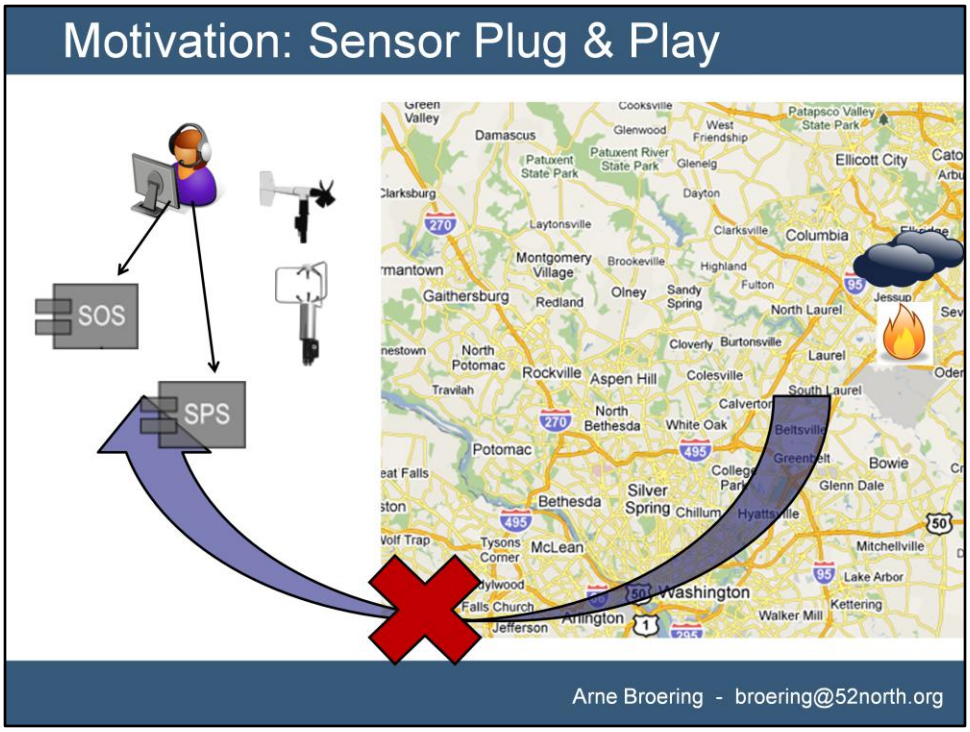
Sensor Interface Descriptors

[OGC 10-134]

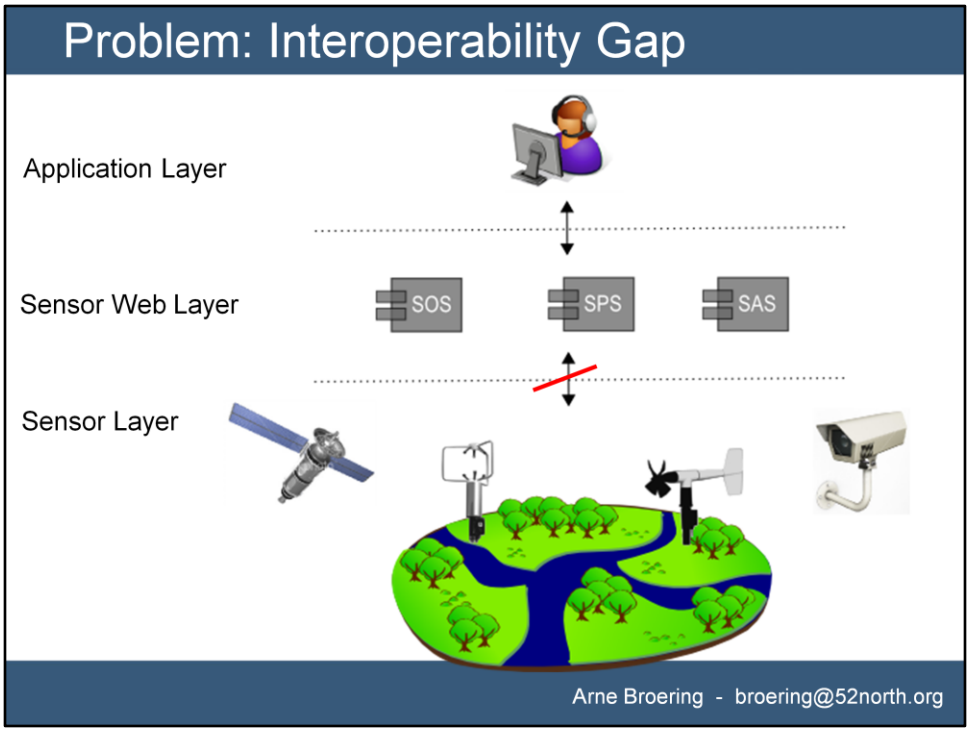
Realizing Sensor Plug & Play within the Sensor Web

Arne Broering

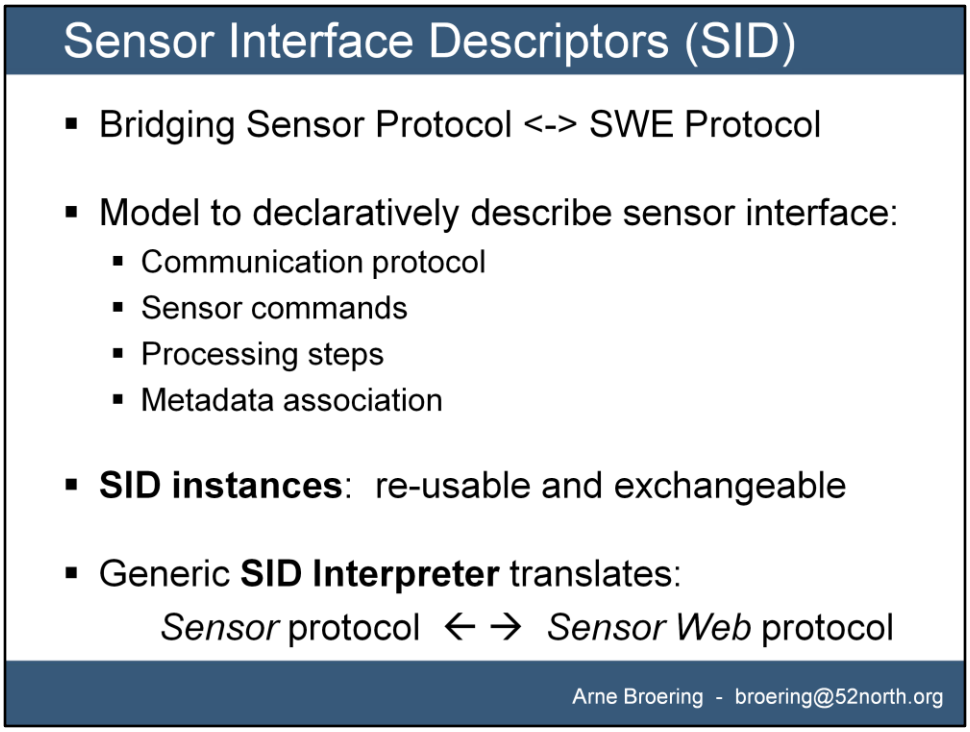
OGC TC Meeting, June 15th, 2010



Assuming, close to Silver Spring, there is some kind of industrial fire which causes a dispersion of pollutants into the air.
Luckily, in this scenario, the disaster management organization has already a Sensor Web in place through which it can access sensors to get an overview of the situation or to compute dispersion models.
However, the density of available wind sensor data is not high enough. So, the disaster management organization decides to deploy new wind sensors on-the-fly in the region.
The newly deployed sensors have to be made available within the SensorWeb so that the existing applications can directly utilize the gathered observations.
So what's necessary is some kind of plug&play of sensors, which currently doesn't work with the SWE framework.



