Hochschule Ostwestfalen-Lippe
University of Applied Sciences

inIT – Institute Industrial IT
Hochschule Ostwestfalen-Lippe
University of Applied Sciences

Liebigstraße 87
D-32657 Lemgo

Professorship of Discrete Systems
Prof. Dr.-Ing. Volker Lohweg
Tel.: 05261 / 702-2408
E-Mail: volker.lohweg@hs-owl.de

Discrete Systems Group
M.Sc. Christoph-Alexander Holst
Tel.: 05261 / 702-5592
E-Mail:
christoph-alexander.holst@hs-owl.de

Technical Report

Automated Fusion System Design and Adaptation Implementation

Author	Date	Version
Christoph-Alexander Holst	24.03.2017	0.0

Contents

1 Introduction

This published prototype is an implementation of the automated fusion system design proposed in the journal article [FMH+17]. The implementation orchestrates a distributed information fusion system, i. e., it identifies features and attribute supported by the system. The automated orchestration is carried out at a central device called *system manager*. Basic elements of the fusion system are *intelligent sensors*. Intelligent sensors monitor a system using elementary sensors (e.g., temperature sensors or acoustic sensors) [MDL+16, FML16]. The sensor signals of all intelligent sensors are gathered and fused to evaluate the condition (i.e., health) of the monitored system. An intelligent sensor is additionally equipped with processor units, memory, and communication interfaces. It is self-adaptable and self-aware. An intelligent sensor hosts a semantic self-description stating available elementary sensors and algorithms. Algorithms are used to extract certain features from sensor signals. This implementation uses the *Raspberry Pi 3B* as platform for intelligent sensors [Ras]. The Raspberry Pis 3B supports several interfaces to read multiple elementary sensor signals. This implementation reads sensor signals via the *Serial Peripheral Interface* (SPI). Communication between intelligent sensors uses the Raspberry Pi's Ethernet interface. All communication for the organisation and configuration of the fusion system uses TCP/IP. Process data (sensor signals and features) are communicated via an Industrial Ethernet in real-time. The process data communication is not part of this publication.

The automated fusion system design is structured into the following four phases:

1. *Discovery*: The system manager searches for available intelligent sensors. The discovery phase is carried out continuously independent of the other three phases. If a new intelligent sensor is discovered, the knowledge building phase is triggered.

2. *Knowledge Building*: Semantic information (self-description of intelligent sensors) is transferred to a knowledge base at the system manager.

3. *Orchestration*: The system manager carries out the fusion system configuration automatically.

4. *Operation*: All intelligent sensors periodically send their sensor signals and features to the system manager using a real-time Ethernet protocol.

Discovery of intelligent sensors and transfer of semantic information is implemented using the *Open Platform Communication Unified Architecture* (OPC UA). OPC UA offers a *Local Discovery Server* (LDS), which exposes available OPC UA servers in a local network. As soon as the system manager has discovered an intelligent sensor, the semantic self-description is collected and stored in the system manager's knowledge base. Then, the fusion system is orchestrated using a rule-based system. The orchestration engine identifies based on available sensors and algorithms features and different kinds of attributes (physical, module, functional,

quality). For details about the orchestration process and the rule-based system the reader is referred to the corresponding journal article [FMH+17]. The last step in the orchestration phase is the creation of an configuration file for the real-time communication. This configuration file is used to determine the layout of the real-time Ethernet communication network.

The remaining parts of this publication are structured as follows. Chapter 2 presents two exemplary application scenarios. The application scenarios are described in detail regarding monitored system, applied intelligent sensors, identified features, and identified attributes. Chapter 3 focuses on the implementation specifics. There, the structure of the provided software, the system- and self-descriptions, and software requirements are detailed. This chapter provides furthermore instructions for starting the orchestration application.

2 Example Scenarios

This chapter describes two application scenarios. In the first scenario an intaglio printing unit is monitored and its condition assessed, whereas in the second scenario a room in a private home with a sleeping infant is observed. It is specified for each scenario which intelligent sensors, elementary sensors, and algorithms are applied. Furthermore, it is shown which features and attributes are identified by the orchestration process. This publication provides the necessary semantic self-description files for the intelligent sensor.

