3b) Pub/sub events (JMS)
4) Lookup service (WRS)
5) Get data (W[M|C|F]S)
6) Read, TX-store (eg. met.objects) (RMI)
7) SELECT (DQL)
8) Subscribe to events
9) Input data
10) Publish events

ESB pilot at the FMI: Mission

- **To build a proof-of-concept: use an ESB as a notification channel of events concerning new data**
- **To find and assess possible benefits of using an ESB as a part of FMI's production environment**
- **To evaluate ease of deployment, ease of configuration and usage of two ESB implementations (Mule, Apache ServiceMix)**
  - The two products are Open Source
  - ServiceMix not evaluated due to lack of time
  - Will keep an eye on Spring Integration, that is still very much in the making
- **To evaluate necessary integration work on already existing software**
- **Performance and maintenance issues decidedly left out of scope at this point**

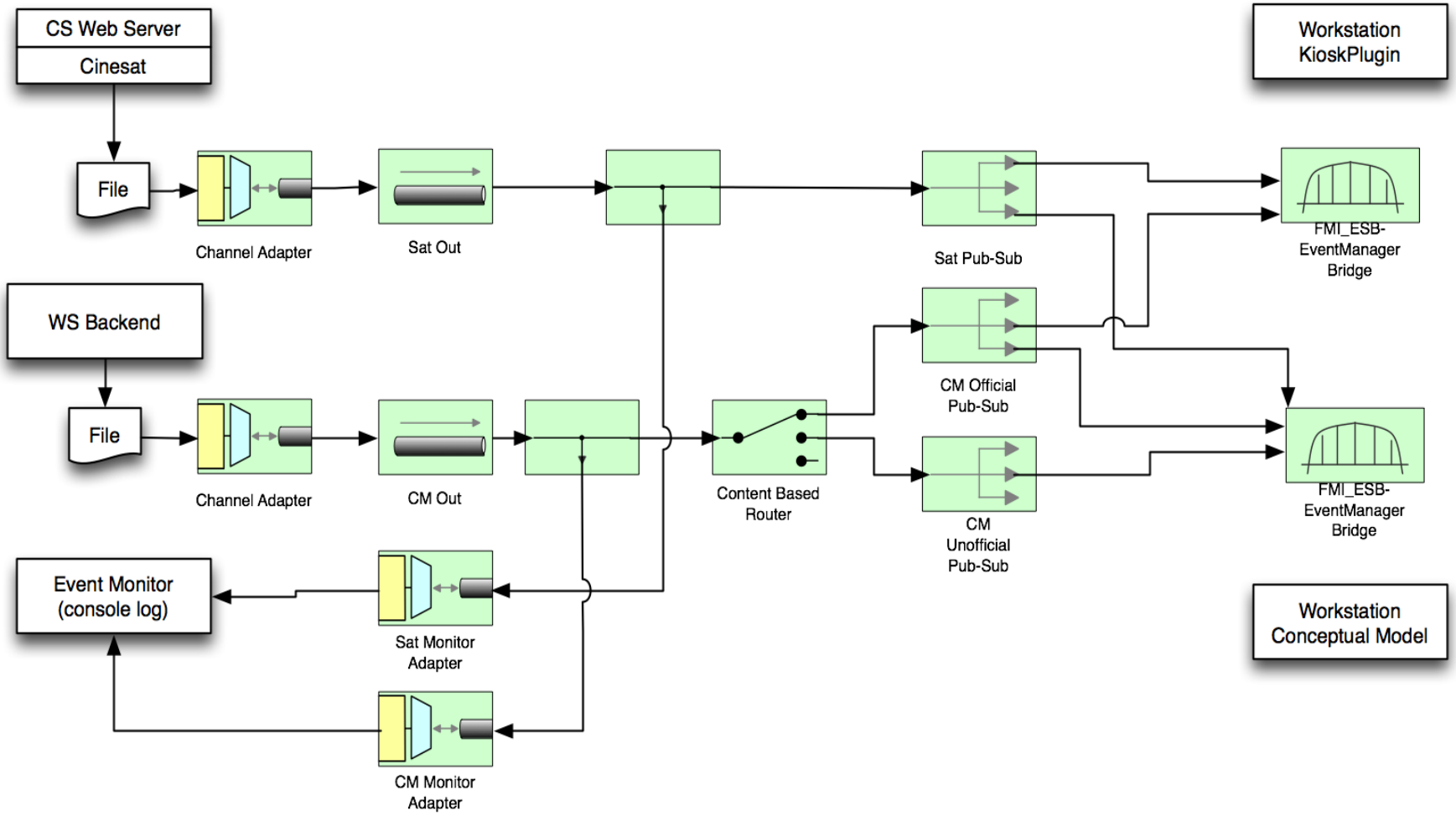
FMI ESB pilot: components

- **Mule: an Open Source ESB, the back bone**
- **ActiveMQ, an Open Source JMS implementation: the primary communication channel**
  - Handles persistence of messages for guaranteed delivery
- **Home grown (Mule) components:**
  - Two transformers to handle transforming of a properties file into a canonical message object
  - A filter to assist in routing canonical messages based on their content

