Fleischer, J., Häner, R., Herrnkind, S., Kloth, A., Kriegel, U., Schwarting, H., Wächter, J. (2010): An integration platform for heterogeneous sensor systems in GITEWS - Tsunami Service Bus. - Natural Hazards and Earth System Sciences (NHESS), 10, 1239-1252

DOI: 10.5194/nhess-10-1239-2010

Nat. Hazards Earth Syst. Sci., 10, 1239–1252, 2010
www.nat-hazards-earth-syst-sci.net/10/1239/2010/
doi:10.5194/nhess-10-1239-2010

Natural Hazards and Earth System Sciences

An integration platform for heterogeneous sensor systems in GITEWS – Tsunami Service Bus

J. Fleischer[1], R. Häner[1], S. Herrnkind[1], A. Kloth[2], U. Kriegel[3], H. Schwarting[2], and J. Wächter[1]

[1]Deutsches GeoForschungsZentrum GFZ, Center for GeoInformation Technology (CeGIT), 14473 Potsdam, Germany
[2]SpaceTech GmbH (STI), Seelbachstraße 13, 88090 Immenstaad, Germany
[3]Fraunhofer Institute for Software and Systems Engineering ISST, Berlin Branch, Steinplatz 2, 10623 Berlin, Germany

Received: 26 February 2010 – Revised: 27 May 2010 – Accepted: 29 May 2010 – Published: 17 June 2010

Abstract. The German Indonesian Tsunami Early Warning System (GITEWS) is built upon a complex sensor data infrastructure. To best fulfill the demand for a long living system, the underlying software and hardware architecture of GITEWS must be prepared for future modifications both of single sensors and entire sensors systems.

The foundation for a flexible integration and for stable interfaces is a result of following the paradigm of a Service Oriented Architecture (SOA). The Tsunami Service Bus (TSB) – our integration platform in GITEWS – realizes this SOA approach by implementing the Sensor Web Enablement (SWE) standards and services.

This paper focuses on architectural and implementation aspects of the TSB. Initially, the general architectural approach in GITEWS by SOA and SWE is presented. Based on this conception, the concrete system architecture of GITEWS is introduced. The sensor integration platform TSB is then discussed in detail, following by its primary responsibilities and components. Special emphasis is laid on architectural transparency, comprehensible design decisions, and references to the applied technology.

1 Objectives

The Tsunami Service Bus (TSB) is the sensor integration platform of the German Indonesian Tsunami Early Warning System (Rudloff et al., 2009). Due to the geological situation in Indonesia, the primary goal of GITEWS is to deliver a reliable tsunami warning message as quickly as possible. This is achieved using several sensor systems: the seismological system, the near real time GPS deformation monitoring system, several tide gauges, and buoy systems. Together they provide the fundamental data necessary to support the prediction of a tsunami wave performed by the warning center. But all these sensors use their own rather fixed proprietary data formats and specific behaviors (Fig. 1).

This is complicated by the fact that GITEWS is a long running system, which has to cope with changing requirements over time: new sensor types might be added while old sensors will be replaced by newer ones. Additionally, new algorithms by experts or including foreign sensor networks are assumed to change interfaces and quantity structures even during the implementation phase. In order to manage this multitude of sensors over time, an additional intermediate layer has to be introduced to provide both the required flexibility for sensor integration as well as a stable and uniform interface for the warning system. Thus, seen from an architectural viewpoint, GITEWS is a typical integration project.

2 Requirements

Although the overall functional requirements for the integration platform TSB could have been stated at project start, the non-functional requirements for system operations were uncertain:

- The TSB shall provide a standardized interface for accessing sensor data as well as for tasking the sensors. The time for accessing sensor data shall be in a range of less than seconds.
- The TSB shall provide midterm storage (weeks) of all incoming sensor data, including post processing and quality checks. Long-term sensor data is not meaningful for an early warning system. The time for processing incoming sensor data shall be in a range of seconds or less.

Correspondence to: J. Fleischer
(jens.fleischer@gfz-potsdam.de)

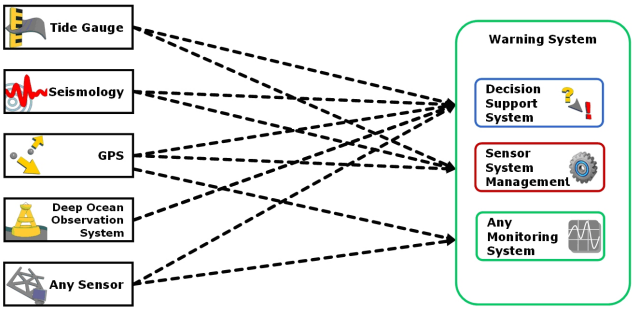

Fig. 1. Multitude of sensors.

- Raw data, like seismic wave data (e.g. MiniSeed), is not stored or processed by the TSB. This work has to be done by the preceding sensors or sensor systems. The TSB provides only already processed sensor data as "information products" necessary for the warning process. E.g. earthquake messages, GPS land displacement vectors, or tide-reduced water heights.
- The TSB shall provide a flexible integration mechanism of new or not foreseen sensor types.
- The implementation of the TSB shall be reliable and robust; operating 24 h a day, 7 days a week.
- A survey of planned sensors and their samplings led to a coarse quantity structure of:
  - The processing and management of ca. 50 sensors and 10 different sensor types.
  - At most a 1 Hz frequency of new arriving sensor messages at the TSB in case of high tsunami sampling rates.
  - At most a peak of 600 sensor samples per second to be processed and stored at by the TSB (accounting the numbers of sensors and their sampling rates).

3 Architectural blueprint

The TSB-approach realizes a functional integration (other than a data level integration), where functionality is provided by dedicated components (or services), communicating via a service infrastructure. These services provide their functionality exclusively via standardized interfaces. Instead of requesting data directly, this approach replaces the tight coupling at data level by a flexible dependency on loosely coupled services. The TSB-approach for functional integration relies both on a Service Oriented Architecture (SOA) as well on an integration concept, built on standardized encodings and protocols provided by Sensor Web Enablement (SWE). A SOA is determined by a layered architecture, consisting

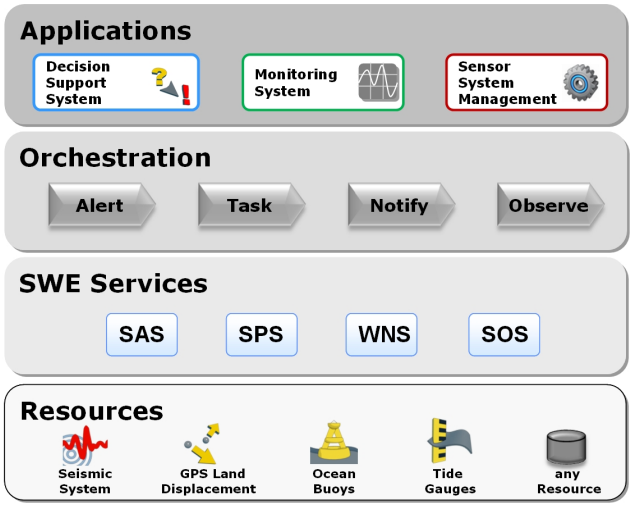

Fig. 2. Layers of a SOA in context of GITEWS.

basically of a resource, service, orchestration, and application layer as shown in Fig. 2 (McGovern et al., 2001; Josuttis, 2007).

Functionality (e.g. alert, task, notify, observe) required by the business processes on the application layer (e.g. Decision Support System) is provided by interoperable services (e.g. SAS, SPS, etc.) that follow common service oriented principles, as service contract, loose coupling, abstraction, reusability, autonomy, statelessness, discoverability, and composability (Erl, 2008). A SOA also orchestrates services to execute specific processes of the application domain, which might be supported by execution languages or workflow engines. The conception of autonomous and loosely-coupled services enables the compilation and reusing of services within the production of manifold multi-hazard systems.

