



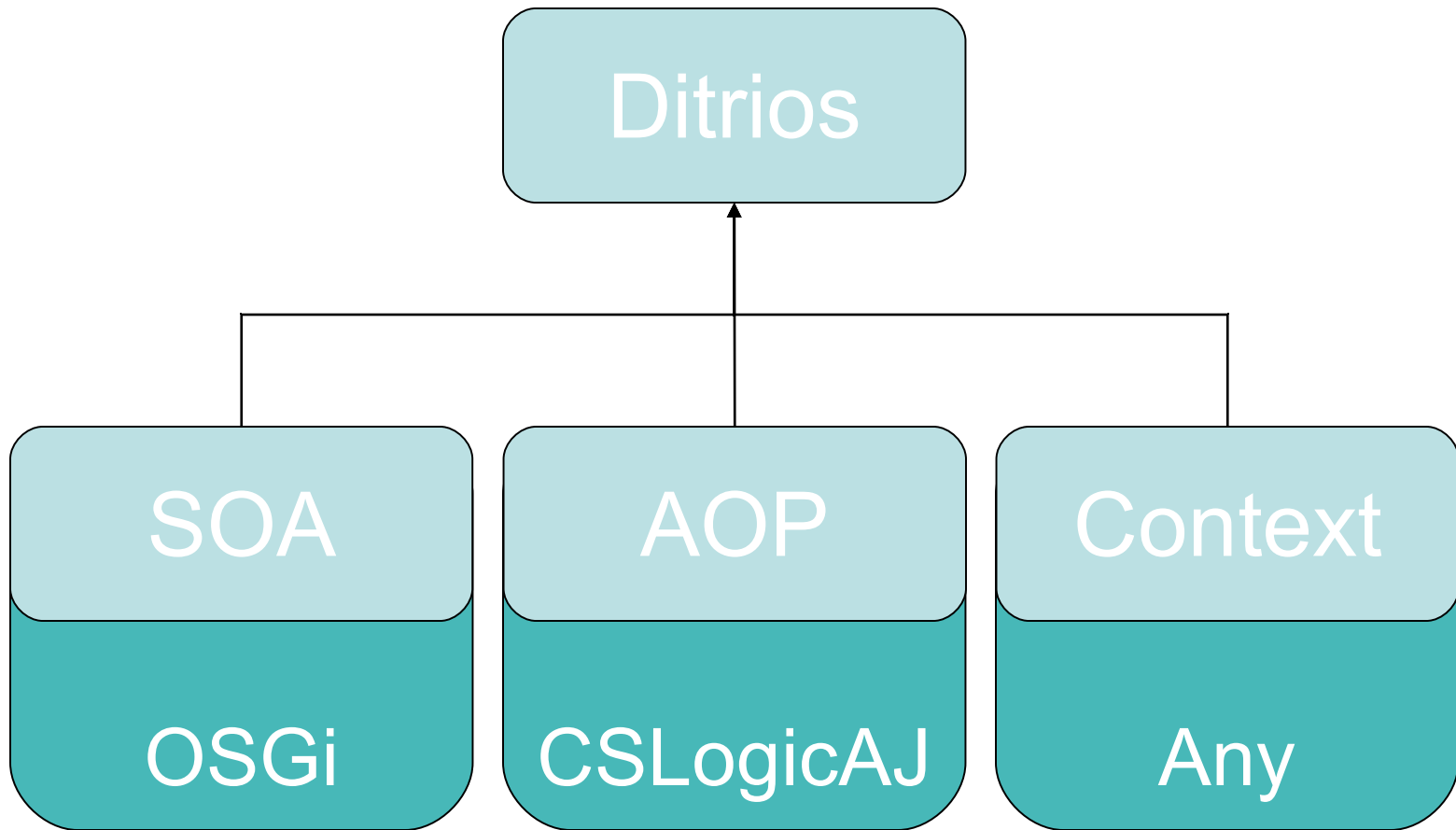
# *Ditrrios*

A context-aware AOP-enabled  
SOA framework

A short example-driven overview

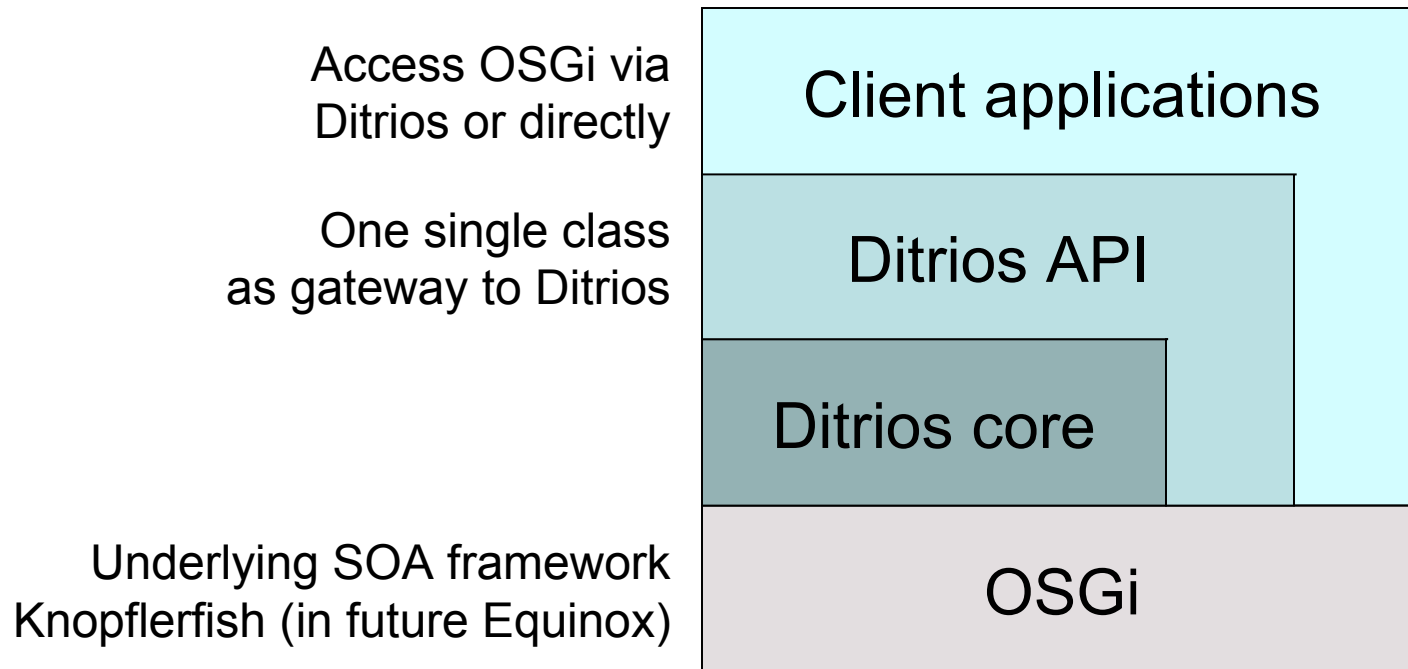
# Overview

---



# Framework

---



# Ditrios features

---

- Encapsulates complete Service management
  - Developers can concentrate on business logic
  - Ditrios deals with middleware concerns
    - Service searching, tracking, providing
    - Stale reference problem
- Services can be adapted due to AOP
  - Synchronous and asynchronous adaptation
  - Decoration, Strategy change
- Comprises context information
  - Adaptations can be context-sensitive