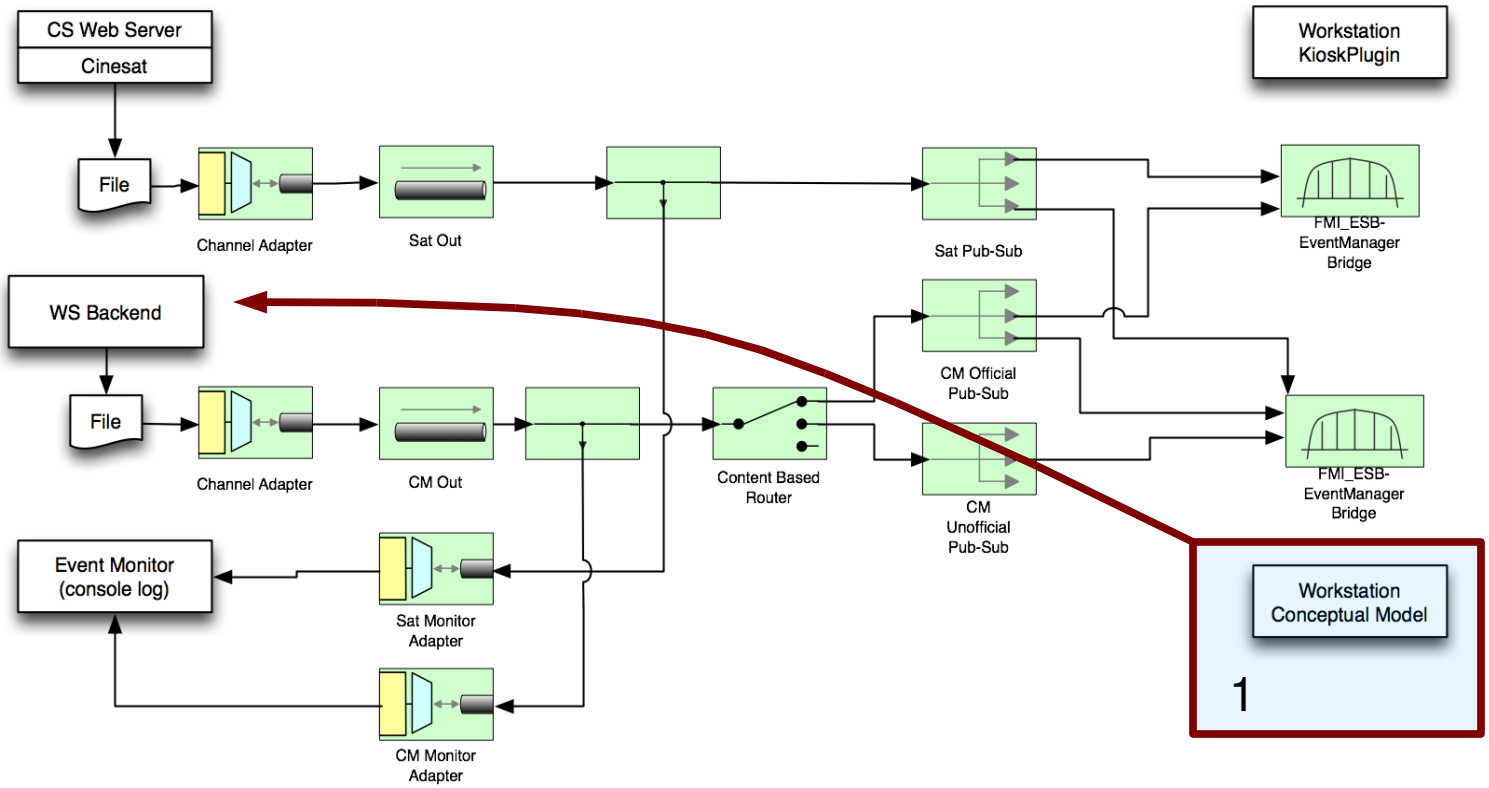
FMI ESB pilot: characteristics

- **We use the bus for passing around *messages about data, not the data itself***
- **None of the participant applications know about Mule:**
  - Input from CineSat & storage servlet via files
  - Java Message Service (JMS) as a communication channel for Smartmet II workstation; standard JMS (JMS *"is a messaging standard that allows application components [- -] to create, send, receive, and read messages. It enables distributed communication that is loosely coupled, reliable, and asynchronous"* (Sun JMS Homepage)
- **The data producing applications don't know**
  - Who is supposed to receive the message
  - Whether anyone ever consumes the event