Nevertheless, a SOA does not specify the actual services, their interaction, and accompanying data model. Instead of defining our own service suite in GITEWS we adopted services of the Sensor Web Enablement (SWE) initiative, which aims "to enable all types of [...] sensors [...] to be accessible and, where applicable, controllable via the Web" (Botts et al., 2007).

The general applicability of SWE for sensor based natural hazard systems was shown in early case studies and implementations (Walkowski, 2005; Moodley et al., 2006; Chu et al., 2006). In addition, the SWE framework is functional complete for the TSB: only a set of four SWE services are sufficient to cover the required functionality as demonstrated in Simonis (2008), or in several projects (as by Kunz et al., 2009 or Kobialka et al., 2010). Furthermore, at the time when GITEWS started in 2006, no other open XML standard existed, which specified sensors and their supporting service infrastructure by means of practicable interfaces as well as

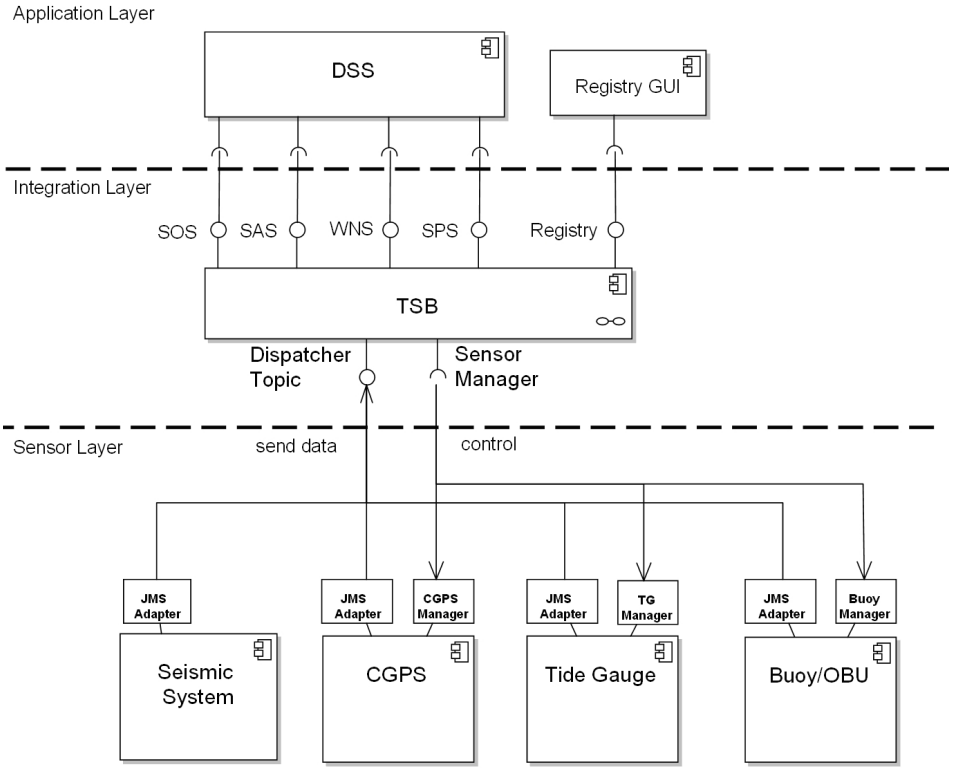

Fig. 3. GITEWS architecture.

a data/metadata model (Klopfer, 2005). And the activities by the Open Geospatial Consortium (OGC[1]) promoting the evolving standard were also very promising.

Unfortunately an open and sufficient SWE implementation did not exist back then. For the Sensor Observation Service (SOS) were only early implementations available (e.g. 52° North[2]). But, not designed for production environments, low performance in data ingestion, and shortcomings of the repository led to the decision to implement the services by ourselves.

The following SWE specifications (OpenGIS) have been implemented:

- Observations & Measurements (O&M): model for observations and measurements.
- Sensor Model Language (SensorML): model for describing sensor systems.
- Sensor Observation Service (SOS): service for obtaining sensor observations.
- Sensor Planning Service (SPS): service for tasking sensors.
- Web Notification Service (WNS): service for asynchronous dialogues.
- Sensor Alert Service (SAS): service for sending alerts.

4 System architecture of GITEWS

The system architecture of GITEWS is depicted in Fig. 3. Following a SOA the system architecture is separated into a *sensor*, *service*, and *application* layer. A separate orchestration layer was omitted (as proposed for common SOAs) since the four SWE services are small enough to be managed by the client applications (DSS, Registry GUI) directly.

4.1 Sensor layer

Sensors and sensor systems are located at the Sensor Layer, providing data for the warning system. This data contains typical sensor measurements from tide gauges, buoys or ocean bottom units (OBU), such as sea water heights, meteorological data, or system health parameters. Moreover, the data for the warning system is also comprised of more computationally intensive events, such as detected earthquakes from the Seismic System or land displacements from the Continuous GPS System (CGPS).

[1]The Open Geospatial Consortium, Inc, http://www.opengeospatial.org/ogc, last access: May 2010.

[2]52North Initiative for Geospatial Open Source Software GmbH, http://52north.org/, last access: May 2010.

Tide gauges and buoys, together with the OBU, are understood as sensors. They are distinguishable measuring units deployed in the field, whereas the Seismic System and the CGPS are understood as sensor systems: extensive computing systems that process data of entire sensor networks. The internal details of sensors and sensor systems as well as their dependencies among themselves are irrelevant for the integration architecture and is not shown (e.g. processing GPS data of the buoy by the CGPS system). In the following the distinction between *sensor* and *sensor system* is omitted for readability.

In order to enable the integration of all sensors the TSB requires two interfaces: the *Dispatcher* and a *Sensor Manager* interface. The Dispatcher interface is the receptacle for receiving all incoming sensor data, including sensor measurements as well as sensor alerts (e.g. sea surface heights, water anomalies or detected earthquakes). On the other hand, the Sensor Manager interface is used for managing and tasking sensors by the TSB. For example, a sensor may be set to a higher acquisition rate mode for near real time processing (called: "Tsunami Mode") by this interface.

Regarding the technical integration both interfaces should be designed as simply as possible, at least on protocol level. They should be independent of the different proprietary data formats and behaviors of the sensors. Consequently, the Dispatcher interface is designed as a simple message receiver, accepting arbitrary Java Message Service (JMS) messages (currently only text and byte messages) and the Sensor Manager interface is designed as an arbitrary synchronous TCP/IP command interface. Additional sensor adapters (JMS-Adapter, Buoy Manager, etc.) are created to enable the sensors to communicate via these interfaces. As a result the integration effort itself is "minimally invasive" on the sensor side. The more difficult task of semantic integration takes place inside the TSB via the concept of "plug-ins".

4.2 Integration layer

The integration layer contains the TSB serving as the integration platform. The TSB provides interfaces up to the application layer comprising the four SWE services (SOS, SAS, SPS, and WNS) and interfaces down to the sensor layer comprising the generic Dispatcher Topic and the Sensor Manager interfaces.

The TSB was realized as a single deployable component including all services of the integration layer (excluding the database system). Separated stand-alone SWE services were not necessary in GITEWS, but nevertheless are supported and can be deployed by means of the TSB implementation.

Furthermore, the TSB also implements the Sensor Registry, responsible for all sensor metadata management in GITEWS. The registry, for example, contains metadata (data about data) for discovering new or modified sensors (via SWE services), along with their complete interface description (sensor name, station codes, description of data streams, etc.).

4.3 Application layer

Finally, the client applications reside at the topmost layer. The most important of these applications is the Decision Support System (DSS), which aggregates the sensor data further to highly aggregated information products to assist the Chief Officer on Duty in his/her decision whether a tsunami warning should be disseminated. Additionally, also the Registry-GUI has access to sensor metadata through the TSB.

5 Sensor integration platform TSB

While the previous chapters have explained the functional and non-functional requirements of the TSB, the next section presents the architectural and technical details of the integration platform. The internal architecture of the TSB – seen from a logical viewpoint (Kruchten, 1995) – reflects the main use cases of the sensor integration platform (Fig. 4).