2.1 Intaglio Printing Unit

The automated fusion design system proposed in [FMH+17] is evaluated with respect to an intaglio printing unit demonstrator. Intaglio is the prevalent technique for manufacturing security prints such as banknotes. Main component in an intaglio printing unit is a plate cylinder carrying the printing ink. The printing pattern is etched into the cylinder surface. The cylinder rotates through an ink container absorbing the ink into the etchings. The actual printing takes place by pressing the rotating plate cylinder against the printing substrate. Frequently occurring errors result from the plate cylinder carrying too much ink. This surplus ink is wiped off by a wiping cylinder, pressed against the plate cylinder and rotating in the opposite direction. The wiping cylinder is linearly movable with a motor to adjust the contact force with which it presses against the plate cylinder.

The hierarchy of the modules is shown in Figure 2.1. The overall monitored system is the printing unit. Its sub-modules are the plate and wiping cylinder. The plate cylinder consists of the module motor1 which rotates the cylinder, whereas the wiping cylinder contains the module motor2 for rotating and module motor3 for moving the cylinder linearly. In total seven elementary sensors are applied to monitor characteristics of the demonstrator. The temperature and electric current of motor1 and motor2 are measured by four elementary sensors. The wiping cylinder is equipped with two additional sensors capturing solid-borne sound and contact force. These determine if the wiping cylinder is pressed against the plate cylinder correctly. The last sensor measures the acoustic of the total printing unit. The printing unit demonstrator is equipped with three intelligent sensors. All applied sensors, their measured physical quantities, monitored modules, related intelligent sensors, and input channel are presented in Table 2.1.

The intelligent sensors additionally provide algorithms for feature computation. In this example scenario two algorithms are implemented. Algorithm A_1 is a mean operator, whereas algorithm A_2 is a variance operator. These are detailed in Table 2.2. Algorithms are only applicable to sensor signals with matching physical quantity and dimensionality.

Table 2.3 shows the set of available intelligent sensors and the implemented elementary sensors and algorithms.

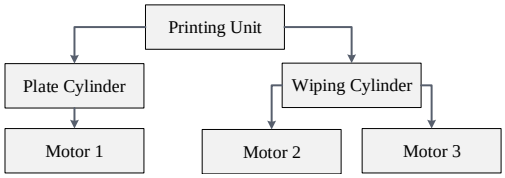


Figure 2.1: Hierarchical representation of the printing unit demonstrator.

Table 2.1: Available sensors and their characteristics.

Solid Sensor	Physical Quantity	Associated Object	Intelligent Sensor	Input
S_1	temperature	motor 1	Intelligent Sensor 3	SPI Channel 0
S_2	temperature	motor 2	Intelligent Sensor 2	SPI Channel 0
S_3	electric current	motor 1	Intelligent Sensor 1	SPI Channel 0
S_4	electric current	motor 2	Intelligent Sensor 1	SPI Channel 2
S_5	solid-borne sound	wiping cylinder	Intelligent Sensor 2	SPI Channel 2
S_6	contact force	wiping cylinder	Intelligent Sensor 1	SPI Channel 4
S_7	acoustic	printing unit	Intelligent Sensor 2	SPI Channel 4

Table 2.2: Available algorithms and their characteristics.

Algorithm	Type	Physical Phenomena
A_1	mean operator	temperature, electric current
A_2	variance operator	acoustic, solid-borne sound, contact force

Table 2.3: Intelligent sensors and their equipment.

Intelligent Sensor	Elementary Sensors	Algorithms
Intelligent Sensor 1	S_1	A_1
Intelligent Sensor 2	S_3, S_4, S_6	$\emptyset$
Intelligent Sensor 3	S_2, S_5, S_7	A_2

Based on the available elementary sensors and algorithms, the orchestration engine infers features. Sensors S_j are assigned to algorithms A_k resulting in a set of features $\mathbf{F}$. Table 2.4 lists the set of inferred features with respect to the previously mentioned system set-up.