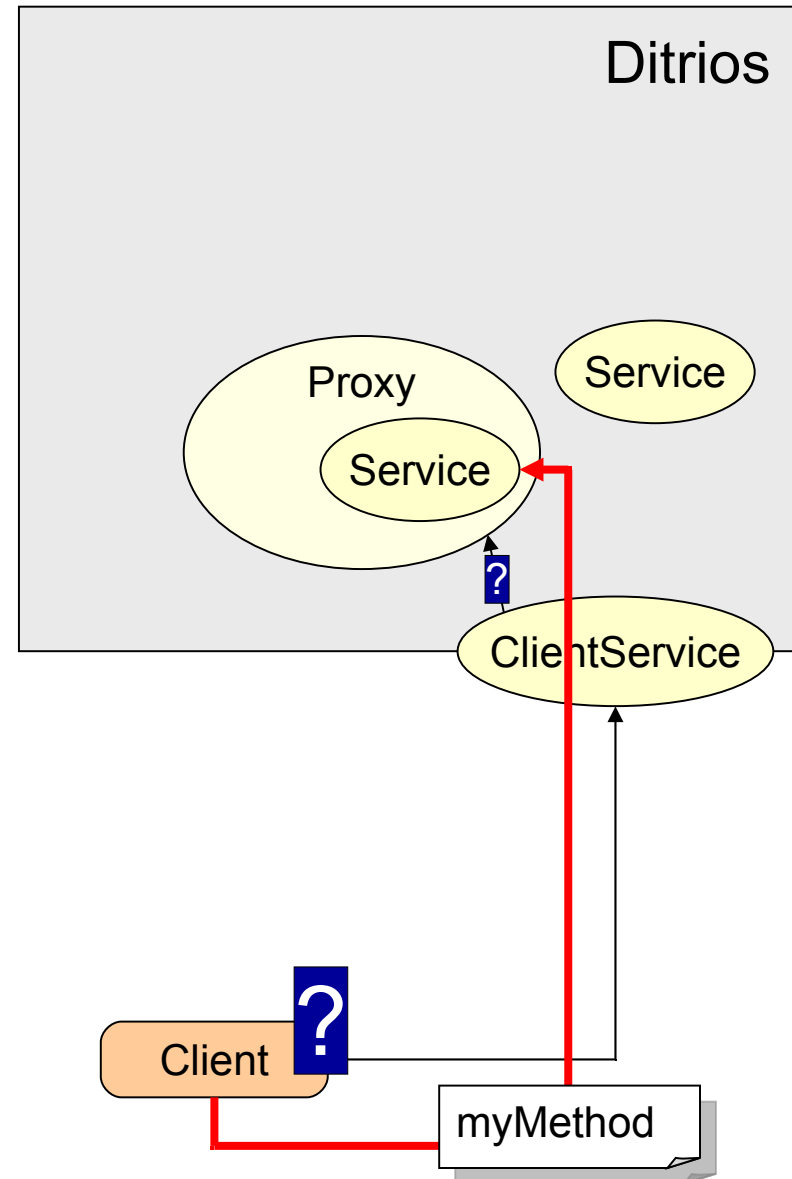
# Service management (I)

---

- The complete service lifecycle is managed
- Clients only request their services
  - and can use them in turn
  - one line of code via simple Ditrios API
- Whiteboard pattern (aka “Hollywood principle”)
  - services are simply registered
  - no need to deal with
    - Trackers resp. Listeners
    - services appearing / disappearing
    - Stale reference problem

# Service management (II)

- *ClientService* is the gateway
- Search request → Service found
- Proxy creation for service;  
bind search request
- Proxy encapsulates **all** matching services
- the first found service is the standard service
- all method calls are delegated to the standard service



# Ditrios AOP

---

- **Synchronous adaptation**
  - Program flow dependent
  - triggered by joinpoints
- **Asynchronous adaptation**
  - Program flow independent
  - triggered by context changes
- **Decoration (only synchronously)**
  - interim-invocation of code
  - dynamic before, after, onchange advice
- **Strategy change (synchronously & asynchronously)**
  - Live switch to another service
  - Service must be compatible (automatically checked)

# Context-Awareness

---

- Context is any persistable context data like
  - user preferences
  - physical sensor data
  - system internal data
- Context is stored in Prolog database as facts
- New aspect language is context-aware
  - Context can be evaluated within pointcuts

