RESTful DDS

Expanding the reach of the information backbone

Reinier Torenbeek
Senior Solutions Architecht
reinier.torenbeek@prismtech.com

OpenSplice DDS

PRISMTECH

RESTful DDS

- Introduction
- What does RESTful mean?
- Why RESTful DDS?
- An Open Source RESTful DDS Webservice
  - Design
  - Implementation
  - Demonstration
- Questions?

RESTful DDS

- Introduction
  - What does RESTful mean?
  - Why RESTful DDS?
  - An Open Source RESTful DDS Webservice
    - Design
    - Implementation
    - Demonstration
  - Questions?

Introduction

There is no *the* RESTful DDS solution or API

Goals of this talk:

- Get you introduced into the concepts of REST in the context of DDS
- Give you a feel for what a RESTful DDS API looks like
- Guide you through an example implementation
- Demonstrate the result

RESTful DDS

- Introduction
- What does RESTful mean?
- Why RESTful DDS?
- An Open Source RESTful DDS Webservice
  - Design
  - Implementation
  - Demonstration
- Questions?

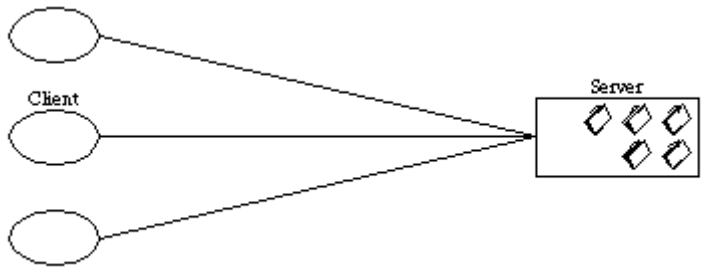


Figure 5-3. Client-Stateless-Server

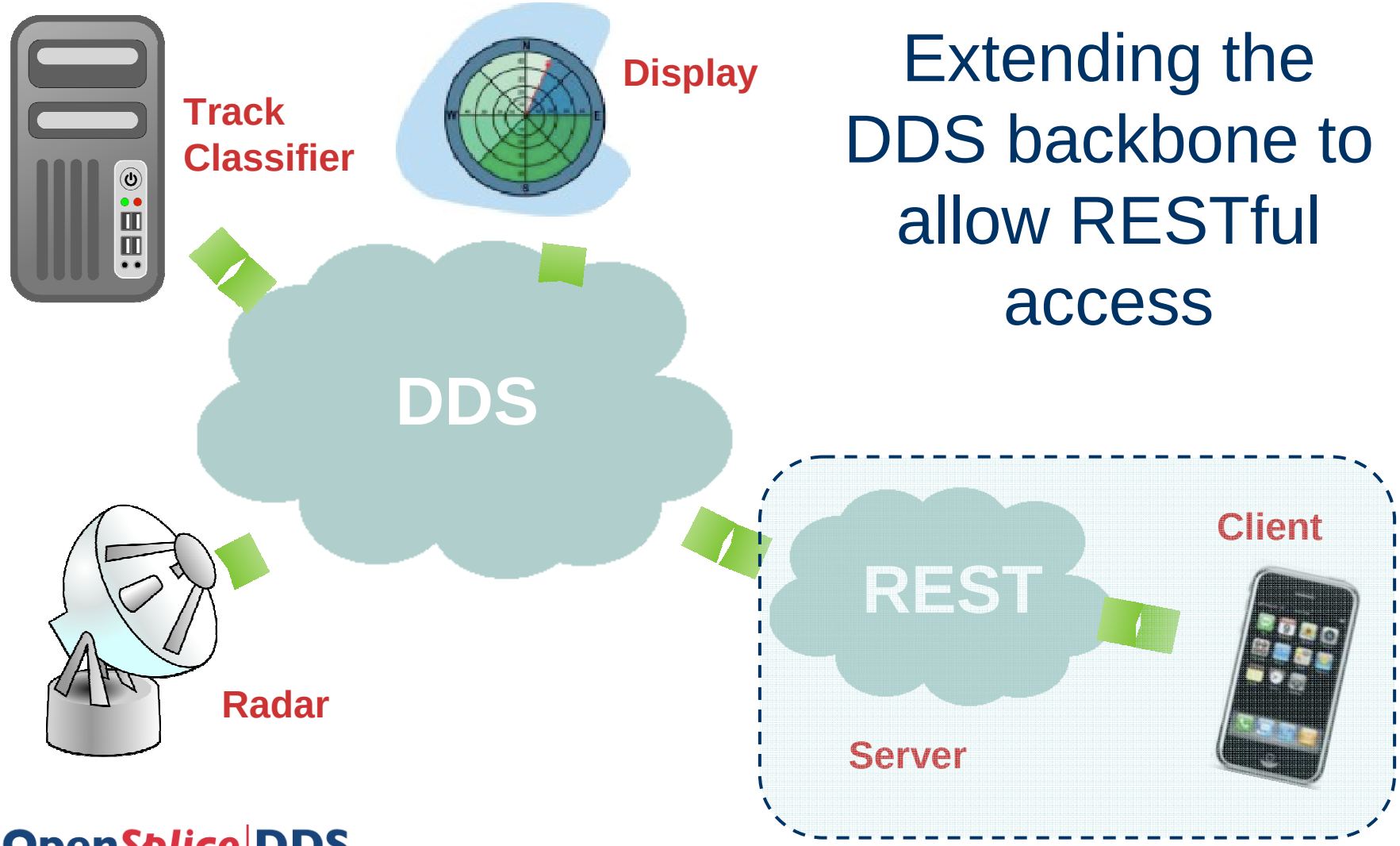
What does RESTful mean?

