SensIDL [1] : Towards a generic framework for implementing sensor communication interfaces

Christoph Rathfelder, Institut für Mikro- und Informationstechnik der Hahn-Schickard-Gesellschaft e.V. (HSG-IMIT), Villingen-Schwenningen, Germany;
christoph.rathfelder@hsg-imit.de

Emre Taspolatoglu, FZI Forschungszentrum Informatik am Karlsruher Institut für Technologie, Karlsruhe, Germany;
taspolat@fzi.de

C Software for smart integrated systems

Motivation

One essential element of a smart system is its ability to communicate [1]. Although being a mandatory building block of smart sensor systems, the implementation of the communication is often seen as a necessary evil and developers want to focus on innovative sensing capabilities and techniques or new algorithms for data processing, fusion and analytics. Supporting the implementation of the communication interfaces, will allow developers to concentrate on the development of new sensing techniques and algorithms, thus increasing the overall quality of the smart sensor systems. With this presentation we want to give some insides into our current research and development of a framework called SensIDL “Sensor Interface Definition Language” (http://www.sensidl.de) targeting the efficient specification and development of communication interfaces for smart sensor systems.

Application Scenarios

The SensIDL development framework targets sensor developers as well as the developers of IT-systems connected to these sensors and processing the collected data. In the following, we highlight the improvements that can be gained by using the SensIDL framework on the side of sensor developers and software developers.

[1] *This work is funded by the Federal Ministry of Economic Affairs and Energy of Germany as IGF-Vorhaben 18363 N.*

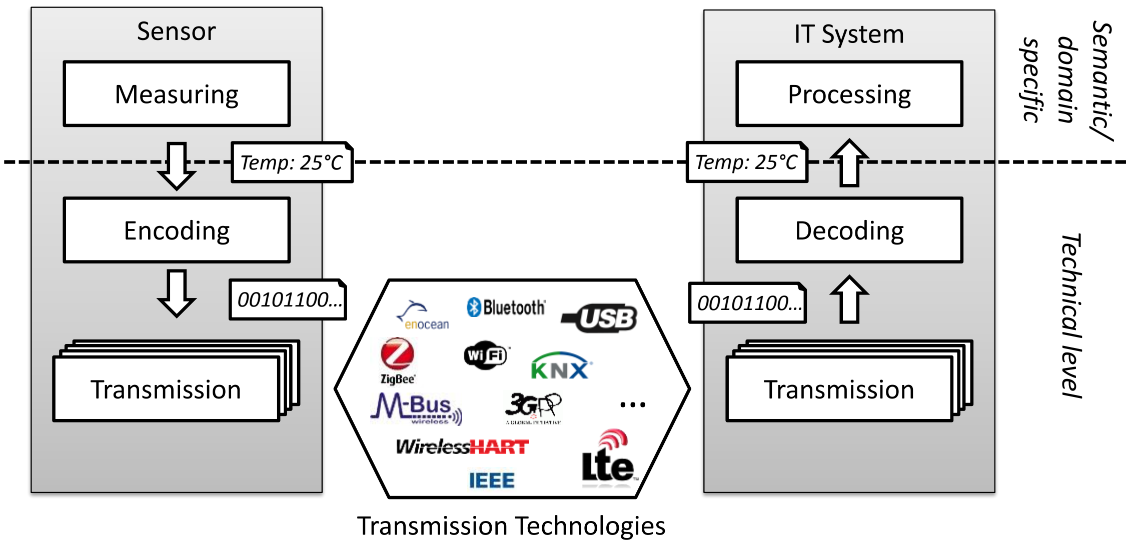

Figure 1: Scenario Overview

Figure 1 sketches the mandatory processing steps when sensor data should be processed on connected IT-Systems. After reading the data measured by the sensor module (e.g. the temperature) this data has to be encoded into a series of bits that can later be sent to the IT-system. Often a sensor system contains multiple sensor modules and thus has several measurements that have to be grouped, but also have to be encoded. For example, a meterorological sensor system measures among others humidity, temperature, and atmospheric pressure. The sensor developer has to define and implement the encoding of data, which can range from a sensor-specific binary format over a simple ASCII encoding of the data values up to structured data formats like XML or JSON. The encoded data is then handed over to a transmission technology that is responsible to transfer the set of bytes to the receiving IT-system.