This is due to the fact that SWE service interfaces (SOS, SPS, SAS… here on the Sensor Web Layer) are intentionally designed from an application-oriented perspective, so that *applications* have an interoperable access to the sensor functionality.
So far, the connection between sensors and SWE services is not yet sufficiently described.
On the sensor layer, there is a huge variety of protocols and the connection of those protocols to SWE services is so far usually established, by manually adapting the SWE service implementation.

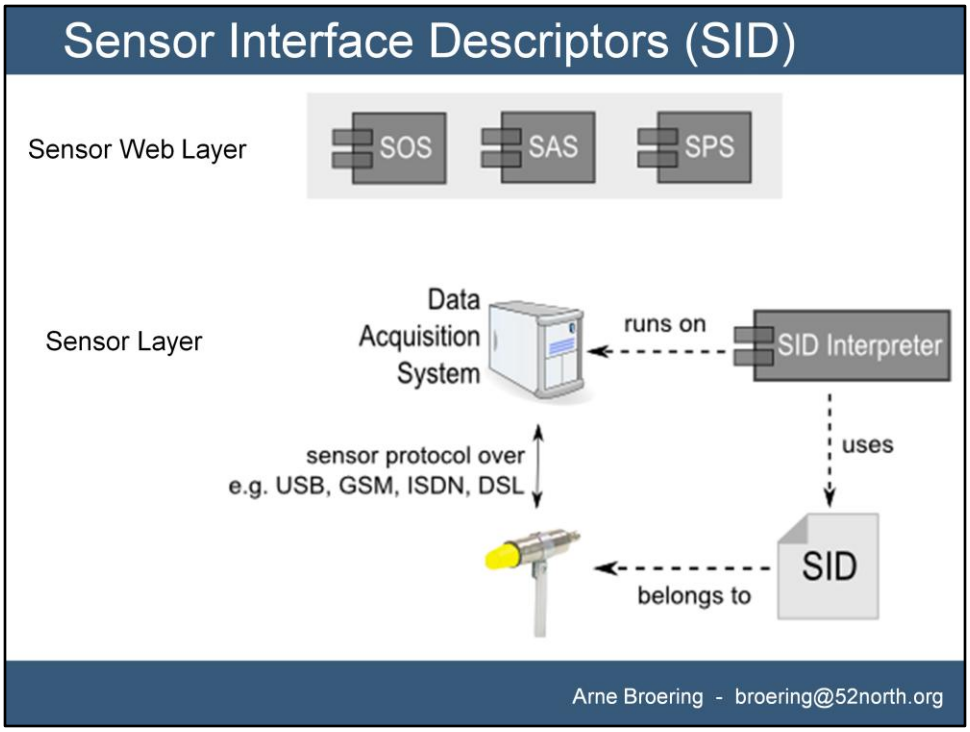


The approach to solve this issue: the concept of Sensor Interface Descriptors, an extension of OGC SWE's Sensor Model Language, to describe the sensor protocol of a particular sensor type in a declarative way.

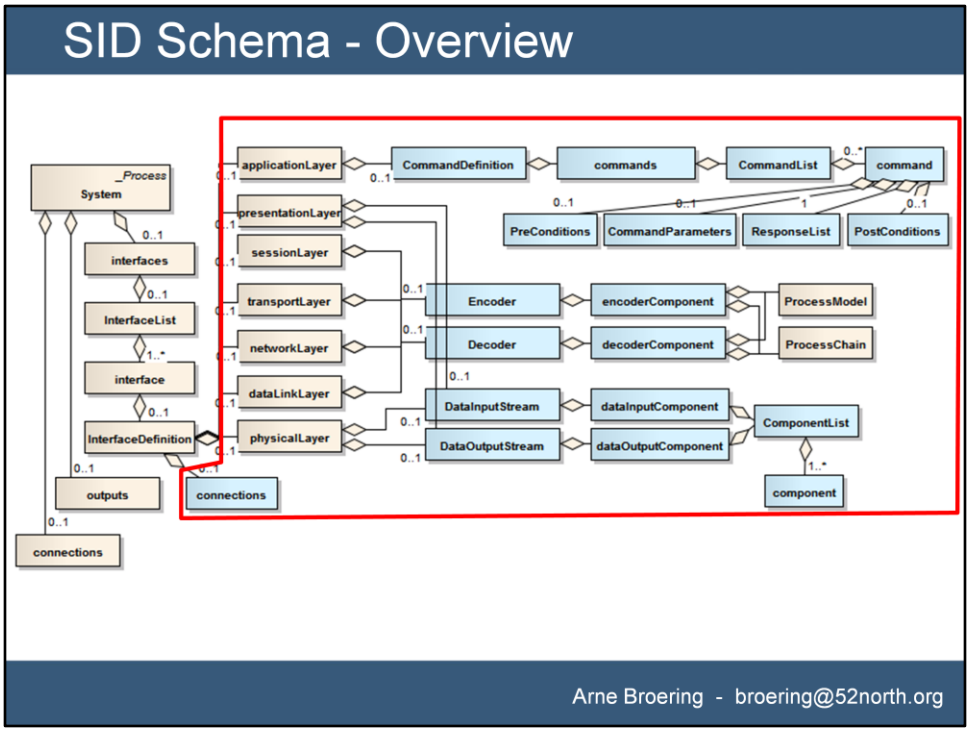
By means of a generic SID interpreter the native sensor protocol can be translated to SWE protocols. The SID interpreter is independent of a particular sensor technology, and can communicate with any sensor whose protocol can be described by a SID. Hence, a repository of SID instances allows users to choose an SID instance which then serves as a “driver” to make the sensor functionality available for the Sensor Web.

Current SWE standards do not deal with actual sensor protocols, and the connection between sensors and SWE services is usually established by manually adapting the internals of the SWE service implementation to the specific sensor interface. Such sensor "drivers" have to be built for each kind of sensor interface, which leads to extensive efforts in developing large-scale systems.