REpresentational State Transfer

- Architectural style for distributed systems
- Defined by “engineering guidelines”
- Not limited to Webservices

RESTful systems have an architecture and interaction patterns in line with the REST guidelines

What does RESTful mean?



Extending the DDS backbone to allow RESTful access

OpenSplice DDS

What does RESTful mean?

REST guidelines relevant for DDS:

- Separation of concerns
  - e.g. Server does DDS, Client does GUI
- Client-Stateless-Server (CSS)
  - e.g. Server manipulates named DDS entities which can be accessed from any session
- Uniform interface
  - e.g. HTTP methods, URL paths, JSON bodies
- Code-On-Demand
  - e.g. JavaScript transferred from Server to Client

RESTful DDS

- Introduction
- What does RESTful mean?
- Why RESTful DDS?
- An Open Source RESTful DDS Webservice
  - Design
  - Implementation
  - Demonstration
- Questions?

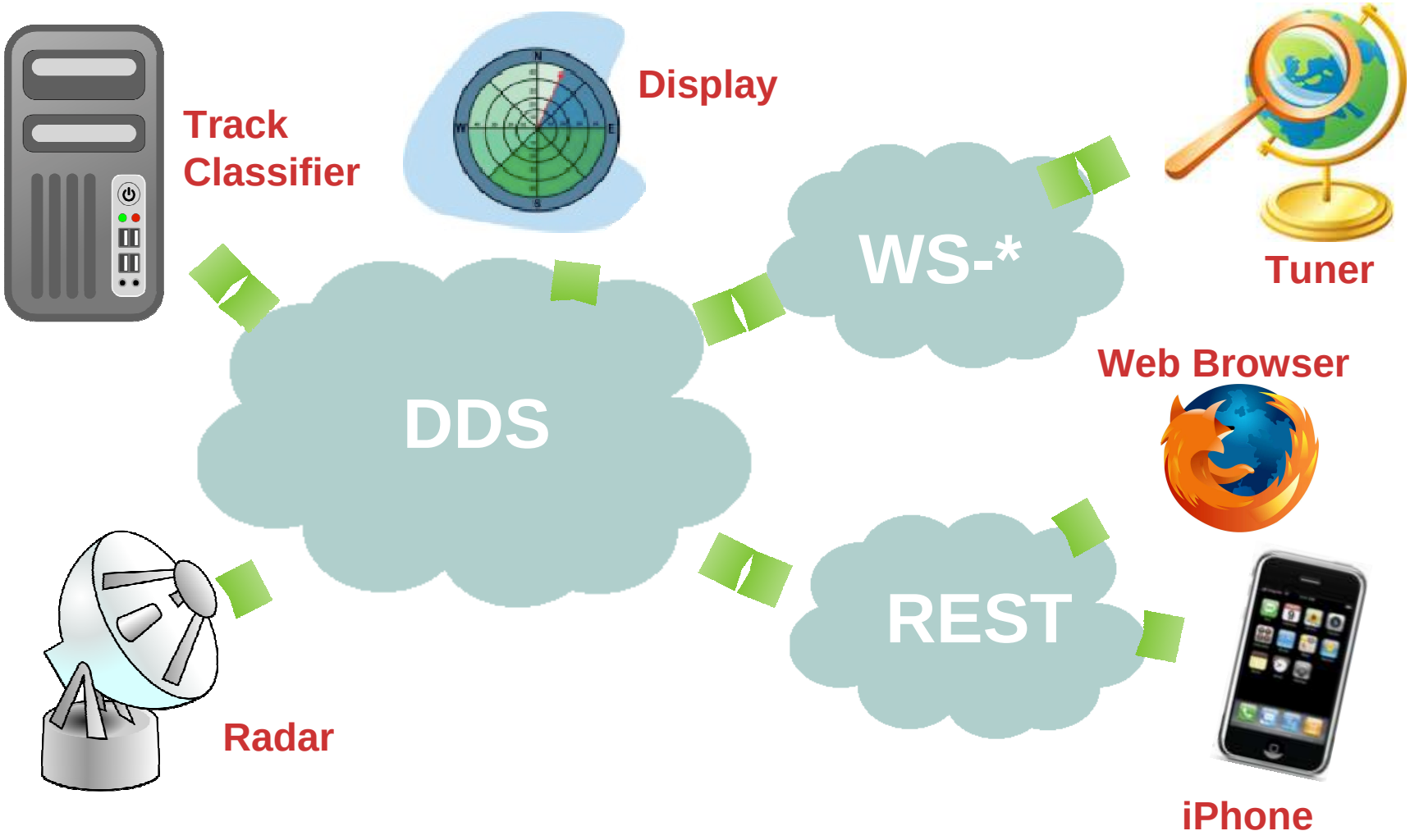
Why RESTful DDS?

- Very lightweight clients can participate
  - No need to install software with Code-On-Demand
  - Resource usage driven by client only
  - Subset of DDS features so limited bandwidth
    - No metadata injection or discovery protocols
- DDS backbone extended but not impacted
  - Allows low bandwidth or intermittent connections
  - Resource constraints do not propagate
- “Best of both worlds”
- Appears in OMG’s Web-enabled DDS RFP

RESTful DDS

- Introduction
- What does RESTful mean?
- Why RESTful DDS?