Table 2.4: Generated features of the orchestration procedure.

Feature	Sensor	Algorithm	Algorithm Type	Physical Phenomenon	Associated Object
F_1	S_1	A_1	mean operator	temperature	motor 1
F_2	S_2	A_1	mean operator	temperature	motor 2
F_3	S_3	A_1	mean operator	electric current	motor 1
F_4	S_4	A_1	mean operator	electric current	motor 2
F_5	S_5	A_2	variance operator	solid-borne sound	wiping cylinder
F_6	S_6	A_2	variance operator	contact force	wiping cylinder
F_7	S_7	A_2	variance operator	acoustic	printing unit

The automated orchestration results in four attributes, two physical (a_1, a_2) and two module attributes (a_3, a_4). Physical attribute a_1 relates to the measured quantity *temperature*, while a_2 relates to the quantity *electric current* Attribute a_3 evaluates the health of the wiping cylin-

der, whereas a_4 evaluates the plate cylinder. Additionally, functional attribute a_5 is designed manually. The functional attribute is specified in the file "FunctionalAttributes.csv" located in the provided source package at ./Orchestration/Java/SystemManager/System/PrintingUnit/. It assesses the *running smoothness* of the overall printing unit. All attributes are listed in Table 2.2.

Table 2.5: Resulting attributes of the orchestration.

Attribute	Attribute Type	Characteristic	Associated Object	C_{a_i}
a_1	physical	temperature	printing unit	$\{F_1, F_2\}$
a_2	physical	electric current	printing unit	$\{F_3, F_4\}$
a_3	module		wiping cylinder	$\{F_2, F_4, F_5, F_6\}$
a_4	module		plate cylinder	$\{F_1, F_3\}$
a_5	functional	running smoothness	printing unit	$\{F_3, F_4, F_5, F_7\}$

2.2 Room Monitoring

The second example scenario included in this publication describes a condition monitoring system of an infant sleeping in a domestic room. In this scenario temperature, acoustics, and light are measured to determine if everything is fine with the sleeping infant. The monitored system comprises two modules, the *room* itself and a sub-module *window* which describes the area of the room around its window. The condition of the room is captured with two temperature, one acoustic, and one light sensor. The window area is again monitored by two temperature sensors. The sensors are assigned to two intelligent sensors and are read via the SPI interface. Details to all sensor are show in Table 2.6.

Table 2.6: Available sensors and their characteristics.

Solid Sensor	Physical Phenomenon	Associated Object	Intelligent Sensor	Input
S_1	temperature	window	Intelligent Sensor 1	SPI Channel 4
S_2	temperature	window	Intelligent Sensor 1	SPI Channel 6
S_3	light / brightness	room	Intelligent Sensor 1	SPI Channel 0
S_4	acoustic	room	Intelligent Sensor 1	SPI Channel 2
S_5	temperature	room	Intelligent Sensor 2	SPI Channel 4
S_6	temperature	room	Intelligent Sensor 2	SPI Channel 6

Similar to the printing unit scenario, the intelligent sensors in this example provide two algorithms, a mean and a variance algorithm, as listed in Table 2.7.

Table 2.7: Available algorithms and their characteristics.

Algorithm	Type	Physical Phenomena
A_1	mean operator	temperature, electric current
A_2	variance operator	acoustic, solid-borne sound, contact force, light

The assignment of sensors and algorithms to each intelligent sensor is shown in Table 2.8.

The orchestration results in six features and three attributes. Automatically identified attributes are two physical stating the condition regarding temperature in the room and at the

Table 2.8: Intelligent sensors and their equipment.

Intelligent Sensor	Solid Sensors	Algorithms
Intelligent Sensor 1	S_1, S_2, S_3, S_4	A_1, A_2
Intelligent Sensor 2	S_5, S_6	$\emptyset$

window. Additionally, the functional attribute *is sleeping* is defined manually which determines if the infant is still sleeping. The functional attribute relies on the acoustic sensor (infant is crying) and the light sensor (infant managed to switch the light on). Identified features are listed in Table 2.9 and identified attribute are shown in Table 2.9.