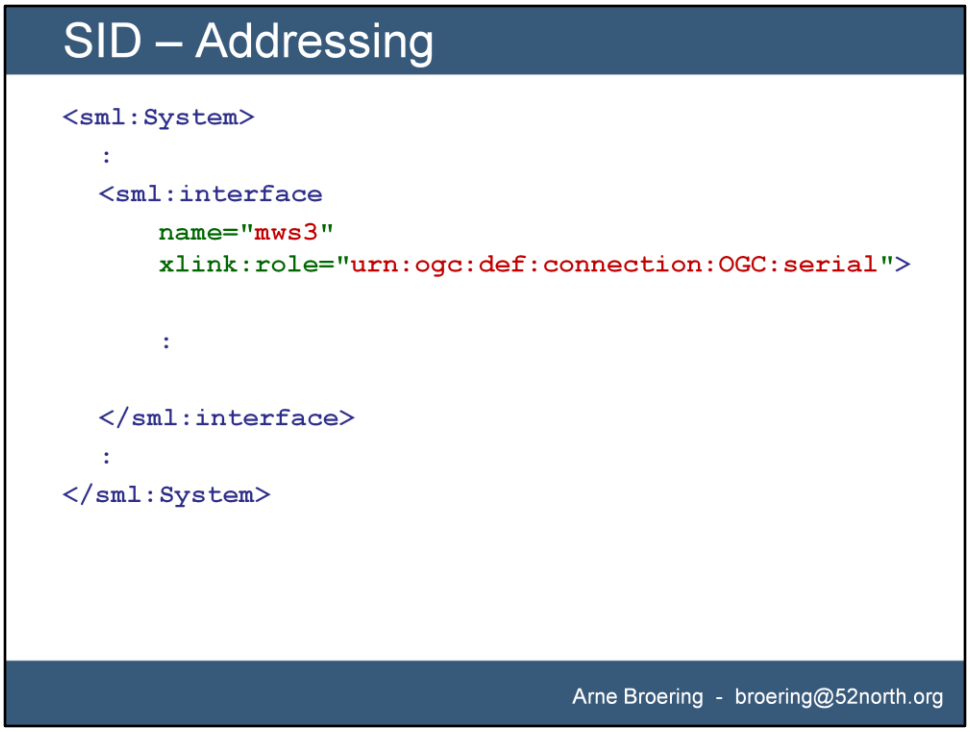
To tackle this issue we have developed a model for Sensor Interface Descriptors (SID) which enables the declarative description of sensor interfaces, including the definition of the communication protocol, sensor commands, processing steps and metadata association. The model is designed as a profile and extension of OGC SWE's Sensor Model Language standard. In this model, a SID is defined in XML for each kind of sensor protocol. SID instances for particular sensor types can be reused in different scenarios and can be shared among user communities. A SID interpreter can be built which translates between various sensor protocols and SWE protocols, hence closing the described interoperability gap. The SID interpreter is independent of any particular sensor technology, and can communicate with any sensor whose protocol can be described by a SID.



This slide shows a typical SWE deployment scenario.
A sensor shall be connected to SWE services (at the top). Usually in between there is some kind of Data Acquisition System, or a sensor network sink.
A sensor communicates with the data acquisition system in its specific sensor protocol over a transmission technology such as ISDN or GSM.
So, to integrate this sensor with the SWE services, we developed a SensorML application schema/profile, called Sensor Interface Descriptors, which can be used to describe the sensor protocol in a declarative way.
So what we need is an SID instance for the sensor. Then, based on the SID schema one can build **sensor technology independent** SID interpreters, which run on the DAS and uses the SID to translate the protocol the SWE world.
The SID interpreter **registers a sensor at a SWE service** and **uploads sensor data to SOS or SAS (or SES**) and is responsible for the opposite communication direction and **forwards tasks received by an SPS to a sensor**.

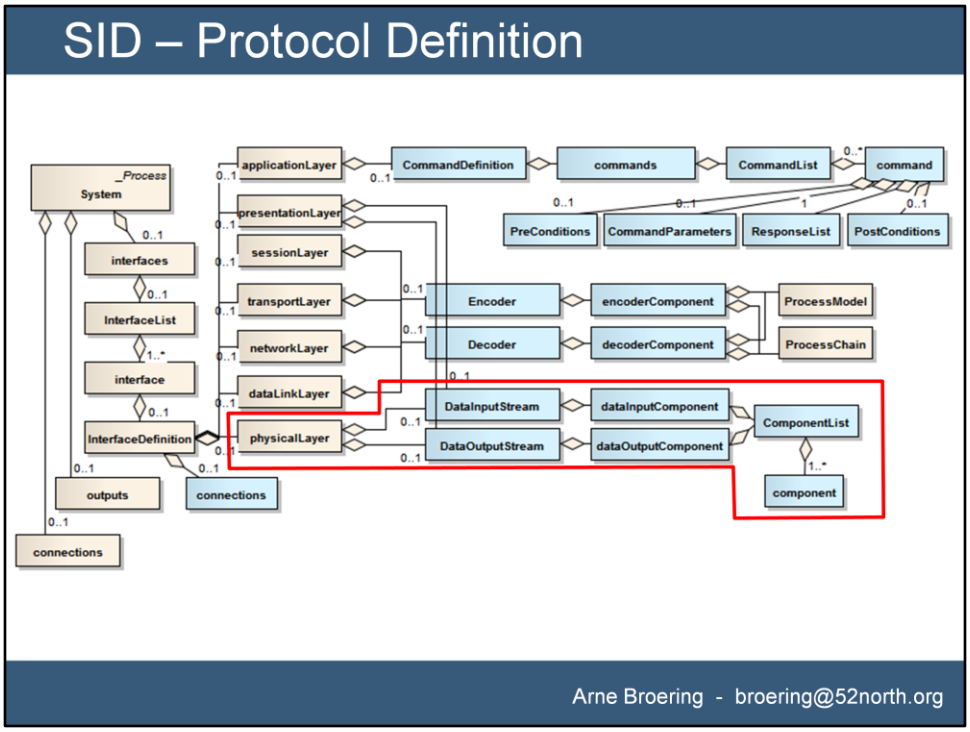


This slide shows an overview of the SID schema.
All the blue classes here are added to the SensorML schema.
The SID (now surrounded in red) is strictly encapsulated within the SensorML structure. This way, it can be easily replaced or removed. This characteristic is important to make it possible to reuse the SID in different use cases for sensors with same interfaces. The ability of reusing SIDs also enables sharing SIDs for particular sensor types among user communities.
For these reasons, the approach developed here, encapsulates all SID specific information within the *sml:interface* element of the *sml:System*.
The **SID schema extends the OSI layer model** and adds certain types to those layers. These types will be explained later on.