- An Open Source RESTful DDS Webservice
  - Design
  - Implementation
  - Demonstration
- Questions?

RESTful DDS Webservice



OpenSplice DDS

RESTful DDS

- Introduction
- What does RESTful mean?
- Why RESTful DDS?
- An Open Source RESTful DDS Webservice
  - Design
    - Implementation
    - Demonstration
- Questions?

RESTful DDS Webservice - Design

- HTTP is available
  - If you don’t use HTTP, you’re not on the web
- HTTP is sufficient
  - Data-oriented method interface (CRUD)
    - POST, GET, PUT, DELETE
- HTTP fits well
  - RESTful by nature
  - Standard and uniform
  - Well-known
  - Easily extended to secure transport

RESTful DDS Webservice - Design

HTTP uses URLs to address resources

- Mapped to DCPS entities
- Constructed according to DCPS hierarchy
- After creation, entities are addressed by name
  - Chosen by client or generated by server
  - Webservice does not maintain client state
  - DCPS entities are anonymous within DDS
- Only existing datatypes can be referenced

For example addressing a DataReader:

```
http://www.prismtech.com:8182/dds/
  <participantname>/<partition>/<topicname>/
  <typename>/subscribe/<datareadername>
```

RESTful DDS Webservice - Design

HTTP messages can contain bodies (documents)

- Different media types possible
- Selected JSON-strings for ease of use
  - Lightweight alternative to XML
  - Simple way of formatting data structures
  - Open source implementations in many languages
- Used in HTTP requests and responses, if required