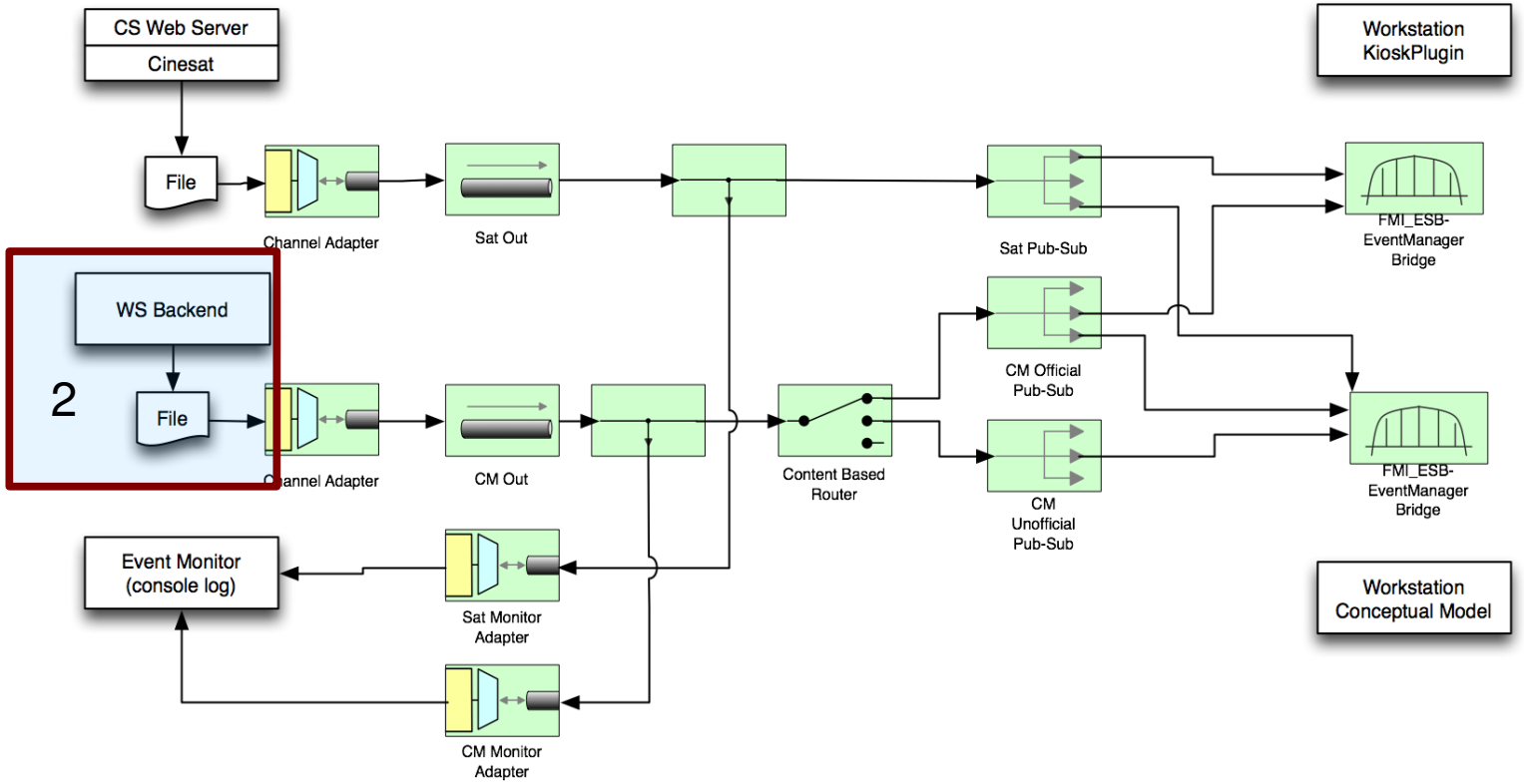
FMI ESB pilot: overview



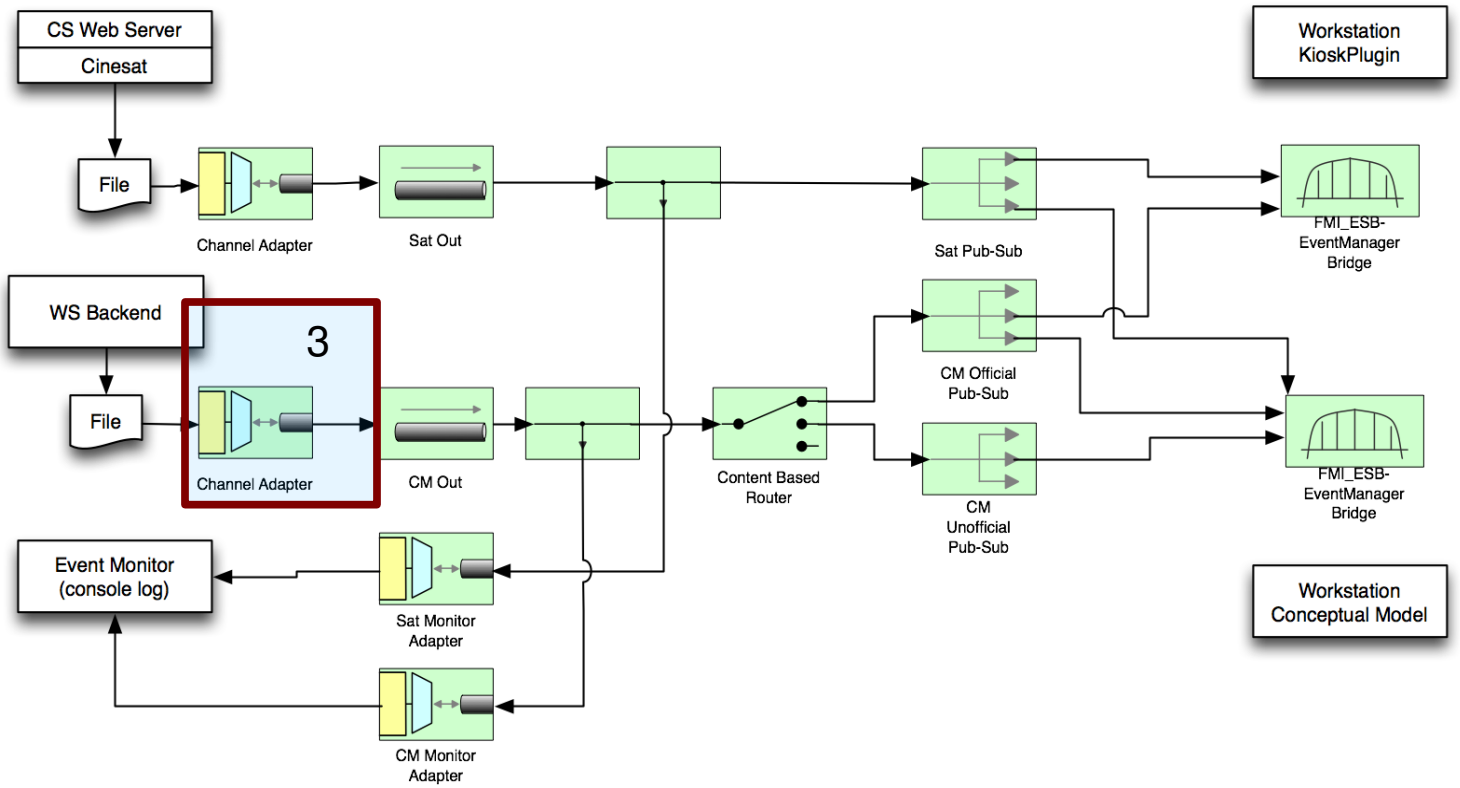
Simple scenario: notify others of storing a set of meteorological objects