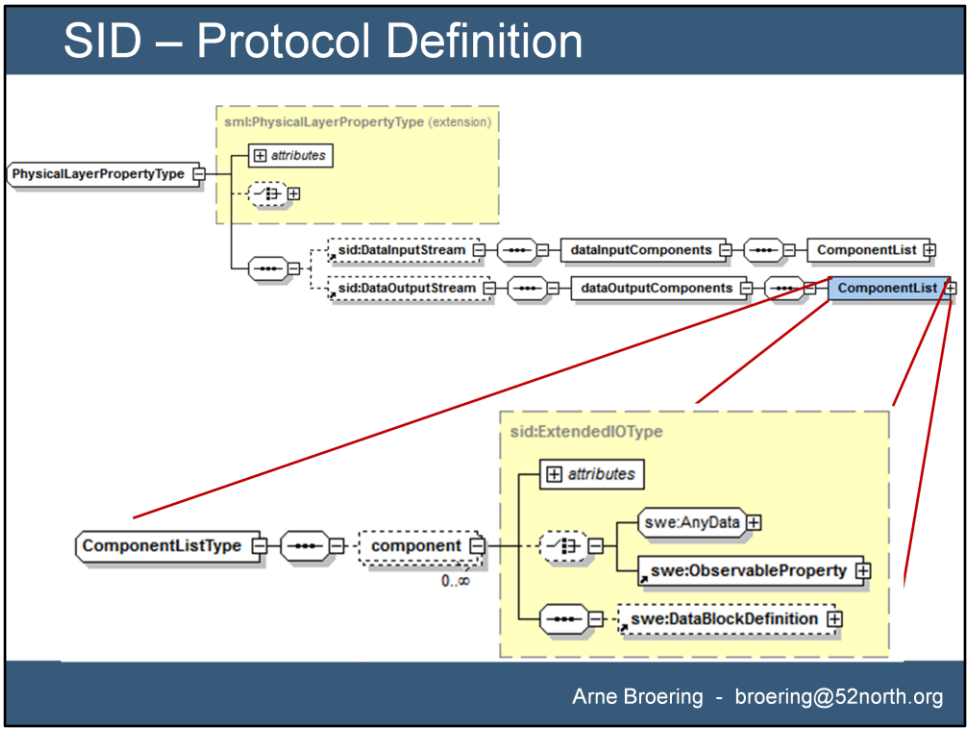


Okay, let's have a closer look at the SID design:

It all starts with establishing a physical connection to the sensor.
This physical connection is established through the OS which runs the SID interpreter.
The type of connection is specified by the *xlink:role* attribute of the *sml:interface* as a URN.
This role attribute together with the *name* attribute of the interface identify the addressing parameters (e.g. port of serial con) which are stored externally in a document accompanying the SID
→ since the SID can be published publicly and the addressing parameters are security relevant.
→the addressing parameters might be OS dependent

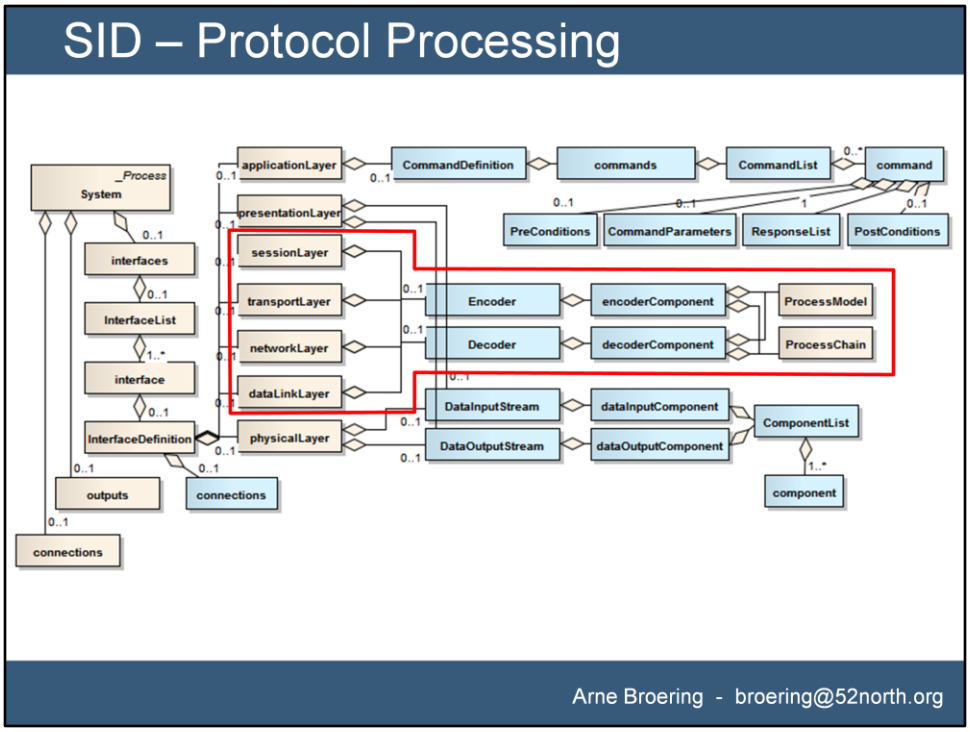


It is essential for the SID, to be able to exactly define the sensor protocol - the structured of raw data streams exchanged between sensor and data acquisition system.
This is defined by the *physicalLayer* element.



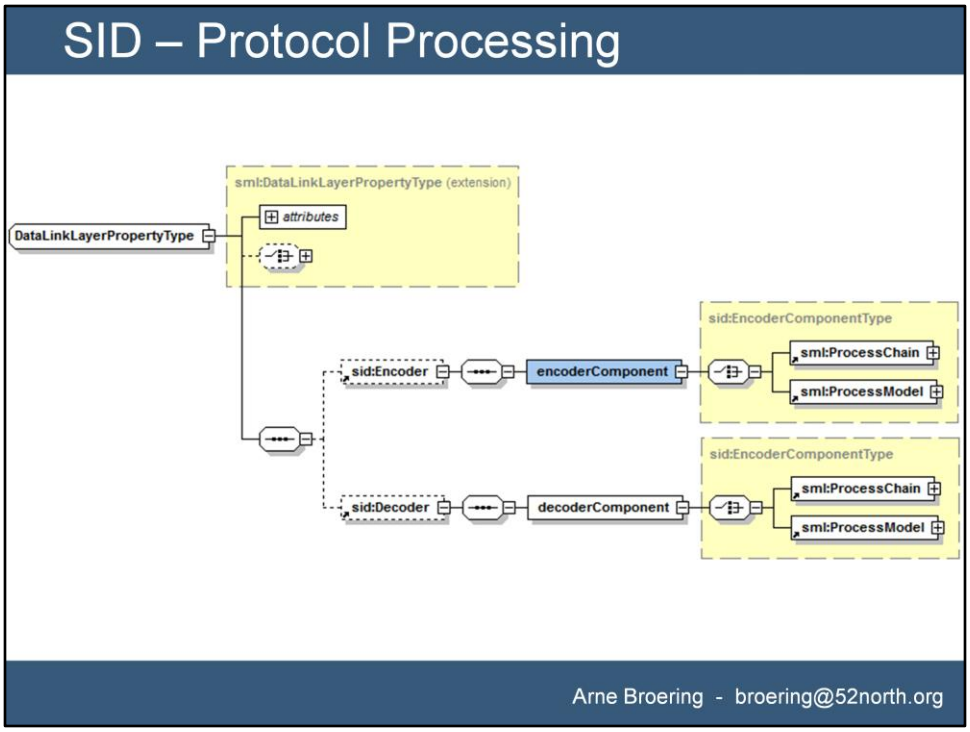
…as shown in this figure, new elements for the data input and data output stream are attached to the layer. The two elements are necessary to **support duplex communication** with sensors.