For example a response containing DCPS samples:

```
{"samples": [
  {"data": {"trackId":0,"x":234,"y":240,"z":0} },
  {"data": {"trackId":3,"x":234,"y":61,"z":28} }
] }
```

Creation using HTTP POST method

Entity creation if URL points to a factory

- QoS in HTTP request body, default otherwise
- Entity name in HTTP request body, generated otherwise
- Optional parameters are appended to URL
- HTTP response code indicates success or failure
- HTTP response body contains name or error msg

For example creation of a DataReader:

```
POST /dds/<participantname>/<partition>/
    <topicname>/<typename>/subscribe HTTP/1.1
```

Note: <partition> maps 1-to-1 on Publisher/Subscriber

Creation using HTTP POST method

Data injection if URL points to a DataWriter

- HTTP request body is JSON-string containing data
- Type of update depends on URL
  - register_instance, write, dispose, unregister_instance
- Optional parameters are appended to URL
- HTTP response code indicates success or failure
- HTTP response body contains error msg for failure

For example writing a sample:

```
POST /dds/<participantname>/<partition>/
    <topicname>/<typename>/publish/<datawritername>/
    write?time_stamp=sec.nanosec HTTP/1.1
```

Querying using HTTP GET method

QoS querying if URL points to an entity

- No HTTP request body
- HTTP response code indicates success or failure
- HTTP response body contains QoS as JSON-string or error msg

For example querying the QoS of a Topic:

```
GET /dds/topics/<topicname> HTTP/1.1
```

Querying using HTTP GET method

Data querying if URL points to a DataReader

- No HTTP request body
- Kind of access depends on URL
  - read, take
- Optionals parameters are appended to URL
- HTTP response code indicates success or failure
- HTTP response body contains data and info sequence as JSON-string or error msg

For example reading new samples:

```
GET /dds/<participantname>/<partition>/
    <topicname>/<typename>/subscribe/<datareadername>/
    read?sample_state=NOT_READ HTTP/1.1
```

Querying using HTTP GET method

Data querying shortcuts to avoid round-trips

- Querying of a non-existing entity implicitly creates it and its factories, if needed
- Querying data on a factory will return any samples available on any of its contained readers

Implicit creation of DataReader:

```
GET /dds/<participantname>/<partition>/<topicname>/
<typename>/subscribe/<datareadername>/read HTTP/1.1
```

Reading all samples in a Participant:

```
GET /dds/<participantname>/take HTTP/1.1
```

Updating using HTTP PUT method

Changing the QoS for any existing Entity

- Entity addressed via URL
- JSON-string for non-default QoS in HTTP request body
- HTTP response code indicates success or failure
- HTTP response body contains error msg

For example updating a DataReader QoS:

```
PUT /dds/<participantname>/<partition>/<topicname>/
    <typename>/subscribe/<datareadername> HTTP/1.1
```

Deleting using HTTP DELETE method

Deleting any existing Entity

- Entity addressed via URL
- HTTP response code indicates success or failure
- HTTP response body contains error msg

For example deleting a DataReader:

```
DELETE /dds/<participantname>/<partition>/<topicname>/
    <typename>/subscribe/<datareadername> HTTP/1.1
```