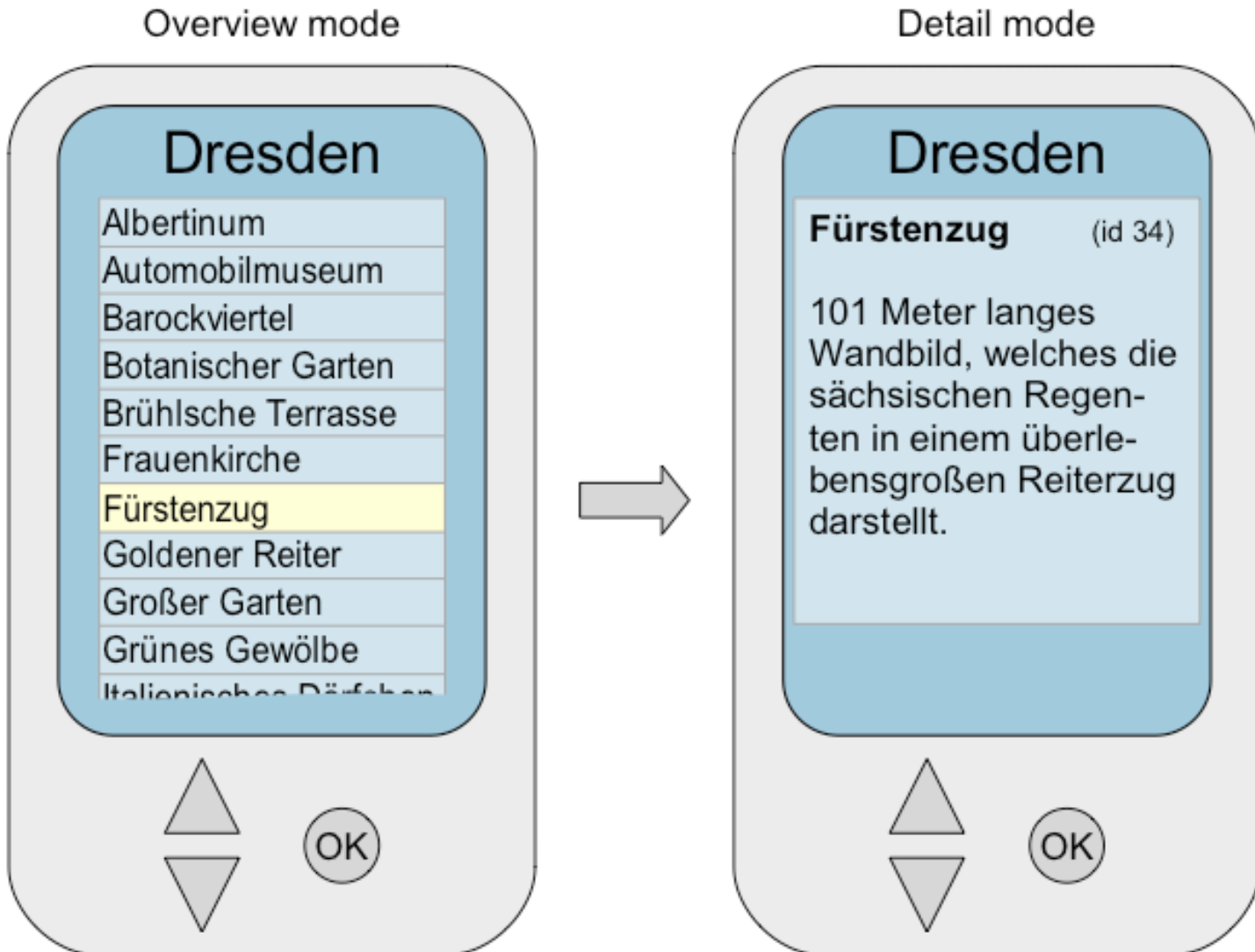
# CSLogicAJ

---

- extends LogicAJ
  - dynamic, context-sensitive, SOA-enabled
- Syntax extensions
  - ***onchange*** advice
  - ***dynamic*** modifier
  - built-in service and context pointcuts