Each of those data stream definitions (here) contains at the end components which define the data structure. And there we reuse of course existing SensorML / SWECommon types.
The *sid:ExtendedIOType* (here) is a combination of *sml:IOComponentType* and *swe:DataBlockDefinition*. The latter one also allows the specification of an encoding.



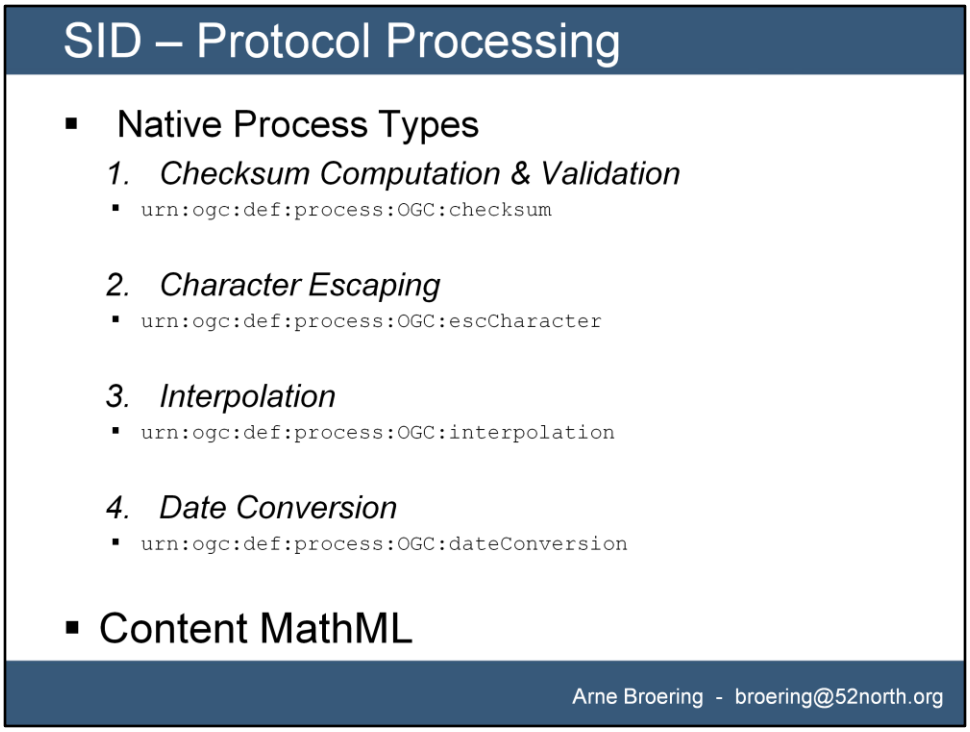
On the upper layers, *dataLinkLayer*, *networkLayer*, *transportLayer*, and *sessionLayer*, protocol processing steps can be defined.

To allow data processing in both directions, from sensor domain to SWE domain and the other way round, elements for *data decoding* and *encoding* are added to each of these layer types.



...this figure shows this extension by example of the DataLinkPropertyType, however, it looks the same for the others.

Instances of the *EncoderComponent* and *DecoderComponent* contain either an *sml:ProcessModel* or *sml:ProcessChain* to define the process.



Due to their significance for sensor communication (from our experience), an SID interpreter shall natively support four process types.
For each of those process types a URN is defined, which can be specified in the ***method*** property of the process model.

Checksum

Checksums need to be computed and validated for reliable sensor communication

Date Conversion

If sensors tags data with time in a different format, a conversion is needed

Character Escaping

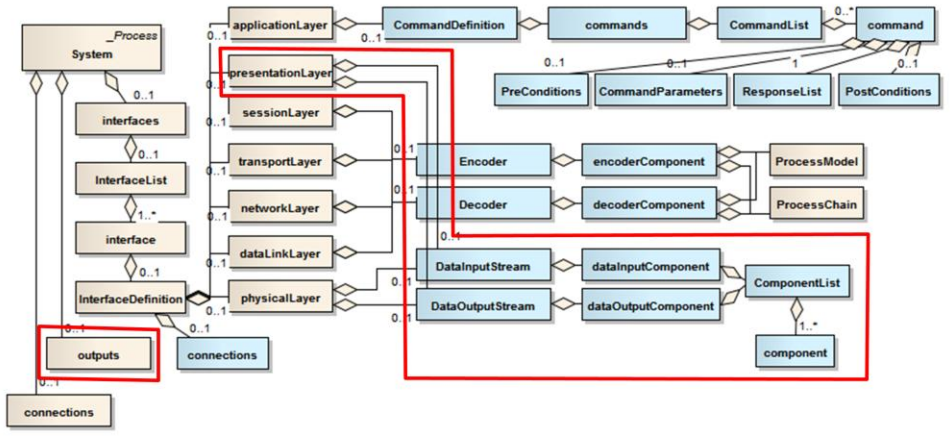
process type for removing and adding escape characters

Interpolation

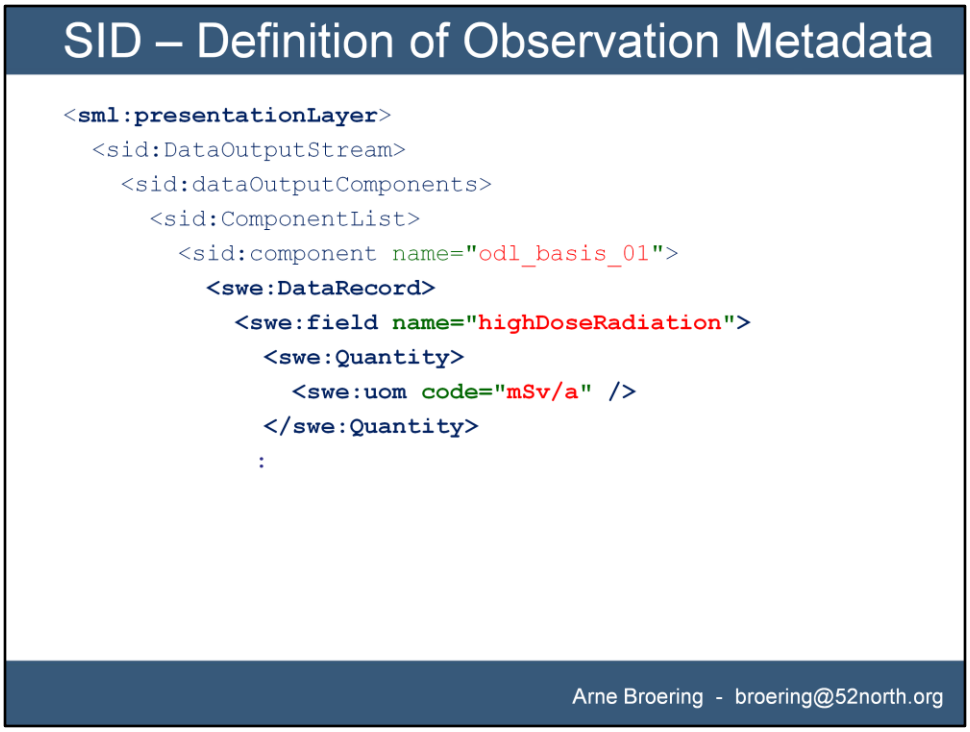
Interpolations need to be computed for example to transform raw sensor data to observations (e.g. an electric current returned by a detector to an actual measurement value)

Besides those four natively supported processes, other process methods can be incorporated by describing them inline using Content MathML.