Logically the TSB is divided into five components:

- The *Processing* component receives incoming data as messages. The data is analyzed, processed, and stored into the database. Registered applications are then informed about the new available data or about system alerts (by WNS resp. SAS).

- The *Provisioning* component provides access to all sensor data for client applications in terms of the SOS interface.

- The *Tasking* component enables the uniform control of sensors. The Tasking component forwards requests to the sensor specific command adapter (via Sensor Manager Interface) and forwards the (asynchronous) results back to the client application.

- The *Registry* is the central provider for all sensor metadata in GITEWS. It also provides functions for the management of sensor metadata (create, modify, delete). The Registry stores its metadata into the Database.

- The *Database* acts as a general storage for all sensor data and metadata.

5.1 Design and implementation decisions

The final architecture of the TSB is determined by several design and implementation decisions which are presented below.

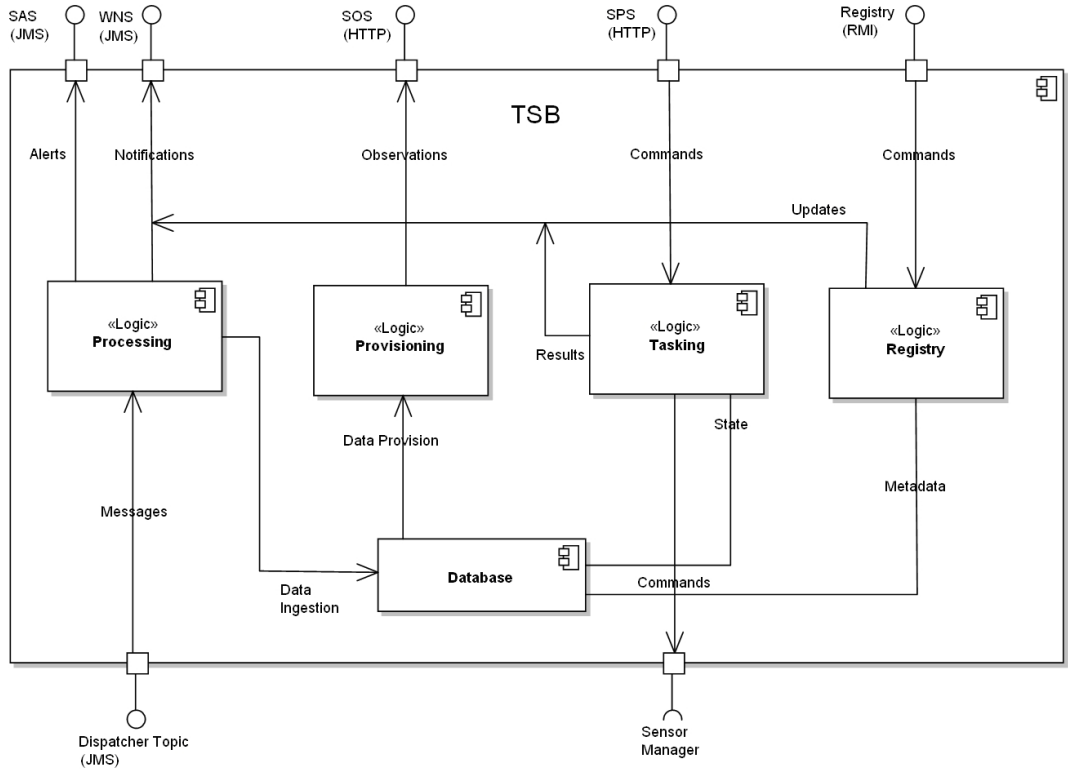

Fig. 4. Logical architecture of the TSB.

5.1.1 Messaging for data and alert transportation

In GITEWS, almost all data transportation between distributed components happens on basis of asynchronously message oriented middleware (MOM), especially for the asynchronous services SAS and WNS. For example, sensor data is first wrapped by messages and then sent to the TSB.

The JMS technology has been chosen as MOM because of its elaborate and practicable interface as well as its robust and high performance implementation by JBoss Messaging. Another option for an asynchronously messaging protocol would have been XMPP. But as stated in Shigeoka (2002), the XMPP standard does not define – in comparison to JMS – any Quality of Service (QoS) features for message delivery. This is left to the XMPP implementations, therefore not standardized, and might vary between implementations. Because of keeping the TSB free of any proprietary extensions, JMS/JBoss was chosen for its defined (by the standard) and guaranteed (by the implementation) message delivery. Consequently we intentionally departed from SWE's recommendation for using XMPP as communication protocol for the SAS. (Nevertheless, for WNS the protocol can be chosen arbitrarily). In particular the following JMS features have been used: Transactionality, Durable Subscription, Server Side Filtering, Message Priority, and Preserving of Message Sequence.

5.1.2 Data access and sensor control by web services

Sensor data is stored persistently in a database accessible via the SOS. For tasking and managing sensors the SPS is used. Both services are implemented as standard web services (servlets), following the synchronous request/response pattern over HTTP.

5.1.3 Reliable data transport

Reliable data transportation is a crucial requirement for the TSB. It is realized by a reliable transport chain from sensors up to the application layer by means of JMS. To be more precisely: The reliable transport chain begins with the JMS adapter, not with the sensor itself. This technology guarantees that messages sent to the Dispatcher interface are sent *once, and only once* to clients of the application layer (e.g. to avoid time consuming detection of data duplicates). This feature is additionally supported by durable subscription, which ensures that even if the client is currently unreachable (in case of downtime, etc.), messages will be delivered later when the client reconnects.

5.1.4 Transactional processing

The processing of incoming messages is composed of several activities inside the TSB, all executed in one single transaction. This guarantees the correct and coherent processing

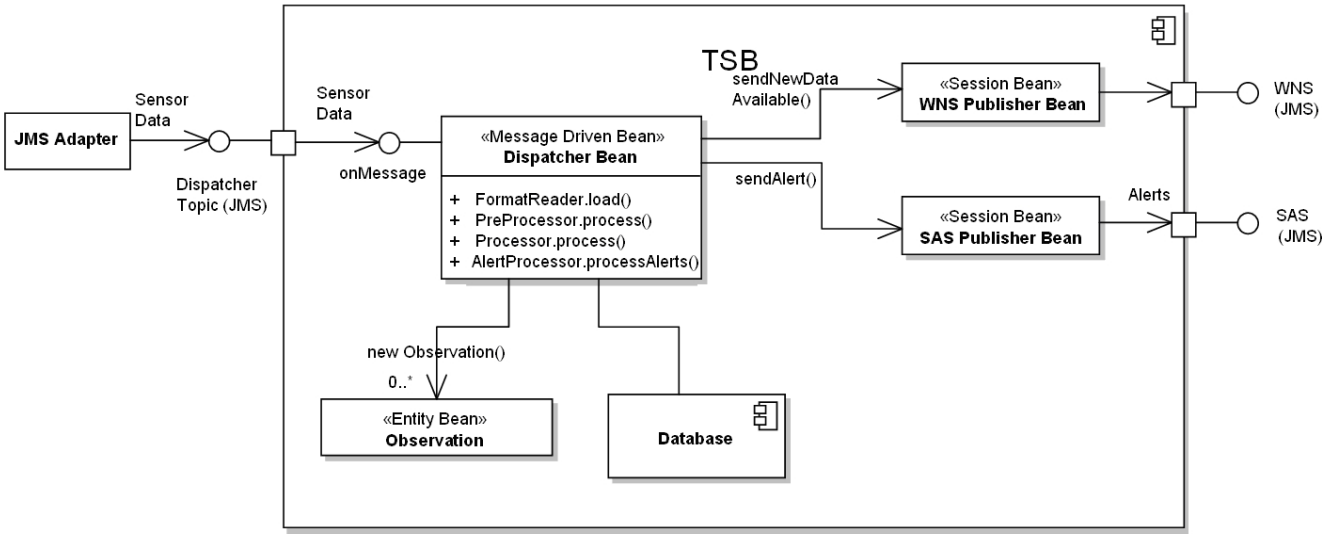

Fig. 5. Processing of sensor data.

workflow for every incoming message. In case of an error, the entire workflow is rolled back, retried again for several times, and if the error cannot be cleared the faulty message ends up finally in a dead letter queue.