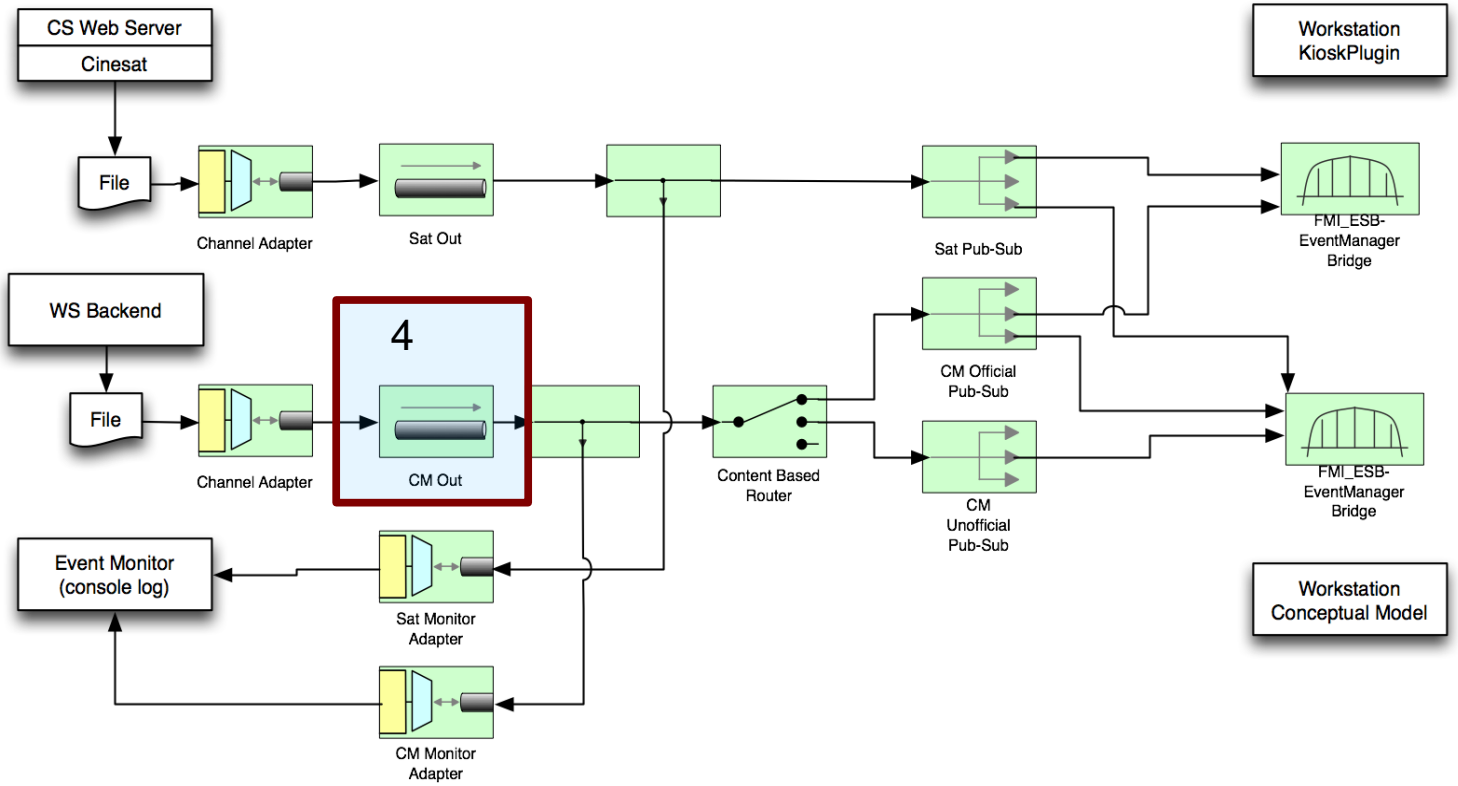
1) A user has edited a set of meteorological objects and asks the Smartmet II workstation to store them. The workstation sends the file(s) to a storage servlet