RESTful DDS

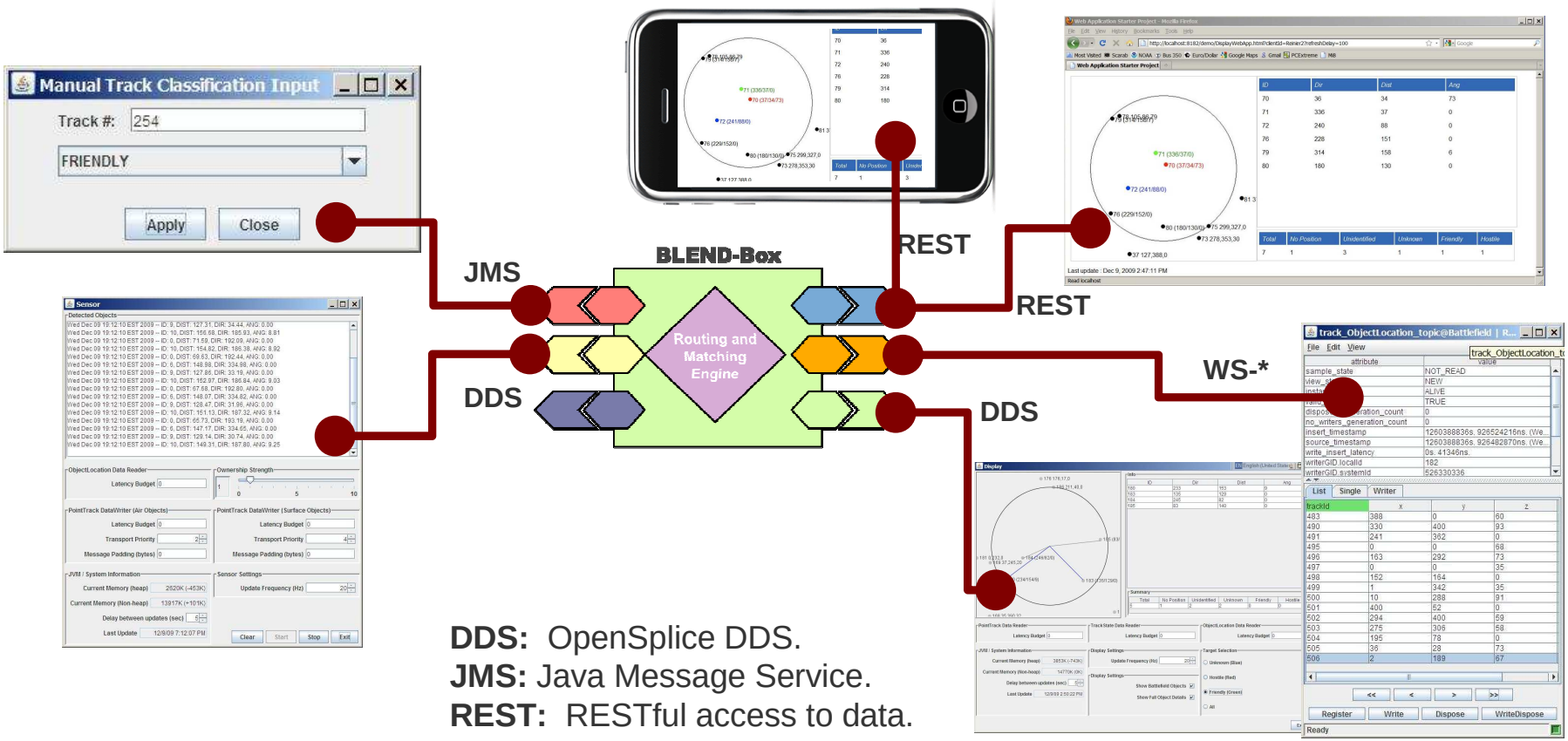
- Introduction
- What does RESTful mean?
- Why RESTful DDS?
- An Open Source RESTful DDS Webservice
  - Design
  - Implementation
  - Demonstration
- Questions?

RESTful DDS Webservice - Implementation

Open Source prototyping effort

- As part of the BLEND-Box project
  - Routing and matching engine
- Written in Java
- Reuses existing open source projects
  - PrismTech's OpenSpliceDDS Community Ed.
  - Noelios Technologies' Restlet v1.1.6
  - Google's Gson
- Allows for static pages
  - for Code-On-Demand (JavaScript) or static information
- Available on Google Projects
  - http://code.google.com/p/restful-dds/