The implementation of the TSB does not use distributed transactions. In particular the JMS-Adapter is not part of a transactional process, because of performance criteria during the long commit phase. Nevertheless, the JMS-Adapter is robust: i) incoming messages are buffered persistently, ii) automatic communication retries in case of network error.

5.1.5 Sensor integration by means of plug-ins

To integrate a new sensor type, its specific semantic and data format has first to be incorporated into the TSB. This is done in a flexible and straightforward way by a plug-in mechanism, which extends the default processing sequence depending on the actual sensor type (Sect. 4.3).

5.1.6 TSB as a Java enterprise application

The operational requirement for a robust, scalable, and maintainable implementation using standard and proven technologies has been met through the use of Java EE together with JBoss as implementation and application framework. According to this technology, the TSB is completely implemented based on Enterprise Java Beans (EJBeans) and is deployed and executed as a single enterprise application component (`*.ear`) onto the JBoss application server.

The next sections provide a detailed description of the TSB along the logical components mentioned above. The notion "bean" is frequently used later on and denotes EJBeans only, i.e. Java classes managed by an application server.

5.2 Processing of sensor data

The processing of all sensor data is shown in Fig. 5. The central class is the Dispatcher Bean, which consumes, analyzes, processes, stores, and publishes (via Publisher Beans) all incoming sensor data. The methods of the Dispatcher Bean are generic and defined by plug-ins (see Sect. 4.3).

The content, format, frequency, and size of the incoming sensor data (data stream) is in general not restricted, as long as matching plug-in exists. Typical input streams are frequently delivered binary sensor readings as well as infrequent and complex earthquake bulletins in XML (SeisComP-ML on basis of QuakeML). Raw data or multi-media files were not intended for the TSB.

The Dispatcher Beans expects all data to be delivered as JMS messages. For this purpose specific JMS-Adapters exist. They enable the communication between the proprietary sensors and the TSB on protocol level. They also add necessary information about the origin of the sending sensor (e.g. type of sensor data or unique sensor identifier).

The main task of the Dispatcher Bean is to perform the specific processing according to the delivered message type ("dispatching"). Thus, for each sensor data type a specific instance of the Dispatcher Bean type exists. The technical assignment of an incoming message to its processing Dispatcher Bean is then automatically performed by the JMS message broker: To assign the incoming message to its appropriate Dispatcher Bean, the JMS feature of "server side filtering" is used (`messageSelector`). However, the JMS usual selector mechanism (by `Topics` resp. `Queues`) is used on a coarser level to distinguish the different subscriber/publisher interfaces (`Dispatcher`, `WNS`, `SAS`, `Log`).

The processing results in a set of observations representing the actual sensor data. These observations are implemented as entity beans (≪`Entity Bean`≫) capable of being stored

automatically into the database (by Java EE's container managed persistence).

At the end of the processing, either for general notification or as a result of sensor alerts, messages are sent up to the application layer by the WNS PublisherBean (e.g. `NewDataAvailable`) or by the SAS PublisherBean (e.g. `SeismicSystemAlert`), respectively.

In addition to the reliability of the processing above, one also obtains important operational qualities by using the Java EE platform, namely:

- transactional processing of sensor data,
- concurrent processing of all data streams,
- concurrent processing of all messages in a single data stream,
- pooling of Dispatcher Beans for efficiency,
- pooling of database connections.

Regarding the concurrency it is worth mentioning that concurrent processing is not useful for certain data streams where the sequence of incoming messages must be preserved. In these cases we use JMS Queues instead.

An example for a Seismic Alert sent by the SAS Publisher Bean is shown in Fig. 6. An earthquake was detected at `samplingTime`, together with the following earthquake source parameters: `numberOfStations`, `originTime`, `longitude`, `latitude`, `depth`, `magnitude`, and `rmsErr`. The XML example is simplified for readability.

5.3 Plug-in concept

The design of the TSB has to be flexible in order to meet the requirement to integrate new sensor types with their new and unforeseen data formats and behaviors. This flexibility is realized by so called *plug-ins*, defining a sensor specific data processing along the Dispatcher's standard execution sequence. The standard execution sequence is made up of five activities:

- Initially the `FormatReader` parses and maps all sensor data into an internal data representation. It also acts as filter reducing the stream to relevant data only.
- Then the `PreProcessor` performs all required processing activities on the sensor data *before* storing it to the database. The intermediate step is necessary because intended database constraints, e.g. unambiguity of ids, would interfere at this stage with the given datasets.
- Then the sensor data is persisted into the database context.
- The `Processor` performs the actual processing exclusively on database data. This processing at a central place is necessary since data streams come from different sources and need to be merged. For example, buoy water height streams are corrected by air pressure data streams.
- Finally the `AlertProcessor` processes the data to identify anomalies (annotated by the sensors themselves), which are then disseminated as alerts.

```
<Alert>
  <Observation>
    <samplingTime>
      <timePosition>2009-08-20T08:53:40.844…
    </samplingTime>
    <procedure …="urn:…:seismology:system:ems"/>
    <observedProperty …="…:phenomenon:earthquake"/>
    <featureOfInterest>…
    <result>
      <SimpleDataRecord>
        <description>Seismic System Records</…>
        <field name="numberOfStations">
          <value>34<value>…
        <field name="originTime">
          <value>2009-08-20T08:50:00.593<value>…
        <field name="longitude">
          <value>101.11392</value>…
        <field name="latitude">…
        <field name="depth">…
        <field name="magnitude">…
        <field name="rmsErr">…
        …
```

Fig. 6. Example of seismic alert.

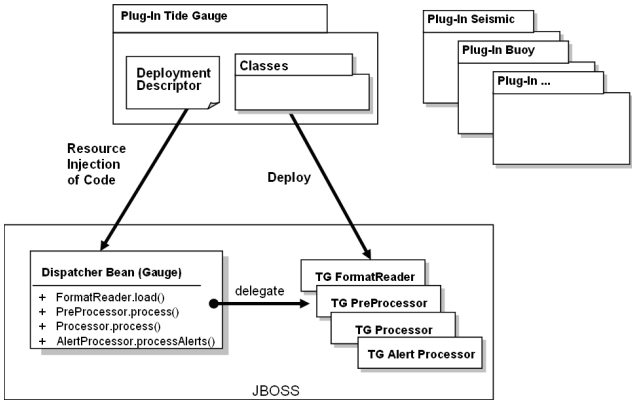

Fig. 7. Plug-in for tide gauge.

The actual processing is hardwired inside the plug-ins and depends on the stream characteristic. Typically for the TSB is to process each arriving data package in one piece. For delegating their processing workload, plug-ins can call external services (e.g. Web Processing Service, WPS of OpenGIS), but restricted by the Java EE architecture plug-ins cannot act as server.

The execution sequence above is implemented using *resource injection* (Java EE 5 feature) into the Dispatcher Bean. Figure 7 shows the concept, based on a tide gauge plug-in.

Table 1. Deployment descriptor of plug-in.