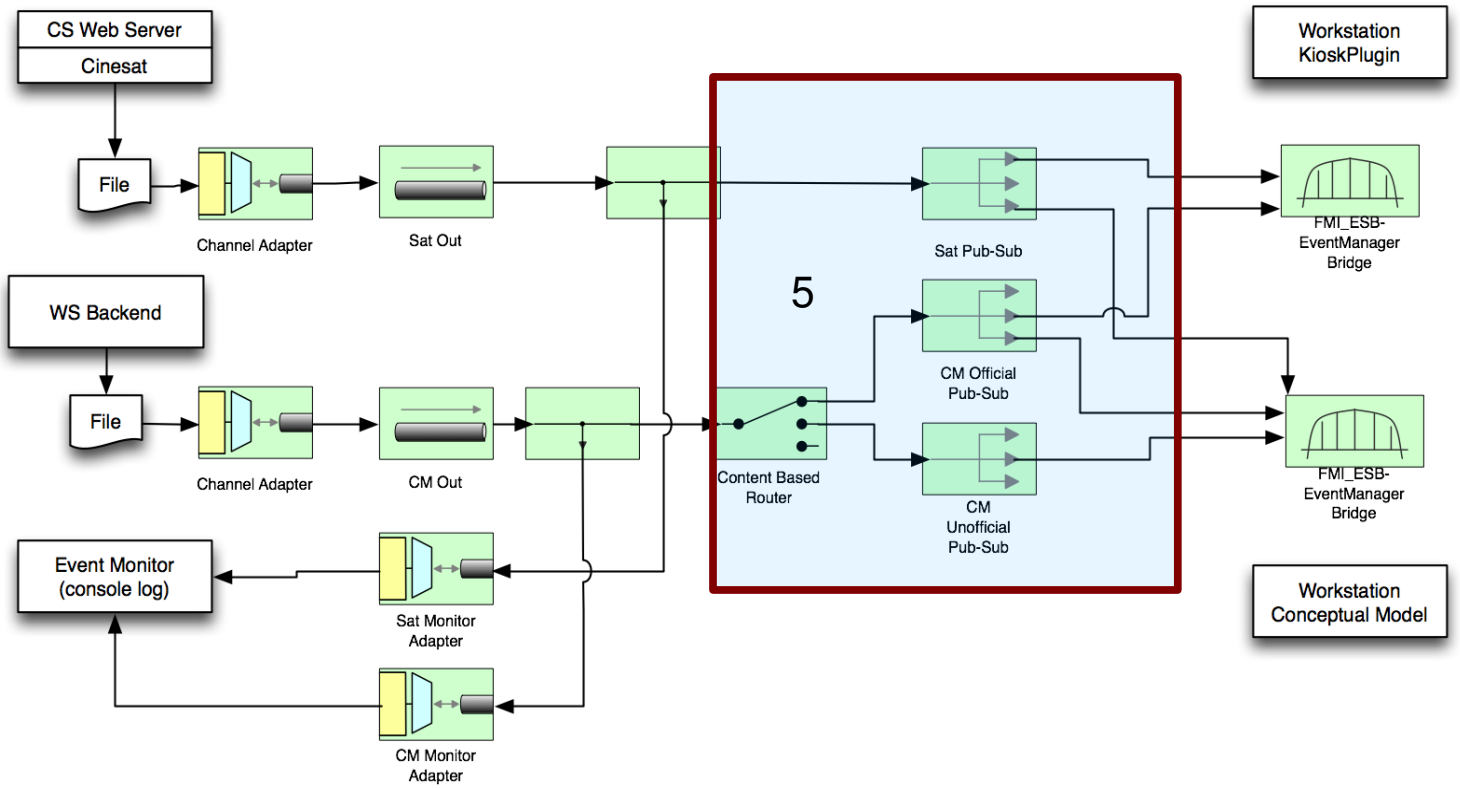
2) The servlet stores the file(s) and writes a descriptive .properties file of the storage event