Demo

# Example: A simple city guide



# City guide feature overview

---

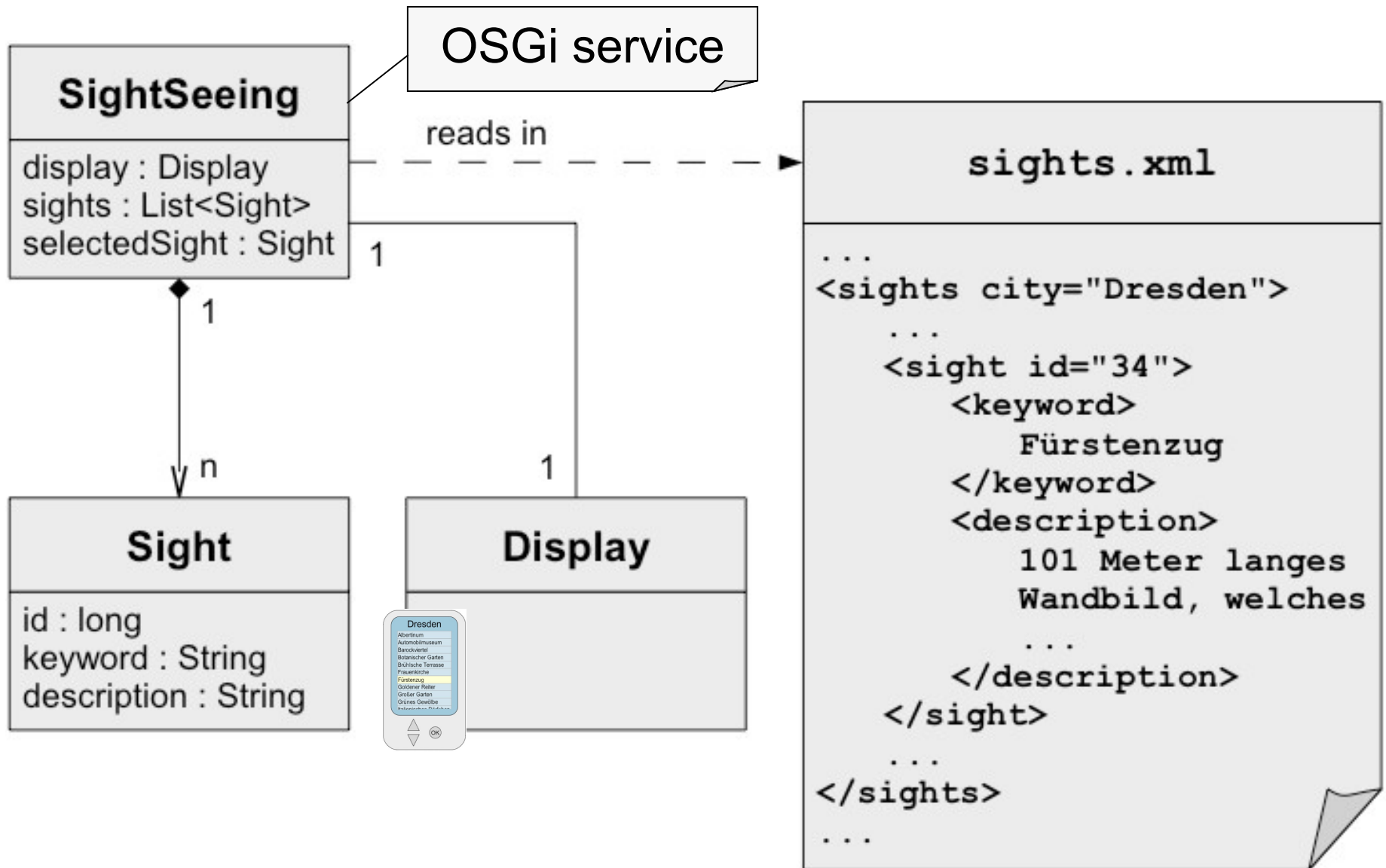
- Displays a list of sights (overview mode)
  - *OK* button displays sight details (detail mode)
  - Sights are stored in an XML file (sights.xml)
  - Sights are displayed in alphabetical order
  - managed via a ***SightSeeing*** service
- 
- Very simple design
  - Minimal functionality
  - Clear API

# City guide API

---

<code>void loadSights();</code>	Reads in the <i>sights.xml</i> file holding the sight information; creates a <i>Sight</i> object for each data set; all objects are stored in the previously emptied sight list.
<code>void displaySights();</code>	Displays the list of sights; is invoked after activating the service resp. when the <i>OK</i> button is pressed in the detail mode.
<code>List getSights();</code>	Returns the list of sights; is used by <i>displaySights()</i> .
<code>void prevSight();</code>	Selects the previous sight in the list; is invoked when the 'arrow-up' button is pressed.
<code>void nextSight();</code>	Selects the next sight in the list; is invoked when the 'arrow-down' button is pressed.
<code>void displaySight();</code>	Displays the sight details; is invoked when the <i>OK</i> button is pressed in the overview mode.