Table 2.9: Generated features of the orchestration procedure.

Feature	Sensor	Algorithm	Algorithm Type	Physical Phenomenon	Associated Object
F_1	S_1	A_1	mean operator	temperature	window
F_2	S_2	A_1	mean operator	temperature	window
F_3	S_3	A_2	variance operator	light	room
F_4	S_4	A_2	variance operator	acoustic	room
F_5	S_5	A_1	mean operator	temperature	room
F_6	S_6	A_1	mean operator	temperature	room

Table 2.10: Resulting attributes of the orchestration.

Attribute	Attribute Type	Characteristic	Associated Object	$\mathbf{C}_{a_i}$
a_1	physical	temperature	window	$\{F_1, F_2\}$
a_2	physical	temperature	room	$\{F_5, F_6\}$
a_3	functional	is sleeping	system	$\{F_3, F_4\}$

3 Implementation

This chapter outlines the structure of the implemented software for intelligent sensors and the system manager. Furthermore, all system- and self-description files are discussed. It is described where changes need to be made to the description files in order to add or alter sensors and algorithms. Following this, the chapter outlines software requirements and necessary installation steps. Finally, instruction on how to start the orchestration application are given.

3.1 Intelligent Sensors

Intelligent sensors are implemented as OPC UA servers. The source code is provided at the directory ./Orchestration/Java/IntelligentSensor/src/. The source coded is structured into six packages. The *fileManager* package implements a function for reading the intelligent sensor's self-description. The fileManager uses a Java implementation of SensorML. The *fusion.manager* package holds Java classes that are required to carry out the information fusion. Intelligent sensors, solid sensors, algorithms, and features are implemented as separate objects. Each object holds semantic information that is read from the self-description. The *fusion.manager.aggregation* package implements a unimodal potential function and fuzzy membership function necessary to fuse features into attributes. The *gpioreader* package includes several interfaces to read signals from solid sensors. The package supports to connect solid sensors to a Raspberry Pi via Universal Asynchronous Receiver Transmitter (UART), Serial Peripheral Interface Bus (SPI), and Universal Serial Bus (USB). The interface of a specific sensor is specified in its self-description. It is automatically initialised after start-up of the intelligent sensor. To access the interfaces the Pi4J [Pi4] library is implemented. The remaining packages refer to OPC UA. The main class *ServerMain.java* of the implementation is available from the *opcua.server* package. The OPC UA part of the application initialises the server and generates OPC UA nodes for the exchange of semantic information of the intelligent sensor. The initialisation is carried out in the class *MyNodeManager.java*. All *opcua.types* packages are auto-generated from the OPC UA information model.

3.2 System Manager

The system manager is implemented as an OPC UA client. The source code is provided at the directory ./Orchestration/Java/SystemManager/src/. The source coded is structured into four packages. The *fusion.manager* package holds Java classes that represent objects for the fusion system (sensors, algorithms, features, and attributes). Note that the implementation is not equal to the package of the server. Attribute fusion is carried out only at the system manager. Thus, the *fusion.manager.aggregation* package implements the *BalTLCS* and *IIWOWA*

operator (cf. [Mön17] for details about these operators). The overall application is a knowledge based system. The knowledge base is implemented in the *fusion.orchestration* package. All information required for self-configuration is stored in this knowledge base. It serves as basis for the orchestration procedure. Similarly to the intelligent sensor implementation, the remaining packages refer to OPC UA and include the main class of the project. The *ClientMain* class implements both the routine for detection of intelligent sensors and the transfer of semantic information. The remaining *opcua.types* packages are auto-generated following from the OPC UA information model.

The system manager includes the specified functional attributes in the file “FunctionalAttributes.csv” located at ./Orchestration/Java/SystemManager/System/"Scenario". It is read automatically by the system manager application.

The system manager can either be installed on a separate device (Raspberry Pi or windows machine), but can also run in parallel to an intelligent sensor application, i. e., a single device (Raspberry Pi) can serve both as an intelligent sensor and as the system manager.