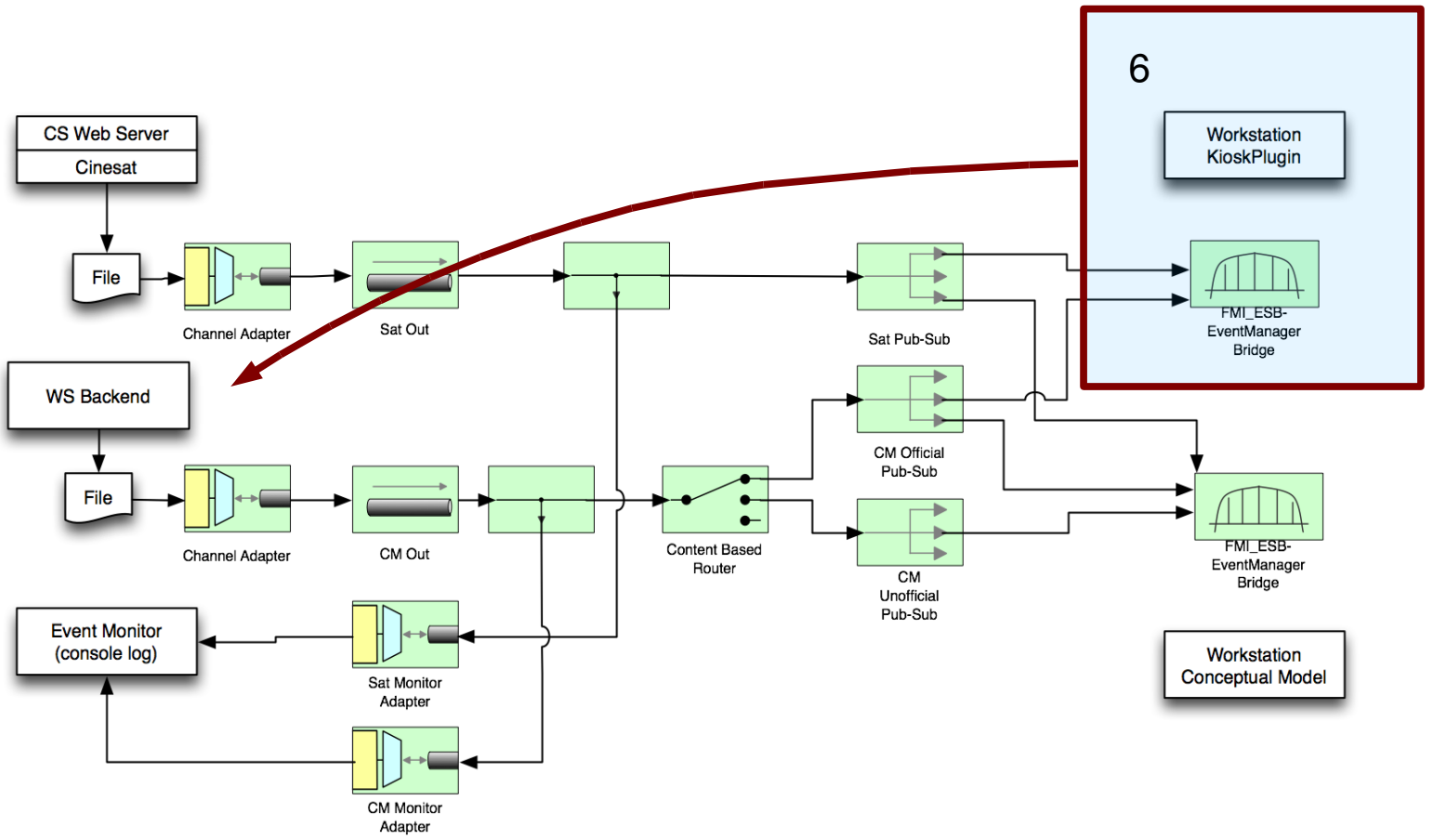
3) Mule reads the .properties file and converts it into a canonical message...



4) ...puts it into a JMS topic and



5) routes it according to routing rules to different subscribers



6) Kiosk-flavoured Smartmet II Workstation receives the message, downloads the new set of Meteorological objects and updates its display to show it

What actually happened?

- **We conveyed a message of a single storage event to one possible consumer of such events**
  - It is worth noting that any number of such consumers may (and generally do) exist
- **From the storage agent's point of view this event was "fire-and-forget"**
- **We managed to monitor the message delivery by wire-tapping the message channel**

FMI ESB pilot: simple scenario II (inform the Workstation of a new satellite image)

1) **CineSat produces a PNG image of some satellite data and writes a descriptive .properties file about the said PNG**
2) **Mule**
   a) reads the file,
   b) converts it into a Canonical Message and
   c) sends it into a JMS Topic to be
   d) routed to the Workstation
3) **Kiosk Workstation receives the message and does nothing currently, but should**
   - present a satellite image view with the corresponding analysis, if available; aggregation is a big part of message routing

FMI ESB pilot participants and integration work done

- **Servlet for storing conceptual model XML files**
  - Added a filter to create and write .properties files based on the storage servlet's response ~ 1 day
- **CineSat**
  - Modified a shell script to produce .properties files of newly created images ~ 1 day
- **Smartmet II workstation**
  - JMS support was already built in the workstation
    - Tweaked bridging from canonical message format into workstation's native event model ~ 2 days
- **Event monitor**
  - Used Mule's console log
- **Three in-house built Mule components (two transformers and a filter) ~ half a day**

Summary: these integration steps were quite cheap

Conclusions

- **An ESB might really help in certain scenarios**
- **Mule looks to be a quite promising ESB implementation**
  - Mule also has some convincing references
- **Integration work was pretty cheap, typically a couple of days or a few hours per application**

Conclusions: testing

- **Easy to develop separately testable components**
- **Difficult to test the whole with truly meaningful results**
  - Are we testing the right thing?