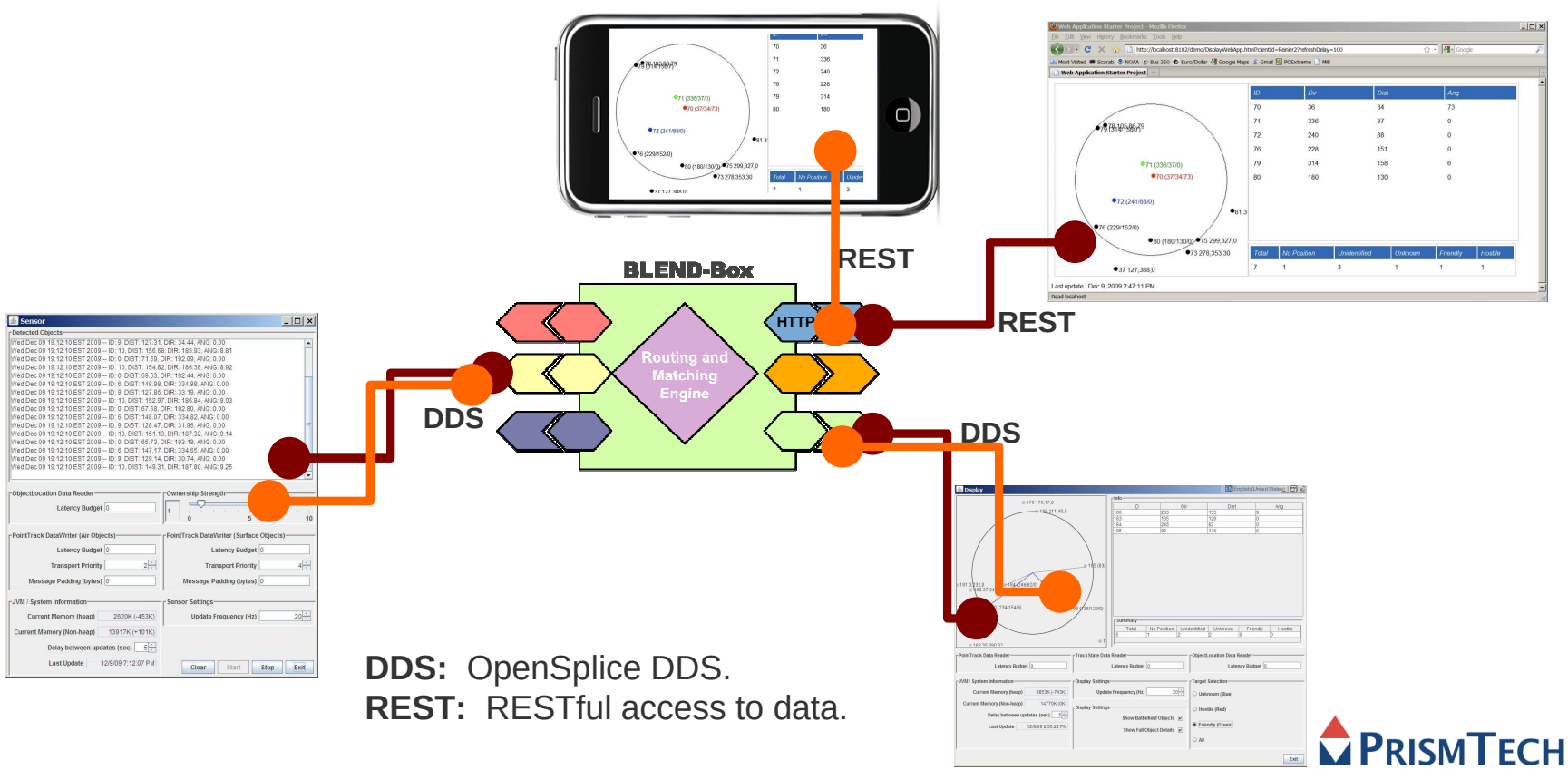
RESTful DDS Webservice - Implementation



DDS: OpenSplice DDS.
JMS: Java Message Service.
REST: RESTful access to data.
WS-*: Web Service.

OpenSplice DDS

RESTful DDS Webservice - Implementation



DDS: OpenSplice DDS.
REST: RESTful access to data.