3.3 System- and Self-Description

The orchestration of an information fusion system is based on semantic descriptions of the monitored system, available sensors and algorithms. The descriptions are written in *SensorML*, which is a XML-based description language specifically designed for use in the sensor domain. A complete introduction to SensorML and its implementation can be found in [BoR14]. SensorML files are used to describe four types of objects: intelligent sensors, solid sensors, algorithms, and modules. Each object (also referred to as component) is specified in a separate SensorML file. Files describing the intelligent sensor, solid sensors, and algorithms are stored at an intelligent sensor. The files are located, e. g., for the printing unit scenario at ./Orchestration/Java/IntelligentSensor/System/PrintingUnit/ . Files for all modules of the monitored system are stored at the system manager. These are located for the printing unit scenario at ./Orchestration/Java/SystemManager/System/PrintingUnit/Modules/ . Each SensorML file is structured into different sections (e. g., *header* or *system description*). In the following the most important sections for each component type (intelligent sensor, module, etc.) are outlined.

All four component types make use of the system description section. It provides general information about the described object. The system description consists of a *description* and a *name* entry, which are human-understandable text strings, and an *identifier* entry. The identifier provides the UID of the component which has to be unique in the fusion system.

An intelligent sensor SensorML file is the basis for the self-description of an intelligent sensor (e. g., file “INTELLIGENT_SENSOR1.xml” for the printing unit scenario provided at directory ./Orchestration/Java/SystemManager/System/PrintingUnit/). It includes and links to separate solid sensor and algorithm descriptions in the *components* section. A SensorML component specifies a name (e. g., *current1*), a title (e. g., *inIT:sensors:current1*) and a role (*sensor* or *algorithm*). It provides additionally a reference to the SensorML file of the component.

The description of a solid sensor states for example its manufacturer, physical characteristics, signal characteristics and monitored module. Most important for the orchestration process are the monitored module, the captured physical quantity and dimensionality of the signal. The

monitored module is specified using the *featuresOfInterest* section. It provides a textual title of the module and a link to the associated SensorML file. For example, the *temperature sensor 2* from the printing unit scenario has the feature of interest *inIT:modules:motor2* whose explicit description is available at the local directory specified by the *href* attribute. The captured physical quantity is stated in two sections, first in the classification section at the term *PhysicalQuantity* and second in the *outputs* section at the field named *type*. To change the captured physical quantity of a solid sensor, both entries need to be altered. The dimensionality of the sensor signal is also stated in the outputs section. There, it is to be found at the field named *dimensionality*.

An algorithm description file states the allowed characteristics of input signals and the characteristics of the extracted feature. Allowed input signals are specified in the *inputs* section comprising dimensionality and a list of physical quantities. The output feature is defined in the *outputs* section stating the dimensionality and the type of the feature (e. g., mean or variance). For instance, the mean algorithm allows one-dimensional input signals capturing the temperature or electric current of the observed object. It applies a mean operator and outputs a one-dimensional feature.

A module file includes besides the system description a section stating child and parent modules with regard to the hierarchy of the observed system. For example , motor2 in the printing unit scenario has the parent *inIT:modules:Wiping Cylinder*. The child and parent identifiers are used by the orchestration engine to build the hierarchy of the monitored system.

3.4 Software Requirements

This section details necessary software installations for the involved computer systems. The following assumes that the system manager and intelligent sensors run on Raspberry Pis. The OPC UA Local Discovery Server needs to run on an external windows machine.