MessageSelector	FormatReader	PreProcessing	Processing	AlertProcessing
`SeismicEventStream`	`SeismicReader`	`NullProcessingBean`	`NullProcessingBean`	`SeismicAlertBean`
`BuoyWaterHeightStream`	`BuoyReader`	`BuoyPreProcessingBean`	`BuoyProcessingBean`	`BuoyAlertBean`
`BuoyMeteorologicalDataStream`	`BuoyMeteoReader`	`NullProcessingBean`	`NullProcessingBean`	`NullAlertBean`
…	…	…	…	…

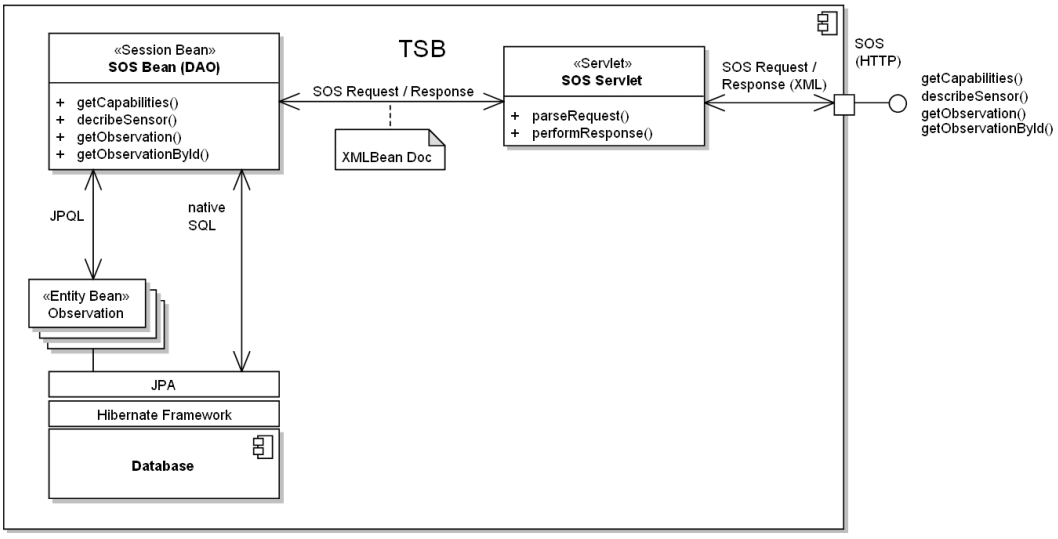

Fig. 8. Provisioning of sensor data.

The standard execution sequence is realized by consecutively calling four interfaces (`FormatReader.load()`, ...). To enable dynamic adaption of this execution sequence the actual executing classes (e.g. `TGFormatReader`,...) cannot be "statically linked". Instead, the execution classes are firstly described as named resources inside the Deployment Descriptor and secondly attached (by name) to the dispatcher code by means of Java annotations. These resources can then be provided at runtime to the Dispatcher Bean (injected) by the Java interpreter. Thus, the behavior of the dispatcher is changeable at runtime by simply changing the Deployment Descriptor. On basis of resource injection a plug-in is nothing but a file package, combining the Deployment Descriptor with its related classes needed for processing.

A typical processing configuration inside the Deployment Descriptor is shown in Table 1.

For the seismic system (`SeismicEventStream`) with its single data stream, a format reader and a bean for alerting are configured. Since data processing was already performed by the seismic system, no additional pre- and processing is needed (`NullProcessingBean`). In contrast to this, the buoy system produces several data streams, each treated differently: for example, the water height stream needs additional preprocessing whereas the meteorological data stream does not need any processing. Currently plug-ins for twelve different data streams have been developed.

5.4 Provision of sensor data

The implementation of the SOS for provision of sensor data (measurements, alerts, and metadata) is straightforward with respect to a typical three tier client/server architecture for web applications (McGovern et al., 2001): a SOS Servlet for the presentation tier, a SOS Bean for the business logic tier, and the entity beans for the data tier (see Fig. 8).

The Sensor Observation Service (SOS) is realized as a simple web service built on top of the Java EE platform. An incoming SOS request is received and parsed by HTTP connection managing SOS Servlet and then converted from XML into an internal Java structure (by XMLBeans) representing the SOS request. Finally, the SOS request (`XMLBeanDoc`) is transferred to the SOS Bean, which executes the submitted SOS request according to the SWE specification: `getCapabilities`, `getObservation`, etc.

```
<GetObservation>
  <offering>…:tideGaugeObservations</…>
  <eventTime>
    …
    <beginPosition>2009-11-26T00:00:00.000</…>
    <endPosition>2009-11-28T00:00:00.000</…>
  </eventTime>
  <procedure>…:procedure:tg:station:sade</…>
  <observedProperty>…tideGaugeWaterHeight</…>
…
```

Fig. 9. Example of SOS request.

```
<ObservationCollection>…
  <samplingTime>…
    <beginPosition>2009-11-26T00:00:00.000</…>
    <endPosition>2009-11-27T07:45:00.000</…>
    …
    <result>
      …
      <SimpleDataRecord>
        <field name="samplingTime">…
        <field name="systemAvailability">…
        <field name="sensor1_rssh">…
        <field name="subsystemVoltage">…
        <field name="sensor1_issh">…
        …
      </SimpleDataRecord>
      …
    <encoding>
      <TextBlock  tokenSeparator=","
                  blockSeparator="@@"…/>
    </encoding>
    <values>2009-11-26T00:00:00.000,,
-0.015,12.1,1.193,true,11.146,true,false,
-0.029,true,-0.029,11.146@@
2009-11-26T00:00:20.000,,0.015,12.1,1.222,
true,11.144,true,false,-0.03,true,-0.03,
11.144@@2009-11-26T00:00:40.000,,0.019,
11.9,1.226,true,11.146,true,false,-0.027,true,
-0.027,11.146@@2009-11-26T00:01:00.000,,
-0.014,12.1,1.193, …
```

Fig. 10. Example of SOS response.

In addition to providing data, the SOS Bean also serves as a Data Access Object (DAO), responsible for the persistence of data. The SOS Bean decouples Java's business logic from the persistence technologies underneath. To accomplish this decoupling, the TSB uses the Java Persistence API (JPA), where queries for entity beans can be written directly in JPQL. For JPA the Hibernate implementation is used. Because the overhead for creating beans is not practical for large results sets, for critical cases the JPQL/entity bean mechanism is bypassed by using native SQL.

Figures 9 and 10 show an example for acquiring tide gauge measurements by the SOS service. In Fig. 9, measurements of sensor type `tideGaugeObservations` are requested. In particular, water heights of the tide gauge station `sade`, during the time of `beginPosition` and `endPosition` are requested.

The response is shown in Fig. 10, which contains the time interval `samplingTime`, the structure of the delivered data record `SimpleDataRecord`, and the actual measurements of the tide gauge `values`. Moreover, the token separators are provided by `tokenSeparator`, etc.

5.5 Tasking of sensors

The implementation of the SPS for tasking sensors is shown in Fig. 11. The architecture is similar to the SOS implementation, but is extended by a controller component (`Controller`) for tasking external sensors.

As in the previous subsection, SPS tasks are submitted via a SPS Servlet and then executed by the SPS Bean. The SPS Bean should represent a generic sensor type, unaware of the actual implementation and specific behavior of the sensor to be tasked. Therefore, the SPS Bean provides, according to SWE, common functions for all sensors: providing metadata information (`getCapabilities`, `describeTasking`), commanding the sensor (`submit`), and getting the status of submitted commands (`getStatus`).

To access a sensor by its native format and protocol, the request is further forwarded to sensor-specific controller classes (`CGPS-`, `TideGauge-`, and `Buoy Controller`), responsible for all the proprietary communication and controlling (hostname lookup, protocol mapping, access list control, etc.). The specific controller class is then determined by a registry look up, thus maintaining the relationship between sensor and controller. For example, the CGPS system is commanded by a custom protocol based on TCP/IP level, whereas the tide gauge and buoy managers are realized by web services, capable of exchanging SOAP messages, specified by a WSDL description.

As an example of a SPS request Fig. 12 shows the command to set the buoy `su01` into a higher sampling mode (`tsunamiMode=true`). However, the buoy can not perform and confirm the request immediately, since the fact that satellite links must established and subsystems must be initialized first. Therefore the final confirmation has to be postponed until the buoy is fully functional and can reply asynchronously via the WNS channel `notificationTarget`.