# Class diagram



# Extension: make the city guide context-aware

---

- Context-driven reordering of sights
  - Order sights according to the user's position
- With the help of Ditrios there's no need to change the original application
- Aspects do the work!

# What is needed? (I)

---

- GPS sensor
- **Converter** service
  - polls or receives the sensor data
  - stores them as context facts in the fact base
- a second XML file (positions.xml)
  - maps the sights to their GPS positions
- **LocationSorter** service
  - reads in the positions.xml file
  - provides a method which sorts the sights according to the current user's position

## What is needed? (II)

---

- ***after advice*** (location sorter)
  - applied after the call of the `loadSights()` method
  - passes *SightSeeing* reference to *LocationSorter*
  - enables *LocationSorter* to reorder the sights
- ***onchange advice*** (update sorting)
  - activates reordering according to current position
  - triggered anytime the position changes
- ***Aspect bundle*** encapsulating the aspects

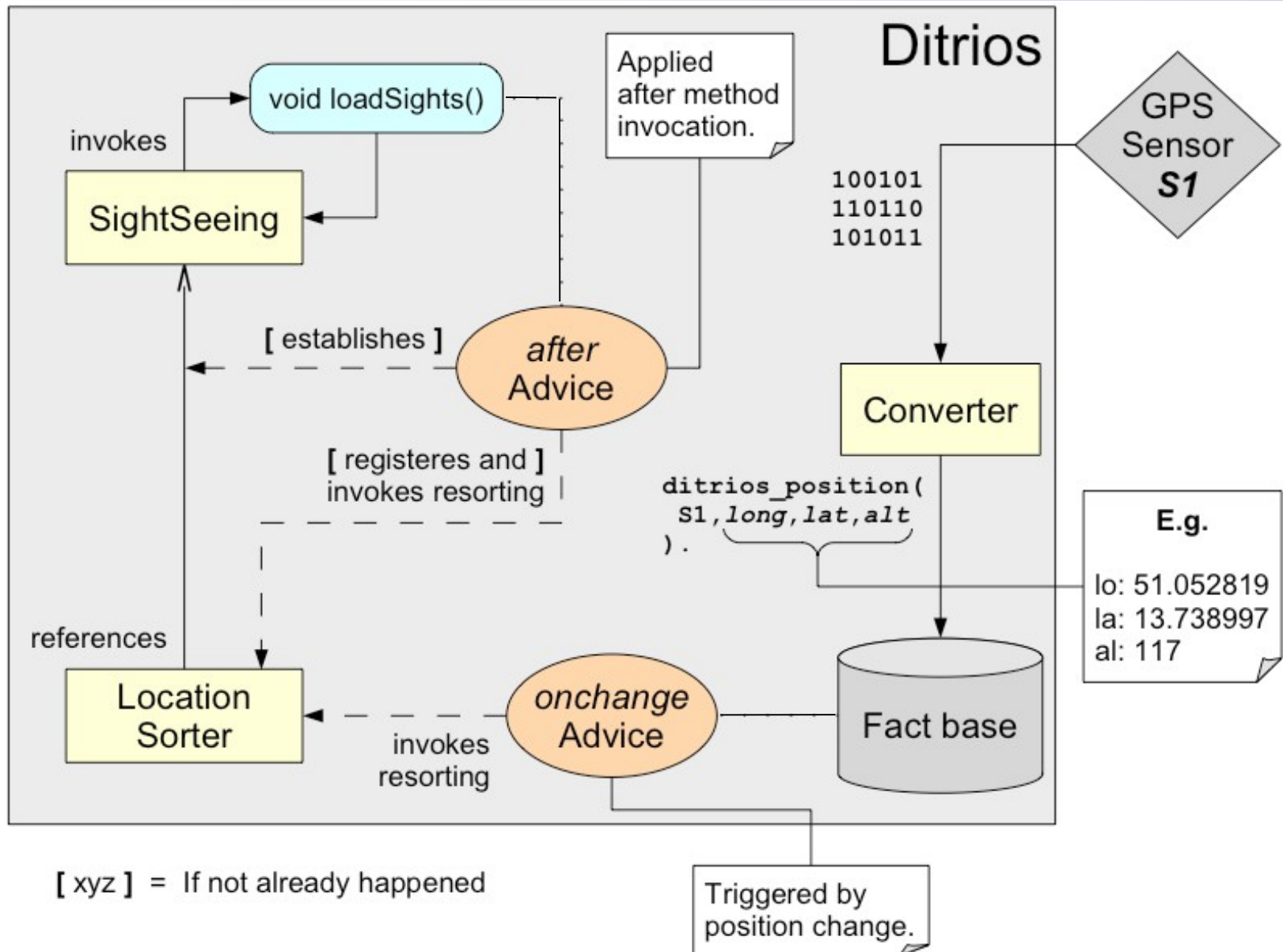
# Finally, ...