3.4.1 Development Environment and Source Code

The code for the implementation was written using the *Eclipse IDE for Java Developers* in version *Neon.2 Release (4.6.2)*[1]. The implemented program depends on libraries and code from the *Prosys OPC UA Java SDK* [2] toolkit. The Prosys software is licensed proprietarily, and is hence not provided in this publication. In order to compile and run the fusion design software, it is necessary to acquire the OPC UA Java SDK software from Prosys directly. We used an evaluation license in version 2.1.0-436. As soon as the Prosys software is available, the following files need to be copied to their intended directories:

- *Library jar-file*: The library file *Prosys-OPC-UA-Java-SDK-Client-Server-*.jar* needs to be copied to ./Orchestration/Java/SystemManager/lib/ and ./Orchestration/Java/IntelligentSensor/lib/ .
- *codegen folder*: The content of the codegen folder (included in the Prosys software) needs to be copied to ./Orchestration/Java/SystemManager/codegen/ and ./Orchestra-

[1] Eclipse website (2017-03-31) https://eclipse.org/
[2] ProSys OPC Ua Java SDK (2017-03-31) https://www.prosysopc.com/products/opc-ua-java-sdk/

tion/Java/IntelligentSensor/codegen/ . Please note, the *codegen.properties* file must not be replaced in the copying process.

– *demo files*: The Prosys software is delivered with a demo application in the folder named *samples*. The samples folder includes a *com* folder. The com folder needs to be copied to ./Orchestration/Java/IntelligentSensor/src/ .

3.4.2 Installations on Raspberry Pi

Before a Raspberry Pi can be used as an intelligent sensor or as the system manager, several libraries and toolkits have to be installed first. The following list shows all necessary installations and configurations:

– *Java JDK*: The implementation of an automated fusion design requires a *Java Development Toolkit* (JDK) version 8 installed, preferable the JDK distributed by *Oracle*. New stock Raspberry Pis running Raspbian or NOOBS (operating system specifically designed for the Raspberry Pi) as of release date 2017-03-03 have an Oracle JDK8 preinstalled. However, it can be manually installed using:

```
sudo apt-get install oracle-java8-jdk
```

The Java JDK needs to be installed on intelligent sensors and the system manager.

– *Pi4J*: Reading analogue signals via the SPI interface requires additional software. We use the *Pi4J* library to access the SPI interface and to read analogue signals [Pi4]. The Pi4J library needs to be installed on each Raspberry Pi prior to executing the orchestration software. An installation guide is available from the Pi4J project-webpage at http://pi4j.com/install.html. The Pi4J library needs only to be installed on devices serving as intelligent sensor. By default SPI is disabled on Raspberry Pis. To enable SPI, access the Raspberry Pi's configuration tool by typing "raspi-config", select "advanced options", then select "A6 SPI", and confirm with "yes".

3.4.3 OPC UA Discovery Server

The OPC UA Local Discovery Server (LDS) needs to be installed on an external windows machine which needs to be available in the local network. The LDS software is distributed freely by the OPC Foundation[3]. In this implementation the version 1.02.335.1 is used. The following steps are necessary for the installation of the LDS.

– Unzip installation file at windows machine.

– Execute msi-file and follow the OPC installation wizard.

– The LDS uses a certificate-based encryption for communication with the intelligent sensors. The certificates are provided at ./Orchestration/OPCUA/Certificates/ . They need to be copied to directory C:\ProgramData\OPCFoundation\UA\Discovery\pki\trusted.

[3] OPC UA LDS (2017-03-23) https://opcfoundation.org/developer-tools/developer-kits-unified-architecture/local-discovery-server-lds

At this point all preliminary installations are completed. The following section describes the required steps to start the orchestration.

3.5 Starting the Orchestration Software

The fusion system design program uses an information model modelled with the Unified Automation UaModeler. This information model is provided as two XML-files (./Orchestration/Java/IntelligentSensor/codegen/intelligentsensor.xml and ./Orchestration/Java/SystemManager/codegen/intelligentsensor.xml). The model needs to be made available for the Java implementation which is done by the codegen software provided by Prosys SDK. It takes the XML representation of the information model, derives Java classes, and adds them automatically to the Java project. The code generation is either executed as an ant build in a development environment (in Eclipse: right click on *build.xml*, *Run As*, *Ant Build*) or as a standalone Java application (by invoking codegen-2.0.0-standalone.jar). The codegen folder and the build.xml file are shown in Figure 3.1. The code generation has to be executed once