Future implementations of the TSB should also apply the generic plug-in concept (Sect. 4.3) for sensor tasking, superseding the registry/controller look up mechanism presented here.

5.6 Management of sensor metadata

The registry is the central authority for sensor metadata in GITEWS. It provides metadata of all sensors, including static information necessary for discovering and using a sensor (see Fig. 13).

The registry aims at two things: Firstly, the registry is the central metadata provider for all GITEWS applications. The client applications have access to the metadata via SOS and

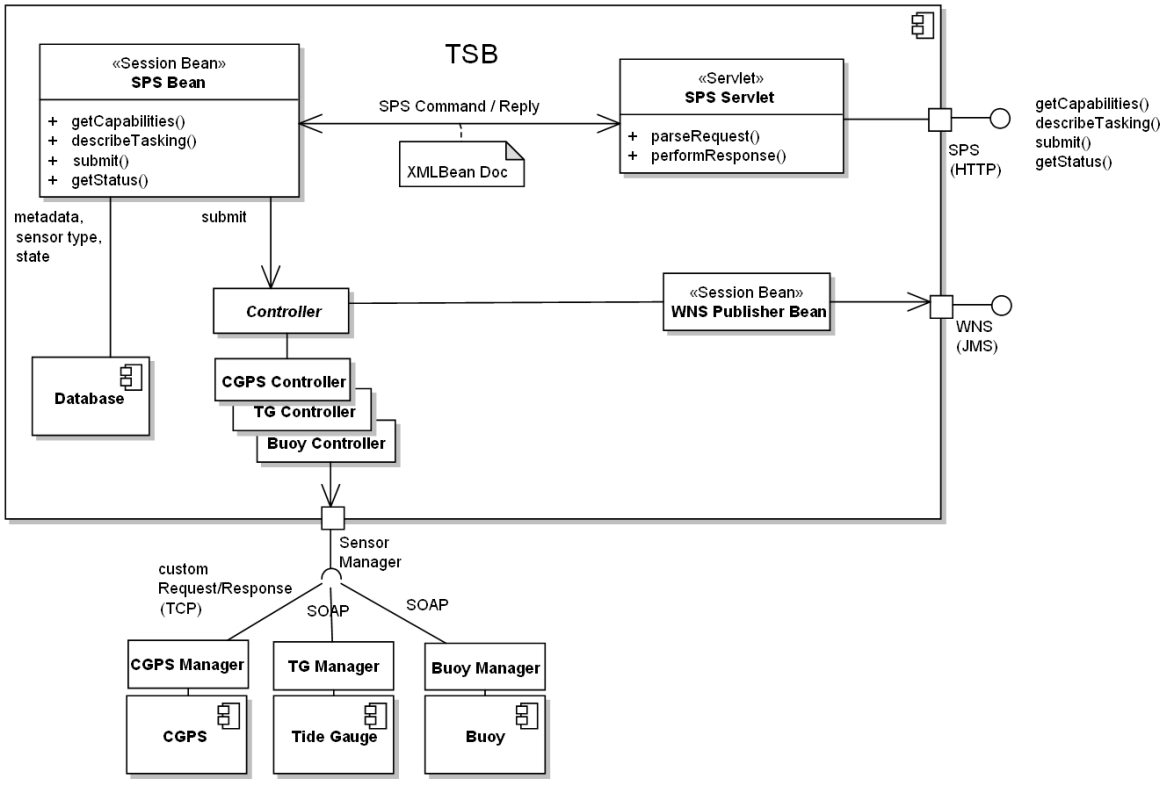

Fig. 11. Tasking of sensors.

```
<Submit>
  <notificationTarget>
    <notificationID>WNS</notificationID>
    <notificationURL>jnp://tsb-prod:1099</…>
  </notificationTarget>

    <sensorParam>
      <sensorID>urn:…:buoy:station:su01</…>
      <parameters>
        <InputParameter … ="tsunamiMode">
          <value>
            <Boolean>
              <value>true</value>
            </Boolean>
          </value>
        </InputParameter>
      </parameters>
    </sensorParam>
</sps:Submit>
```

Fig. 12. SPS request for setting the tsunami mode.

SPS according to the SWE specification. Secondly, the registry enables the direct manipulation of metadata, which includes the creation and removal of sensors ("bringing into existence"). A dedicated user interface (Registry GUI) exists for managing the metadata.

A typical use case for the registry thus might consist of the following individual steps:

1. Get the list of all available offerings.
   An "offering" is the topmost data category and represents a sensor type in GITEWS.
2. Get the list of available sensors providing that offering.
3. Get the list of available data streams for that offering.
   An "offering" can contain several data streams (phenomenon).
4. Get the description of the sensor itself in SensorML.

The metadata for steps 1–3 are delivered by the SOS in a single `getCapabilities()` response. The metadata for step 4 is delivered by the SOS in a `describeSensor()` response.

Figure 14 shows an excerpt of offerings in GITEWS provided by `getCapabilities`.

As an example Fig. 15 shows all available tide gauges, i.e. the list of sensors providing the offering `tideGaugeObservations`.

Every sensor (= "procedure" in the notion of SWE) is identified by its unique Unified Resource Name (URN). The URN is composed of: organization, project, marker "def" for a subsequent definition, marker "procedure" for sensors, classification "tg" for tide gauges, marker for "sensor" (= `station`) or "sensor system" (= `system`), and finally the sensor name ("sade"). Currently over 200 sensors and sensors systems are known by the registry.

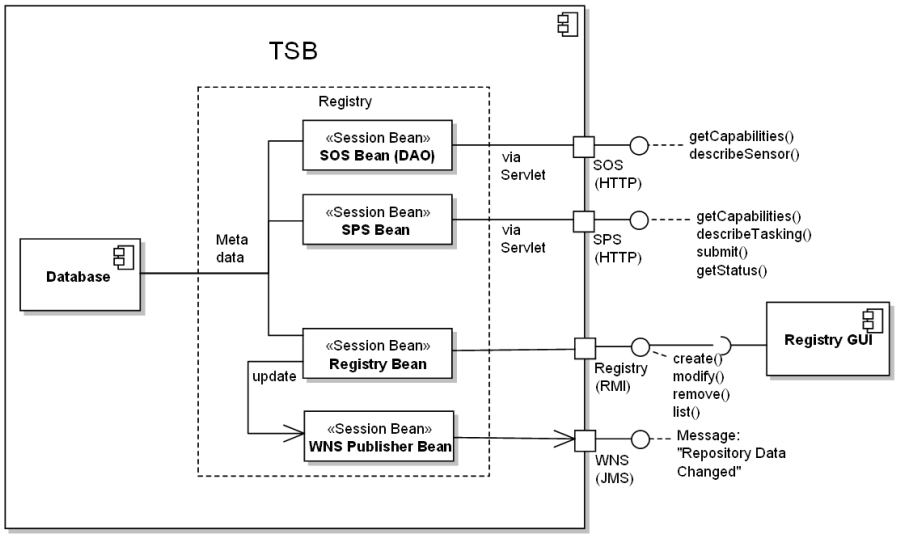

Fig. 13. Managing of sensor metadata.

```
<Capabilities>
  <Contents>
    <ObservationOfferingList>
      <…"seismicHeadsUps"> …
      <…"seismicAlerts"> …
      <…"doosObservations"> …
      <…"obuObservations"> …
      <…"pactObservations"> …
      <…"cgpsObservations"> …
      <…"buoyObservations"> …
      <…"tideGaugeObservations"> …
…
```

Fig. 14. Listing of all available offerings in GITEWS.

```
<Capabilities>
…
<procedure … ="urn:gfz:gitews:def:procedure
                :tg:station:sade"/>

<procedure … ="…:tg:station:tdal"/>
<procedure … ="…:tg:station:tjls"/>
<procedure … ="…:tg:station:sebl"/>
…
```

Fig. 15. Listing of available sensors.