SID – Definition of Observation Metadata



Arne Broering - broering@52north.org

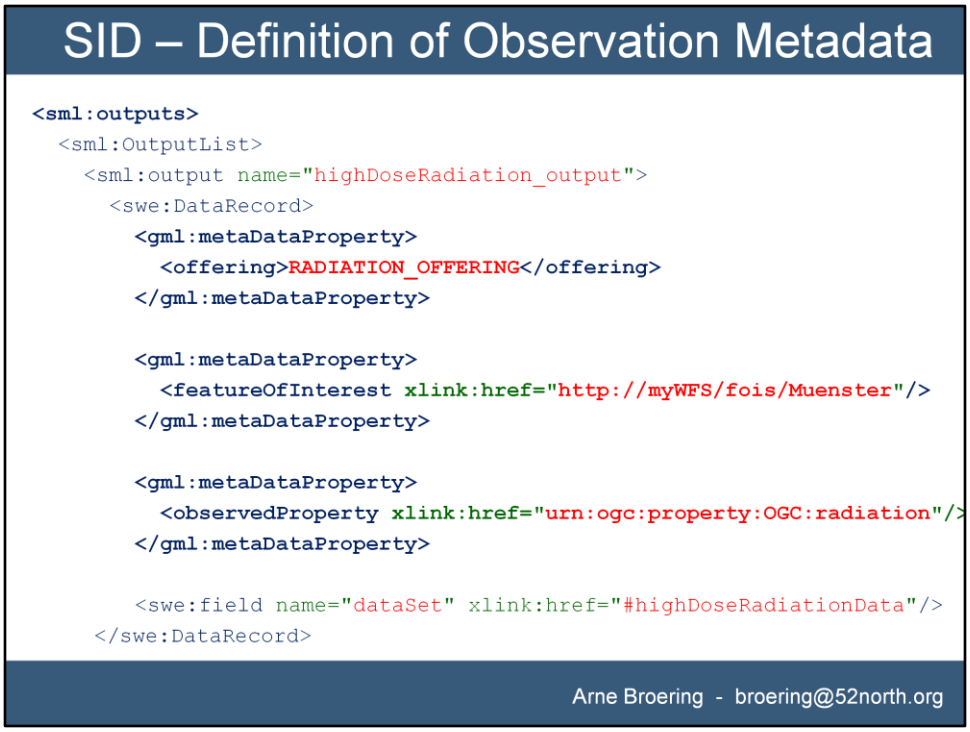


The data, resulting from the preceding processing steps, has to be associated with certain O&M metadata before it can be forwarded to an SOS.

Measured data needs to be associated with units of measure so that an interpretation is possible. Further, the data needs to be linked to the observed property and feature of interest.
Association of a unit of measure is done on the *presentationLayer* as here.

PresentationLayer is structured in the same way as the physicalLayer.

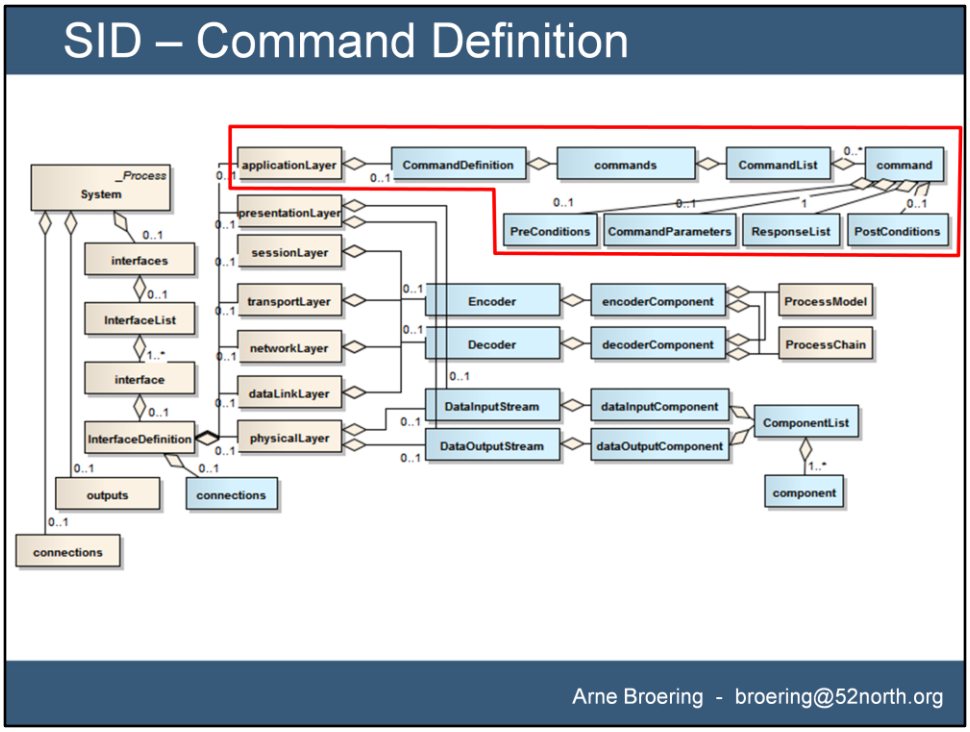
The *DataOutputStream* element is used to describe the data coming from the sensor on this layer. For example, the *component* named '*odl_basis_01*' contains the field descriptions of the measured sensor data. The *field* named '*highDoseRadiation*' represents the radiation data measured in millisievert per year ('mSv/a').



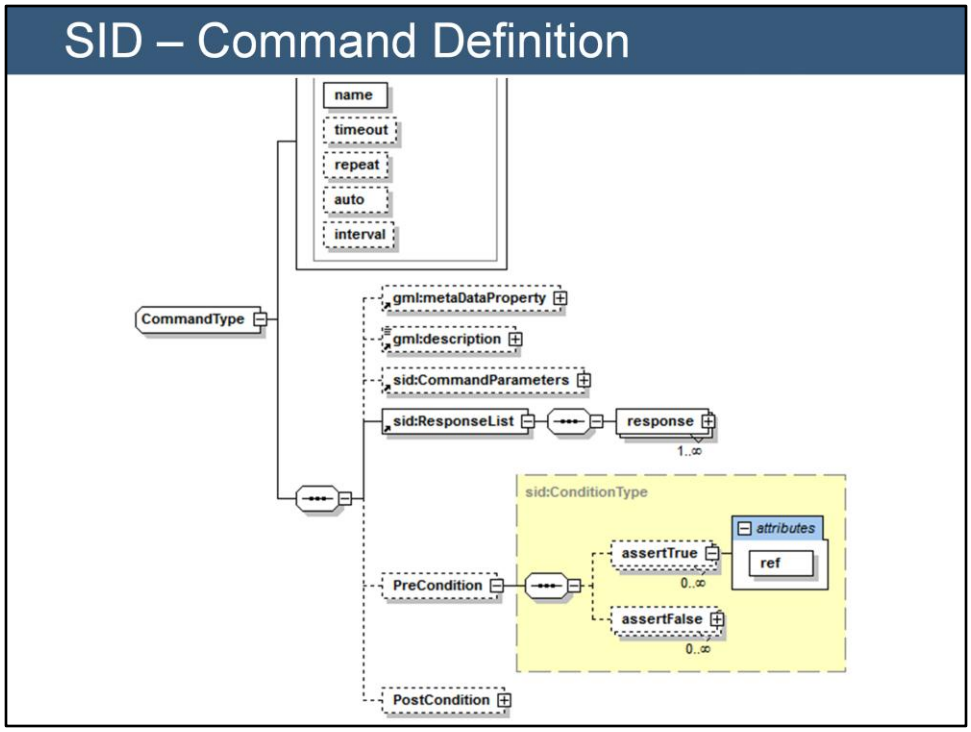
In the *sml:outputs* element of the *sml:System*, the sensor data is linked to the observed property, and to the feature of interest and to an observation offering which is necessary for being able to execute the *InsertObservation* operation of the SOS.