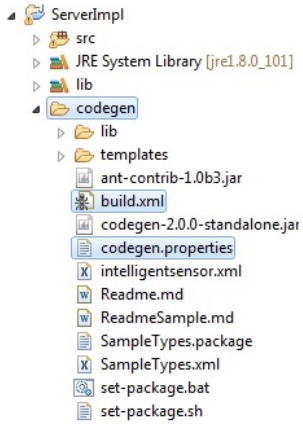


Figure 3.1: Java code generation from external information models.

for the system manager code and once for the intelligent sensor code. If the generation is successful, java-files are generated at ./Orchestration/Java/IntelligentSensor/src/opcua/types and ./Orchestration/Java/SystemManager/src/opcua/types as shown in Figure 3.2. Detailed instructions for this code generation are given in the *Readme.md* file located in the codegen folder.

Now the program code needs to be compiled and exported as a runnable jar file. The export procedure is depicted in Fig. 3.3. A context menu is available after right clicking the project in Eclipse. The menu shows an *Export* functionality, which has to be selected (cf. Fig. 3.3a). An export window opens that includes a directory *Java* where the option *Runnable Jar file* needs to be selected. In the following window, specifics for the export have to be defined (cf. Fig. 3.3b). The project has to be selected and the destination folder has to be defined.

After export, the client implementation (provided at ./Orchestration/Java/SystemManager/

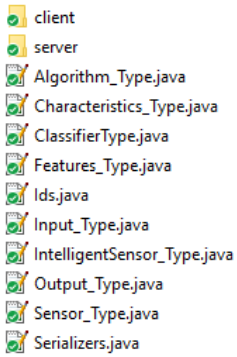
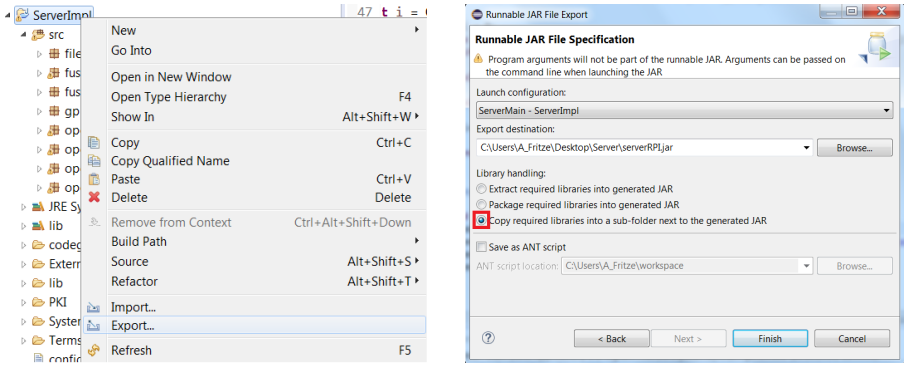


Figure 3.2: Java code generation from external information models.



(a) (b)

Figure 3.3: Runnable jar file export. (a) Project context menu. (b) Export options.

) needs to be copied to the system manager, whereas the server implementation (provided at ./Orchestration/Java/IntelligentSensor/) needs to be copied to each intelligent sensor. The System and Terms folders include semantic information and the vocabulary of the orchestration system. In ExternalJavaFiles the actual implementation of algorithms is available. The PKI folder holds certificates for securing the OPC UA communication.

Additionally, the intelligent sensors need to be equipped with their self-description file. The application scenarios provide the self-description files for all involved intelligent sensors. For example, the printing unit scenario includes three files. The Java implementation of an intelligent sensors expects a file named *INTELLIGENT_SENSOR.xml*. Thus, the provided files need to be renamed at each intelligent sensor. For instance, the file *INTELLIGENT_SENSOR1.xml* needs to be renamed to *INTELLIGENT_SENSOR.xml* at *Intelligent Sensor 1*.