Thoughts and concerns

- **Event Driven programming model**
  - Messaging integration is by nature event driven
- **Technology risks**
  - Handling of unpredictable loads may be a problem
  - A good candidate to be a single point of failure
- **Some coupling still**
  - You need to be very careful in versioning your canonical message type

Experiences with Mule

- **Configuration rather clear, especially with the 2.0.* version's XML Schema based syntax → proper XML tools can assist a whole lot**
- **Documentation quality ranges from poor to excellent (2.0.* documentation still a work in progress)**
  - Javadoc, code, Google
- **Extending Mule is extremely easy**
- **Problems with JMS Connector; workaround found**

Experiences with Mule: configuration

- **Declarative forming of message routes helps bring clarity to a complex system; all logical routing can be kept in a single file**
- **Spring-based configuration is a huge plus (for those who like Spring)**
- **Visualizations are easy to create from the configuration XML**

```
<custom-transformer name="properties-to-conceptualmodeldata-event"
  class="fi.fmi.common.esb.mule.transformer.message.PropertiesToBasicMessageTransformer">
  <spring:property name="sourceId">
    <spring:bean class="fi.fmi.common.naming.ComponentIdentifier">
      <spring:constructor-arg value="conceptualmodel" />
      <spring:constructor-arg value="mirri-conceptualmodel" />
      <spring:constructor-arg value="0.2-SNAPSHOT" />
      <spring:constructor-arg value="kopla" />
    </spring:bean>
  </spring:property>
  <spring:property name="wrappedObjectFQN"
    value="fi.fmi.mirri.core.event.data.DataEvent" />
</custom-transformer>

<file:endpoint name="conceptualmodel-files-in"
  connector-ref="non-streaming-file-connector"
  address="${cinesat-in-directory-url}"
  transformer-refs="propertiesfile-to-properties properties-to-conceptualmodeldata-event" />

  <service name="conceptualmodel-in">
    <inbound>
      <inbound-endpoint ref="conceptualmodel-files-in" />
    </inbound>
    <outbound>
      <filtering-router>
        <jms:outbound-endpoint ref="conceptualmodel-official-topic" />
        <filter ref="conceptualmodeldata-official" />
      </filtering-router>
      <outbound-pass-through-router>
        <jms:outbound-endpoint ref="conceptualmodel-topic" />
      </outbound-pass-through-router>
    </outbound>
  </service>

  <service name="monitoring">
    <inbound>
      <jms:inbound-endpoint ref="satellite-topic" />
      <jms:inbound-endpoint ref="conceptualmodel-official-topic" />
      <jms:inbound-endpoint ref="conceptualmodel-topic" />
    </inbound>
    <outbound>
      <outbound-pass-through-router>
        <jms:outbound-endpoint ref="monitor-topic" />
      </outbound-pass-through-router>
    </outbound>
  </service>
```

What next?

- **Seems that we will employ an ESB, possibly Mule**
  - At least for Smartmet II workstation's benefit
    - Dump/debug utility for analysing error situations
    - User feedback channel via JMS
    - Chat for Meteorologists
  - Will probably try out ServiceMix, Spring Integration
- **Serious testing ahead, as well as evaluation of maintenance costs**
- **Possibly a trial to orchestrate an execution chain from raw data into an (image) Product**

Resources

- http://en.wikipedia.org/wiki/Enterprise_service_bus
- Mule home: http://mule.mulesource.org/
- Sun JMS home: http://java.sun.com/products/jms/
- Apache ServiceMix home: http://servicemix.apache.org/
- Spring integration home: http://www.springframework.org/spring-integration
- Enterprise integration patterns: http://www.enterpriseintegrationpatterns.com/

Thank you

(Easy) questions?

Contact: pekka.rantala@fmi.fi