As shown here, the *gml:metaDataProperty* is used for this linking. The feature of interest and the observed property are referenced by the means of the *xlink:href* attribute. For the observation offering, its service side identifier is given.

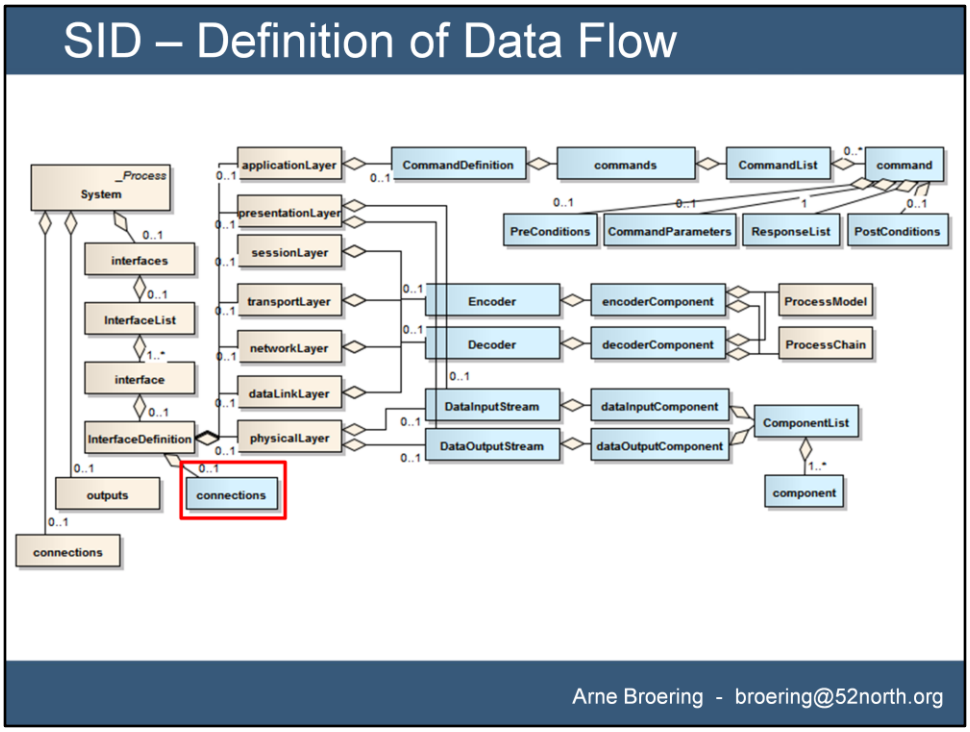
The *sml:outputs* element is not part of the SID, since it is not a sub-element of the *InterfaceDefinition*. The information contained in the *sml:output* elements is intentionally kept out of the SID, due to the fact that the linkage of a sensor to a feature of interest, an observed property, and an observation offering is dependent on the particular use case, not the sensor interface.



The *applicationLayer* is used here to define the commands accepted by the sensor.
Parts of **the command definitions can be used by an SPS** so that it can provide information to clients how to task a sensor...



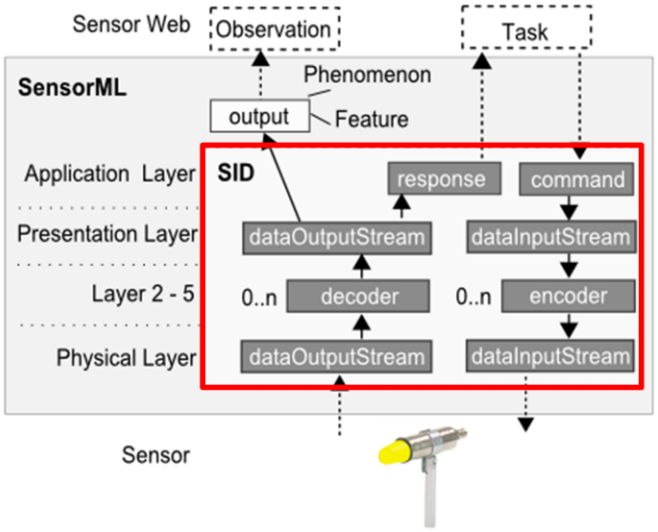
…this figure shows the structure of the SID CommandType.
Each command has a unique *name*.
The *timeout* attribute specifies a time in milliseconds until a response of the sensor has to be received.
The *repeat* attribute can be used to define whether a command call has to be repeated before a response is received.
The *auto* attribute defines whether the command call is automatically executed every n seconds. The number of seconds is defined by the *interval* attribute.
The element *sid:CommandParameters* contains the parameters of the command and is of type *sml:IoComponentPropertyType* and uses a *swe:DataRecord* to list the parameters in *swe:field* elements.
Under consideration that a command might have different responses with different meanings, the element *sid:ResponseList* contains the possible responses of the command.
The *response* element is of type *sid:IoComponentType* which extends the *sml:IoComponentPropertyType* by adding the *valid* attribute of type *xs:boolean*. This attribute marks whether the response represents a successful or unsuccessful execution of the command.
The element *PostCondition* shall be used to reference commands which are executed after execution of this command. The *PreCondition* element references commands which shall be executed before this command. The sub-elements *assertTrue* and *assertFalse* of *PreCondition* and *PostCondition* shall be used to demand a successful or unsuccessful execution of those referenced commands.