The orchestration implementation is now ready to be executed. The client implementation is carried out at the system manager with the following command:

Listing 3.1: Execution of the system manager application.

```
Usage: sudo java -jar clientRPI.jar

-d IP           Define the DiscoveryServerUrl (IP) to register the application to
-desc 'folder'  Set folder name of system descriptions for the use case scenario.
                The files have to be available from the 'System' folder that has to be
                located in the same directory as the program file. Functional attributes
                are defined in an external csv-file in this directory.
-?; -h; -help   Show this help text

Example:
sudo java -jar clientRPI.jar -d 192.168.1.150 -desc RoomMonitoring
```

The server implementation needs to be started at each intelligent sensor:

Listing 3.2: Execution of the intelligent sensor application.

```
Usage: sudo java -jar serverRPI.jar

-d url          Define the DiscoveryServerUrl (IP) to register the application to
-s              Initialize interface for sensor data access (GPIO, UART, USB (Serial))
-desc 'folder'  Set folder name of system descriptions for the use case scenario.
                The files have to be available from the 'System' folder that has to be
                located in the same directory as the program file.
-?; -h; -help   Show this help text

Example:
sudo java -jar serverRPI.jar -s -d 192.168.1.150 -desc RoomMonitoring
```

The command-line argument *-desc* specifies which application scenario is used, i. e., where to find the self- and system-description files. This publication provides two example scenarios as described in Section 2. The printing unit scenario is selected with *-desc PrintingUnit* and the room monitoring scenario with *-desc RoomMonitoring*. All intelligent sensors now register themselves at the discovery server at the specified IP-address (in this example: *-d 192.168.1.150*). The system manager queries the discovery server periodically for available intelligent sensors. As soon as all intelligent sensors are known to the system manager, it orchestrates the information fusion system. The identified attributes are outputted to the terminal. A successful orchestration for the room monitoring scenario is shown in Figure 3.4.

```
######## Available Attributes ########
# - UID                                  - Attribute Type          - Module                - List of features

1 - inIT:attributes:window:temperature   - physical               - inIT:modules:window   - [inIT:features:mean:temp1,
 inIT:features:mean:temp2]
2 - inIT:attributes:system:temperature   - physical               - inIT:modules:system   - [inIT:features:mean:temp1,
 inIT:features:mean:temp2, inIT:features:mean:temp3, inIT:features:mean:temp4]
3 - inIT:attributes:is sleeping          - functional              - inIT:modules:system   - [inIT:features:variance:ac
oustic, inIT:features:variance:light]
```

Figure 3.4: Output at system manager after completed orchestration for the room monitoring example.

Bibliography

[BoR14] BOTTS, Mike ; ROBIN, Alexander: *OGC SensorML: Model and XML Encoding Standard*. Open Geospatial Consortium, 2014

[FMH+17] FRITZE, Alexander ; MÖNKS, Uwe ; HOLST, Christoph-Alexander ; LOHWEG, Volker: An Approach to Automated Fusion System Design and Adaptation. In: *Sensors* 17 (2017), Nr. 3, 601. http://dx.doi.org/10.3390/s17030601. – DOI 10.3390/s17030601

[FML16] FRITZE, Alexander ; MÖNKS, Uwe ; LOHWEG, Volker: A Support System for Sensor and Information Fusion System Design. In: *3rd International Conference on System-Integrated Intelligence - New Challenges for Product and Production Engineering*, Paderborn, Germany, 2016

[MDL+16] MÖNKS, Uwe ; DÖRKSEN, Helene ; LOHWEG, Volker ; HÜBNER, Michael: Information Fusion of Conflicting Input Data. In: *Sensors (Basel, Switzerland)* 16 (2016), Nr. 11. http://dx.doi.org/10.3390/s16111798. – DOI 10.3390/s16111798. – ISSN 1424–8220

[Mön17] MÖNKS, Uwe: *Information Fusion Under Consideration of Conflicting Input Signals*. 1st ed. 2017. Berlin : Springer Berlin and Springer Vieweg, 2017 (Technologien für die intelligente Automation). – ISBN 9783662537510

[Pi4] PI4J - JAVA I/O LIBRARY FOR THE RASPBERRY PI: *http://pi4j.com/, accessed 2017-03-17*. http://pi4j.com/

[Ras] RASPBERRY PI FOUNDATION: *https://www.raspberrypi.org/, accessed 2016-11-30*. https://www.raspberrypi.org/