---

- ... just register
  - the ***Converter*** and ***LocationSorter*** services
  - to activate the position dependent sight ordering



# The complete context-sensitive system



# The location sorter after advice

```
1  after initLocSorter(SightSeeing service) :
2    execution( void SightSeeing.loadSights() ) &&
3    target(service) &&
4    position(?PDASensor, ?Long, ?Lat, ?Alt) &&
5    osgiService(?SorterId, ?LocSorter) &&
6    equals(?LocSorter, "org.foo.LocationSorter") &&
7    serviceIdMapping(?SorterId, ?OsgiId)
8  {
9    // get Ditrios facade
10   DitriosFacade df = thisJoinPoint.getDitriosFacade();
11
12   // get location sorter service
13   ?LocSorter sorter = (?LocSorter)
14     df.getRegistryOffice().
15     getServiceById(?"OsgiId");
16
17   // pass the reference to the sight seeing service
18   sorter.setSightSeeingRef(service);
19
20   // (re)sort the list according to
21   // the current position
22   sorter.sort(
23     new Double(?"Long").doubleValue(),
24     new Double(?"Lat").doubleValue(),
25     new Double(?"Alt").doubleValue());
26 }
```

# The update sorting onchange advice

```
1  onchange updateSorting(?Long, ?Lat, ?Alt) :
2    position(?PDASensor, ?Long, ?Lat, ?Alt) &&
3    osgiService(?SorterId, ?LocSorter) &&
4    equals(?LocSorter, "org.foo.LocationSorter") &&
5    serviceIdMapping(?SorterId, ?OsgiId)
6  {
7    // get Ditrios facade
8    DitriosFacade df = thisJoinPoint.getDitriosFacade();
9
10   // get location sorter service
11   ?LocSorter sorter = (?LocSorter)
12     df.getRegistryOffice().
13     getServiceById(? "OsgiId");
14
15   // (re)sort the list according to
16   // the current position
17   sorter.sort(
18     new Double(? "Long").doubleValue(),
19     new Double(? "Lat").doubleValue(),
20     new Double(? "Alt").doubleValue());
21 }
```

# The aspect

---

```
1 public dynamic aspect LocationSorter
2 {
3     after initLocSorter ( SightSeeing service ) :
4         /* ... */
5     {
6         // ...
7     }
8
9     onchange updateSorting ( ?Long , ?Lat , ?Alt ) :
10        /* ... */
11    {
12        // ...
13    }
14
15    /* optional helper methods ... */
16 }
```

# Register the *SightSeeing* service

---

- the registration of an OSGi service
  - can be carried out anywhere in the OSGi bundle

```
1  /* ... */
2
3  SightSeeing sightSeeing = new SightSeeing();
4  Hashtable properties = new Hashtable();
5  // properties.put("put props", "if you want");
6  bundleContext.registerService(
7      sightSeeing.class.getName(), sightSeeing, properties);
8
9  /* ... */
```