PRISMTECH

OpenSplice DDS

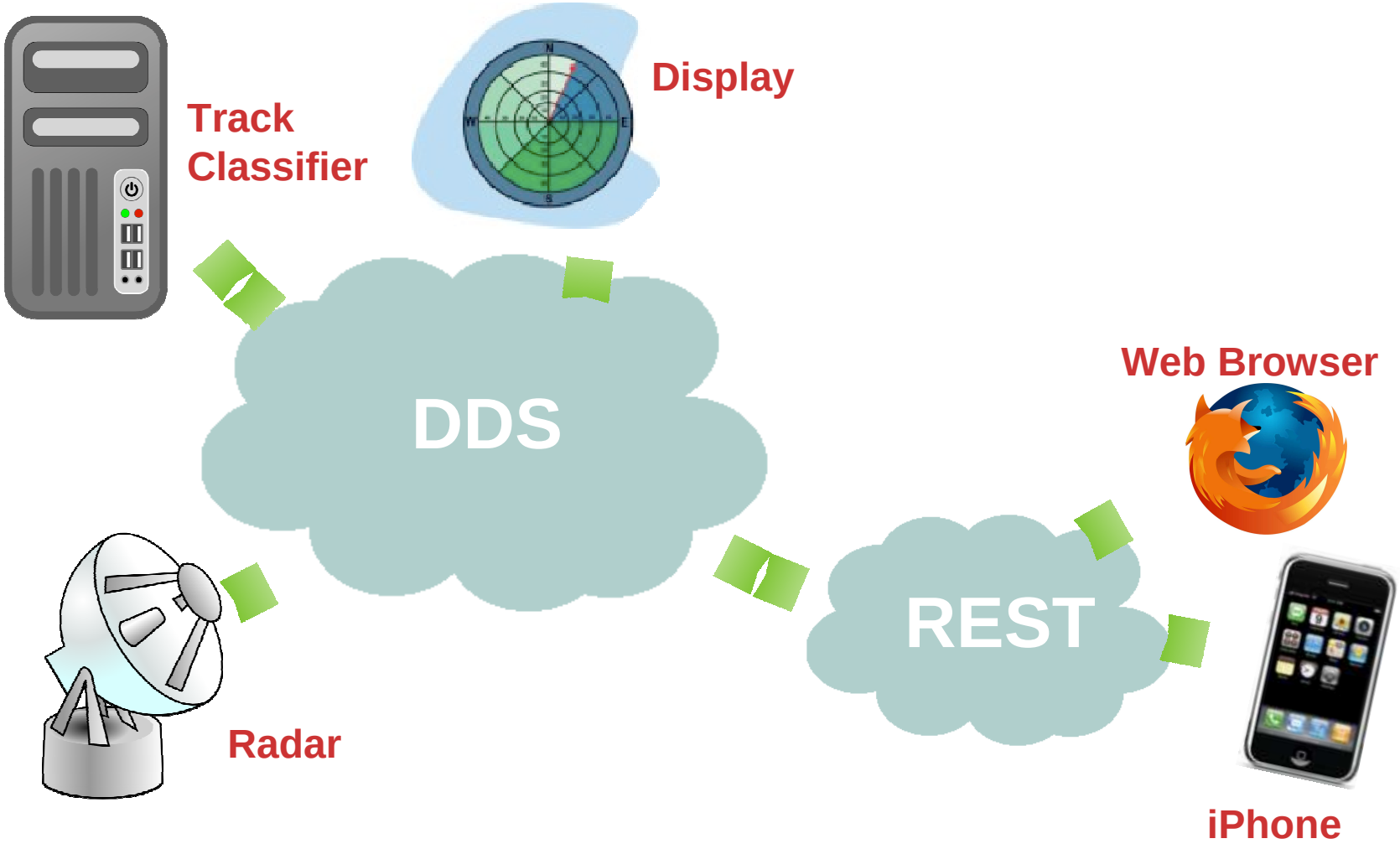
RESTful DDS Webservice - Implementation

- DDS Entities created/destroyed by RESTful DDS Webservice according to HTTP requests
- DDS, not the Webservice, is responsible for maintaining the state of its Entities
- No client-specific state stored in the Webservice
- Webservice acts as a façade to the DDS Entities, which are referred by name
- “RESTful access to data” essentially means “RESTful access to DDS Entities”
- Garbage collection done by Webservice by deleting DDS Entities in case of no activity

RESTful DDS

- Introduction
- What does RESTful mean?
- Why RESTful DDS?
- An Open Source RESTful DDS Webservice
  - Design
  - Implementation
  - Demonstration
- Questions?

RESTful DDS Webservice - Demonstration



OpenSplice DDS

RESTful DDS

- Introduction
- What does RESTful mean?
- Why RESTful DDS?
- An Open Source RESTful DDS Webservice
  - Design
  - Implementation
  - Demonstration
- Questions?

Questions?

Thank you!

reinier.torenbeek@prismtech.com