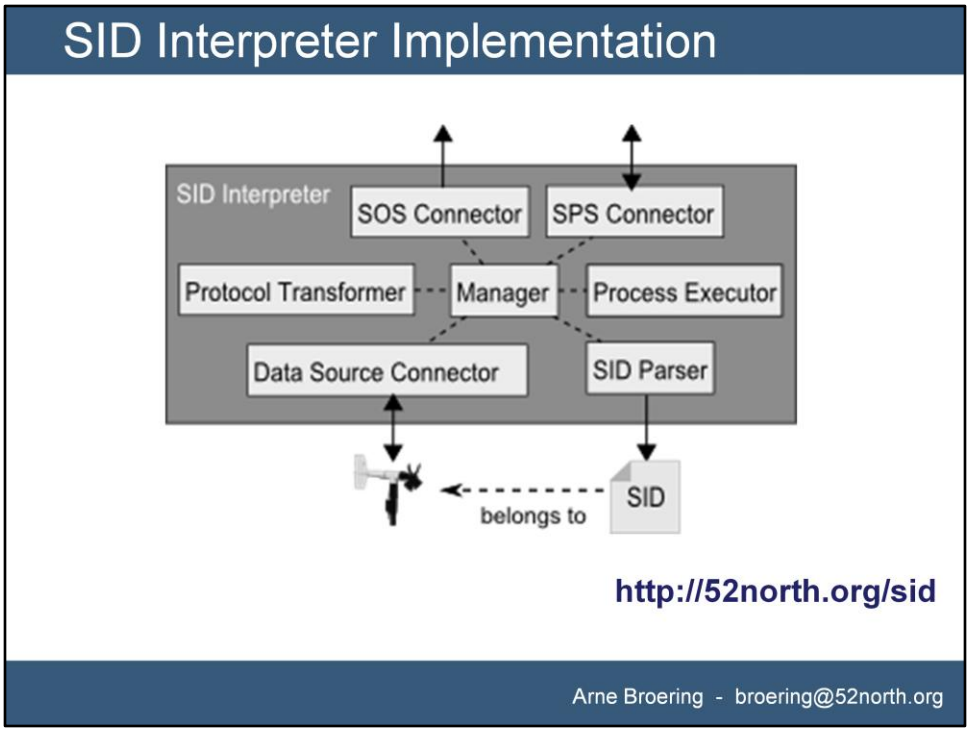
Also the connections between the different components of the SID have to be described within the SID. That's why we reuse the *sml:connections* element and associate it with the *sml:InterfaceDefinition*.

These connections model the data flow between SENSOR and the SWE world through the SID as shown on the next slide…

SID – Definition of Data Flow

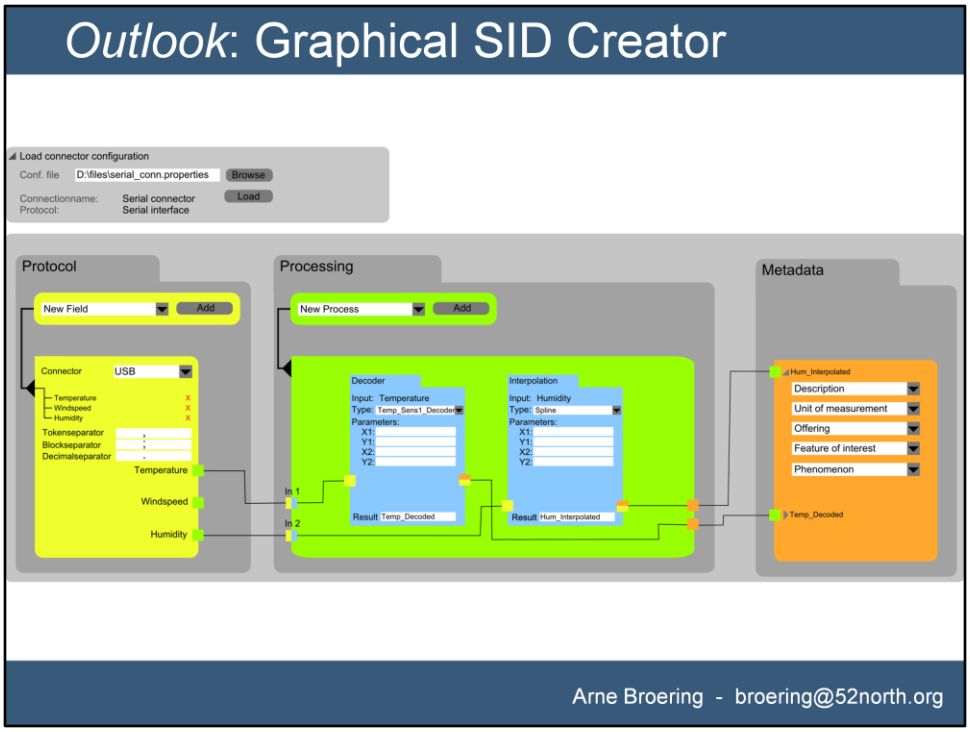


Arne Broering - broering@52north.org



Besides the developed SID schema we also have a first SID Interpreter implementation.

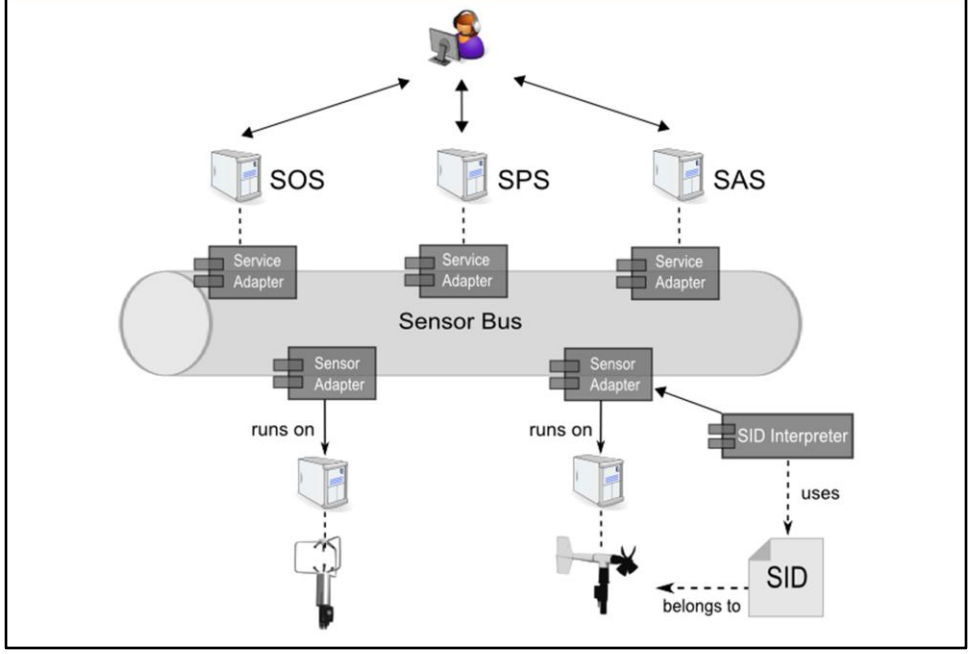
It is based on the OSGI framework and hence consists of several pluggable and loosely coupled Bundle components.



We are currently developing a graphical tool which helps you in creating SIDs.

A very first draft of this GUI is shown here.

Outlook: Combination with Sensor Bus



Questions?

Thank you!

Arne Broering

52north
exploring horizons

ifgi
Institute for Geoinformatics
University of Münster

ITC

SID project:	**http://52north.org/sid**
Sensor Web community:	http://52north.org/SensorWeb
Sensor Web lab:	http://swsl.uni-muenster.de