The getCapabilities request delivers also the available data streams (composite phenomenon) for the tideGaugeObservations offering. Figure 16 shows the six streams for tide gauges: `tideGaugeMeteorologicalState`, …, `tideGaugeGPS`, and `tideGaugeWaterHeight`. For the latter the internal structure is also shown.

Finally, the metadata for sensors is shown in Fig. 17 (as retrieved by `describeSensor()`). Metadata for sensors is reduced to what is technically relevant. An elaborated

```
<Capabilities>
  …
  <CompositePhenomenon …
      "tideGaugeMeteorologicalState" …> …
  <…  "tideGaugeMeteorologicalData" …> …
  <…  "tideGaugeWaterHeightState"> …> …
  <…  "tideGaugeState" …> …
  <…  "tideGaugeGPS" …> …

  <…  "tideGaugeWaterHeight" …>
        <component …"…:systemAvailability"/>
        <component …"…:sensorAvailability"/>
        <component …"…:reducedSeaSurfaceHeight"/>
        <component …"…:instantSeaSurfaceHeight"/>
          …
```

Fig. 16. Listing of available data streams (Phenomena).

```
<SensorML>
  <member …>
    <System>
      <characteristics>
        <SimpleDataRecord>
    <field name="shortName">…
    <field name="longName">…
    <field name="stationCode">…
    <field name="stationType">…
    <field name="systemType">…
    <field name="geographicDescription">…
    <field name="responsibility">…
    <field name="reliability">…
    <field name="gpsForSensorAvailable">…
    <field name="taskable">…
    <field name="integratedIntoSimulation">…
    <field name="integratedIntoTSB">…
  …
```

Fig. 17. Sensor metadata in GITEWS.

technical (self-) description for discovering sensors (e.g. using UDDI) or a human readable approach for metadata was not intended in GITEWS.

6 Results

6.1 Applicability of SWE

An important motivation for choosing SWE was to get a mature sensor and data model as a solid conceptional fundament for implementing the TSB. SWE has satisfied this requirement in GITEWS well. Indeed, the SWE features for characterizing sensor data (by offering, procedure, phenomenon, event time, and feature of interest) were general enough for using them also as super keys for our relational database model (flexible key-value tables). Consequently, new sensor data types fit into the TSB as long as they fit into SWE data model. Moreover, SWE's applicability was also proven by the DSS, where SWE is used as interface standard for accessing huge geodata, for sensor data management, and for map display and communication with the GITEWS simulation system (Raape et al., 2010; Behrens et al., 2010).

Although the data model of SWE is perfect for time series, it is not well suited for complex data structures. Confined to SWE's fixed metamodel for observations it is not possible to deviate from it. For example, it is not possible to build nested observations needed for hierarchical data structures, or to add a unique identifier for retrieving named observations, or to introduce a second independent time dimension for time series.

The complexity of the SWE standards itself can be tedious. For example the TSB uses only a subset of SWE, but which is made up of twelve different XML Schemata with a total of over 1000 defining XML Elements. As a consequence a great deal of effort was required for tailoring SWE down to find a proper and compliant SWE subset, which serves as a practical dialect for GITEWS. Our SWE dialect ("best practices") comprises first of all structural patterns for describing data, as for time series, data records, data types, and a URN nomenclature for names, units, and identities. Simple behavioral patterns are provided as well. By means of this dialect the entire interface between sensors and clients was specified and is serving as a binding contract between TSB and DSS.

Another result is that GITEWS' sensors do not "speak" SWE natively, simple as a matter of implementation and bandwidth costs. But also because the focus of sensor experts is different than SWE, who favor their own (binary) standards, which satisfy their domain requirements in terms of functionality and transmission efficiency best. The interdisciplinary integration work is therefore left to a sensor specific processing by adaptors and plug-ins of the TSB.

The behavioral services of SWE (SPS, SAS, and WNS) are simple and generic, but sufficient enough for GITEWS. Thus, the ability of the TSB for sensor integration is only limited by SWE's data model, taken into account SWE's unsuitability for binary sensor protocols (for the simple reason that SWE uses XML and HTTP).

Our implemented SWE services (SOS, SPS, SAS, and WNS) follow the specifications. But as they were not formal tested by the OGC[1] they cannot be marked as "compliant". Nevertheless, our SWE services deliver valid XML documents according to the schemata.

6.2 TSB as enterprise application

The TSB was finally implemented as a single Java EE/JBoss application using important QoS features of this framework, such as reliable messaging, transactional processing, container based persistence, scalability, connection pooling, and monitoring.

But Java EE was not set from project start; it was rather a result of several prototypes. Because only Java and JMS were obligatory, early TSB implementations were made up of several stand-alone services (SOS-server, SAS-server, database, etc.) linked together only by JMS middleware, without Java EE and EJBeans. Though suitable for prototyping these implementations did not meet operational requirements: especially they lacked in performance and maintainability. During further development the components of the TSB were migrated into the Java EE framework and were consolidated as one single deployable Java EE application, gaining the required robustness for the operational field.

6.3 System operations

Although not all commissioned sensors deliver live data at present, the TSB is going to satisfy the operational requirements at the final project stage. Performance tests show that the TSB can process and store about 3500 sensor samples per second, six times more than the expected sensor data peak in case of a tsunami. The data delivery of the SOS is capable of providing 100 000 sensor samples in three seconds, satisfying GITEWS' requirements very well (database filled with 140 million samples, consuming 30 GiB). The required performance will even hold for the integration of external sensors, which increases the number of sensors up to 200 (tide gauges of the Intergovernmental Oceanographic Commission – IOC, Indonesian tide gauges, etc.).

Since 2009 the TSB operates at the BMKG[3] tsunami warning center in Jakarta, with an availability of 0.996 for the last half of 2009. The ongoing activities for hardware redundancy will increase the availability further.

7 Conclusions

This paper has presented the concept, architecture, and implementation of the integration platform TSB. The SOA approach has partitioned the system GITEWS into different layers, following the principle of *separation of concerns*. The TSB, as intermediate integration platform, enables both a flexible integration mechanism for sensors as well as a stable and uniform access for the client applications.

[3]Badan Meteorologi Klimatologi dan Geofisika, http://www.bmkg.go.id, last access: May 2010

By applying SWE as a service framework, the project included a suitable set of services and an appropriate data model from the start. This avoided the need for developing our own service models and data specifications (and the resulting time-consuming negotiations among the project partners). Although only perfect for time series, the maturity of the OGC[1] standards for SOS, SAS, and O&M provided benefit to the project by stabilizing the interfaces at an early project stage. This results in a service and sensor specification that is "of one piece". The aim to provide a universal and standardized sensor model by the TSB was achieved.

The internal architecture of the TSB is composed of four (logical) components, reflecting its main use cases. This clear functional decomposition enables a straightforward implementation by EJBeans, in which each use case is implemented by a small set of beans (message, session, or entity beans). Finally, the plug-in mechanism realizes the flexible integration of new sensor types at runtime by defining sensor-specific data processing.

The TSB is entirely implemented on basis of Java EE technology satisfying the demand of a mission critical tsunami early warning system. The TSB has proven its reliability and robustness at the BMKG[3] tsunami warning center in Jakarta, where it has been in operation since 2009.

The TSB is not confined to GITEWS only. The TSB is used by another decision supporting systems DEWS[4], which focuses the interoperability of distributed tsunami early warning systems. The TSB also serves for the monitoring system CAWa[5], which provides regional hydrometeorological data in real-time via satellite communication.

Appendix A

Glossary

CGPS	Continuous GPS System
DSS	Decision Support System
EJBeans	Enterprise Java Beans
Hibernate	JPA Implementation
Java EE	Enterprise Edition
JBoss	JBoss Application Server
JBoss Messaging	JMS Implementation for JBoss
JMS	Java Message Service
JPA	Java Persistence API
JPQL	Java Persistence Query Language
OBU	Ocean Bottom Unit
O&M	Observations and Measurements
OpenGIS Standards	Standards of OGC[1]
QuakeML	XML representation of seismological data
SensorML	Sensor Model Language
SAS	Sensor Alert Service
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SOS	Sensor Observation Service
SPS	Sensor Planning Service
SWE	Sensor Web Enablement
TSB	Tsunami Service Bus
UDDI	Universal Description, Discovery and Integration
URN	Unified Resource Name
WNS	Web Notification Service
WPS	Web Processing Service
WSDL	Web Services Description Language
XMLBeans	XML Java Binding
XMPP	IETF RFC 3920-3923: extensible Messaging and Presence Protocol.