After receiving the set of bytes within the IT-system, the data has to be decoded. To implement the decoding, the software developer needs to know the encoding mapping implemented within the sensor system. Without this knowledge the software developer can not extract the semantically usefull information from the set of bytes. For this reason, the documentation of the sensor system has to include a semantical description of the measured data along with a technical description of the encoding mappings.

With the SensIDL framework, we target the reduction of implementation effort and thus increasing the efficiency for both the sensor developers and the developers of the IT-systems, which are often different individuals from different companies.

The ***sensor developer*** is supported by an editor allowing to describe the data on a semantical level. Based on this desciption, the code for encoding the data as well as a semantically enriched API to set the data is automatically generated. Thus the sensor developer does not have to take care for the encoding mappings and algorithms. Next to the input for the code generation, the sensor description can also act as formal documentation of the sensor, thus the sensor developer does not have to write and maintain a separate documentation.

In the same way, the ***software developer of the IT-system*** can use the sensor description to generate the code for decoding the data and a semantically enriched API to access it. Since the code generators of the SensIDL framework will be implemented for different platforms and programming languages, the sensor system and IT-system are not limited to the same programming language or plattform. For these reasons the software developer profits from the formal specification and description of the sensor interface and the automated generation of decoding code with a semantically enriched API to access the information. In the following section, we provide some more insides into the SensIDL framework.

The Open Source Project SensIDL

SensIDL is an open source development framework based on a model-driven software development (MDSD) approach supporting the sensor developers in specifying and implementing the communication interfaces. Figure 2 provides an overview on the different artifacts the SensIDL framework consists of:

1) **SensIDL Language**
   The SensIDL language is a dedicated domain specific language (DSL) [2] acting as an interface description language for specifying the data provided by a smart sensor. In addition to the specification of data types and structures, it allows to define additional meta-data, like burst or single value transmission, push or poll communication, etc., that is required when implementing the communication layer. Dedicated graphical and textual editors based on the Eclipse Modelling Framework [3] allow an easy and intuitive specification of sensor interfaces.

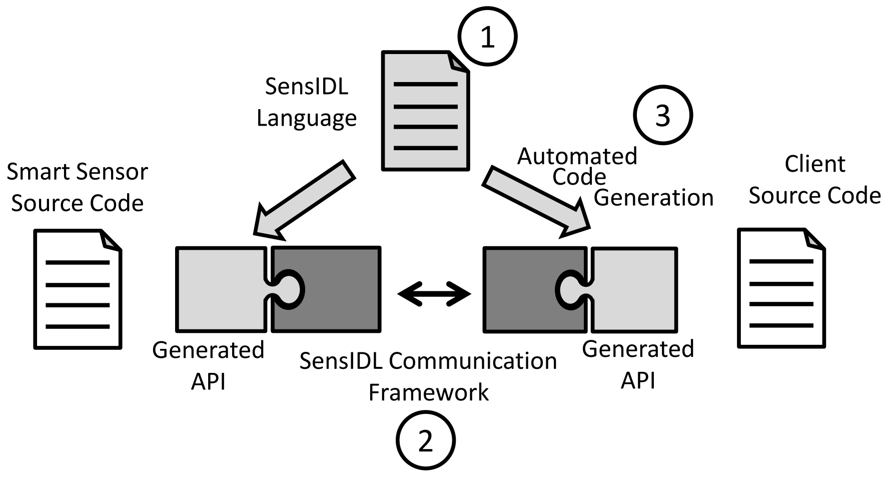

Figure 2: Overview on the SensIDL Framework

2) **SensIDL Communication Framework**

   The communication framework provides basic communication functionalities like marshaling, message handling, etc. through a generic API. However being intuitively applicable, sensor developers require a more sensor specific API that is automatically generated by the code generators described in the next sub section. This separation into generic framework code and application specifically generated code is a frequently used pattern in MDSD frameworks [2].

3) **SensIDL Code Generators**

   The code generators take the interface specification based on the SensIDL language as input to automatically generate a sensor- or rather interface-specific API on top of the generic communication framework. Providing a very specific and type safe API reduces the risk of implementation errors resulting in different data interpretations or parsing. Since the generators can be used to generate the sensor-side code as well as the client-side code, the SensIDL does not only ease the sensor implementation but also the implementation of client code to access the sensor data.