[4]Distant Early Warning Systems, http://www.dews-online.org, last access: May 2010.

[5]Water in Central Asia (CAWa), http://www.cawa-project.net, last access: May 2010.

Acknowledgements. The GITEWS project (German Indonesian Tsunami Early Warning System) is carried out through a large group of scientists and engineers from different German research institutes under the leadership of the GFZ, German Research Centre for Geosciences. Funding is provided by the German Federal Ministry of Education and Research (BMBF), Grant 03TSU01, publication no. 118. We thank the participating research groups for their discussion and integration efforts, in particular the GITEWS workgroups: Decision Support System (DSS), seismology, oceanography, and continuous GPS. We also thank the editor and the reviewers for their comments and reviews that helped to improve the manuscript.

Edited by: A. Rudloff
Reviewed by: S. Scheer and T. De Groeve

References

Behrens, J., Androsov, A., Babeyko, A. Y., Harig, S., Klaschka, F., and Mentrup, L.: A new multi-sensor approach to simulation assisted tsunami early warning, Nat. Hazards Earth Syst. Sci., 10, 1085–1100, doi:10.5194/nhess-10-1085-2010, 2010.

Botts, M., Percivall, G., Reed, C., and Davidson, J.: OGC Sensor Web Enablement: Overview And High Level Architecture, Open Geospatial Consortium, available at: http://portal.opengeospatial.org/files/?artifact_id=25562 (last access: May 2010), 2007.

Chu, X., Kobialka, T., Durnota, B., and Buyya, R.: Open Sensor Web Architecture: Core Services, in: Proceedings of the 4th International Conference on Intelligent Sensing and Information Processing, Bangalore, India, ICISIP 2006, IEEE Press, Piscataway, New Jersey, USA, ISBN 1-4244-0611-0, 98–103, available at: http://www.gridbus.org/papers/ICISIP2006-SensorWeb.pdf (last access: May 2010), 15–18 December 2006.

Erl, T.: SOA: principles of Service Design, Prentice Hall, 2008.

Josuttis, N. M.: SOA in Practice: The Art of Distributed System Design, O'Reilly, 2007.

Klopfer, M.: Interoperability & Open Architectures: An Analysis of Existing Standardisation Processes & Procedures, OGC, available at: http://portal.opengeospatial.org/files/?artifact_id=10594 (last access: May 2010), 2005.

Kobialka, T., Buyya, R., Deng, P., Kulik, L., and Palaniswami, M.: Sensor Web: Integration of Sensor Networks with Web and Cyber Infrastructure, in: Handbook of Research on Developments and Trends in Wireless Sensor Networks: From Principle to Practice, edited by: Jin, H. and Jiang, W., ISBN: 978-161-520-701-5, IGI Global, USA, available at: http://www.buyya.com/papers/SensorWeb2009Chapter.pdf (last access: May 2010), February 2010.

Kruchten, P.: Architectural Blueprint – The "4+1" View Model of Software Architecture, IEEE Software, 12(6), 42–50, available at: http://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf (last access: May 2010), 1995.

Kunz, S., Usländer, T., and Watson, K.: A Testbed for Sensor Service Networks and the Fusion SOS: towards plug & measure in sensor networks for environmental monitoring, 18th World IMACS/MODSIM Congress, Cairns/Australia, 973–9, available at: http://www.mssanz.org.au/modsim09/C4/kunz.pdf (last access: May 2010), 2009.

McGovern, J., Ambler, S., Stevens, M., Linn, J., Sharan, V., and Jo, E.: Practical Guide to Enterprise Architecture, Vol. 1, Prentice Hall, http://www.cs.umt.edu/u/wright/GIF/PGEADraft4.pdf (last access: May 2010), 2001.

MiniSeed: Seed, Reference Manual, V 2.4, Appendix G (MiniSeed), available at: http://www.iris.washington.edu/manuals/SEEDManual{_}V2.4.pdf (last access: May 2010), 2009.

Moodley, D., Terhorst, A., Simonis, I., McFerren, G., and van den Bergh, F.: Using the Sensor Web to Detect and Monitor the Spread of Wild Fires,, available at: 52NSensorWeb_Gi4DM_2006.pdf (last access: May 2010), Paper presented at the Gi4DM Symposium at Goa India from 25–30 September 2006.

OpenGIS: OpenGIS® Standards, http://www.opengeospatial.org/standards. In particular:

i) O&M: Observations and Measurements – Part 1 – Observation schema and Part 2 – Sampling Features, Version 1.0, available at: http://portal.opengeospatial.org/files/?artifact_id=22466 resp. id=22467, (last access: May 2010), 8 December 2007.

ii) SensorML: OpenGIS® Sensor Model Language (SensorML) Implementation Specification, Version 1.0.0, available at: http://portal.opengeospatial.org/files/?artifact_id=21273 (last access: May 2010), 17 July 2007.

iii) SOS: Sensor Observation Service, Version 1.0, available at: http://portal.opengeospatial.org/files/?artifact_id=26667 (last access: May 2010), 26 October 2007.

iv) SPS: OpenGIS® Sensor Planning Service Implementation Specification, Version 1.0, available at: http://portal.opengeospatial.org/files/?artifact_id=23180 (last access: May 2010), 2 August 2007.

v) SAS: OGC® Sensor Alert Service Candidate Implementation Specification, Version 0.9, available at: http://portal.opengeospatial.org/files/?artifact_id=15588 (last access: May 2010), 13 May 2006.

vi) WNS: Draft OpenGIS® Web Notification Service Implementation Specification, Version 0.0.9, available at: http://portal.opengeospatial.org/files/?artifact_id=18776 (last access: May 2010), 18 November 2006.

vii) WPS: OpenGIS® Web Processing Service, Version 1.0.0, available at: http://portal.opengeospatial.org/files/?artifact_id=24151 (last access: May 2010), 8 June 2007.

Raape, U., Teßmann, S., Wytzisk, A., Steinmetz, T., Wnuk, M., Hunold, M., Strobl, C., Stasch, C., Walkowski, A. C., Meyer, O., and Jirka, S.: Decision Support for Tsunami Early Warning in Indonesia: The Role of OGC Standards, in: Geographic Information and Cartography for Risk and Crisis Management: Towards Better Solutions, Springer, 2010.

Rudloff, A., Lauterjung, J., Münch, U., and Tinti, S.: Preface "The GITEWS Project (German-Indonesian Tsunami Early Warning System)", Nat. Hazards Earth Syst. Sci., 9, 1381–1382, doi:10.5194/nhess-9-1381-2009, 2009.

Shigeoka, I.: Instant Messaging in Java: The Jabber Protocols, Manning, 2002.

Simonis, I.: OGC Sensor Web Enablement Architecture, Version 0.4.0, Doc-Nr.: 06-021r4, OGC, available at: http://portal.opengeospatial.org/files/?artifact_id=29405 (last access: May 2010), 2008.

Walkowski, A. C.: Active Sensor Collection Service - enhancement of the OpenGIS Sensor Collection Service based on a use case of the flood management, in: Remote Sensing & GIS for Environmental Studies: Applications in Geography, edited by: Erasmi, S., Cyffka, B., and Kappas, M., Proceedings of 1st Göttinger GIS and Remote Sensing Days Environmental Studies, 7–8 October 2004, 323–330, http://www.ggrs.uni-goettingen.de/ggrs2004/CD/Applications_in_Geography/GGRS2004_Walkowski_G323.pdf (last access: May 2010), 2005.