Model-Driven Engineering Approach

Model-driven software engineering approach combines essentially two main technology concepts and applies this combination on models of the considered systems. These technologies are the domain-specific languages (DSLs) as well as transformation engines and generators [4]. Instead of discussing the general concepts and advantages of model-driven engineering approach, such as separation of concerns, level of abstraction, less-error prone and faster development, better validation aspects, etc., [5]. In the following, we map distinct model-driven concepts to the key SensIDL artifacts and discuss the planned contributions:

1) **Domain-Specific Languages**

   A domain-specific language formalizes the structure, behavior and requirements of an application within exact domain boundaries [4]. They have a more limited focus than multipurpose programming languages like object-oriented languages, but they allow developers to concentrate only on the relevant aspects and conditions of their systems. Therefore it is common to apply DSLs especially in scenarios where, the complexity of certain implementation details and complexity should be hidden [2]. In the SensIDL project a dedicated interface description language allows to concentrate on specifying the data that should be transferred without the need to specify the encoding and transmission techniques in parallel. The SensIDL language should ease the work of sensor developers in defining and describing the specifications of the sensor communication interfaces, the types, and the structure of the provided data that is provided by the sensor. By using a specific definition language and dedicated utilities such as graphical or textual editors based on the defined DSL, sensor developers benefit from straightforward and explicit definition of sensor interfaces, so that they can focus on their own competences, thus increasing the development productivity [2]. Another important aspect of using a DSL for the task is its ability to reference a meta-model that can be defined in a way to support a variety of different types of interface data structures. With utilizing models in the core of the SensIDL language, we benefit from separating the concerns and a greater level of abstraction regarding the only to be transferred sensor data as well possibility to validate the defined language in early development phases [5].

2) **Code Generators**

   Code generators and transformation mechanisms are responsible for the practical appliance of the model-driven engineering approach. They evaluate the aspects of models and use them as inputs to construct further elements such as generating source code, which can be deployed and executed, or transforming into other models, which can be again evaluated under a different viewpoint [4]. Generating the code from an input can be interpreted as the compilation of a general-purpose language, but in the context of domain-specific languages [2]. Our project makes use of these aspects of the model-driven approach and uses the definitions of sensor interface data specifications built by the SensIDL language as input and generates so called glue codes. The glue code closes the gap between sensor- and application side communication APIs and the generic communication framework. The central generic communication framework supports various communication technologies and standards as well as is responsible for the common functions of transmitting data. By generating the sensor- and receiver-specific communication APIs, we can ensure the consistency between the deployable implementations and the models representing the necessary functional requirements [4]. Using generation techniques furthermore, we benefit from a few more aspects such as less error-prone and type-safe implementation of complex and differentiating sensor-/application-side APIs, reusability of the existing components and the generic communication framework as well as cost-effective, shorter time-to-markets [5] [6].

Conclusion and Outlook

In this paper and the belonging presentation, we gave an overview on the recently started research and development project SensIDL (http://www.sensidl.de), whose aim is the development of an integrated tool set supporting sensor developers in specifying and implementing data interfaces of sensor systems. We introduced the envisioned application scenarios and sketched the building blocks the later SensIDL development framework will consist of. Additionally, we provided an introduction into the model driven software engineering approach and its practical application within the SensIDL project.

The next development steps are the detailed specification and implementation of the interface description language including the realization of an editor allowing an ease and intuitive specification of sensor interfaces. Based on this DSL and a first manual reference implementation, the code generators and generator templates are derived. Finally, the different artifacts are combined and integrated as SensIDL tool using the eclipse framework as a basis.

References

[1] EPoSS - The European Technology Platform on Smart Systems Integration, "Strategic research agenda of the european technology platform on smart system integrartion," 2013. [Online]. Available: http://www.smart-systems-integration.org/public/documents/ publications/EPoSS%20SRA%20Pre-Print%20September%202013. [Accessed Jan 2015].

[2] M. Fowler, Domain-Specific Languages, Addison Wesley, 2010.

[3] Eclipse Foundation, "Eclipse Modeling Project," [Online]. Available: http://www.eclipse.org/modeling/. [Accessed October 2014].

[4] D. C. Schmidt, "Model-Driven Engineering," *IEEE Computer,* vol. 39, no. 2, pp. 25-31, 2006.

[5] T. Stahl und M. Völter, Modellgetriebene Softwareentwicklung, Heiderlberg: dpunkt.verlag, 2005.

[6] C. Bunse, H.-G. Gross and C. Peper, "Applying a Model-based Approach for Embedded System Development," in *33rd EUROMICRO Conference on Software Engineering and Advanced Applications*, Lubeck, 2007.