Context-Aware Documentation in the Smart Factory

12

Ulrich Beez, Lukas Kaupp, Tilman Deuschel, Bernhard G. Humm, Fabienne Schumann, Jürgen Bock, and Jens Hülsmann

Key Statements

1. In factory environments, it is important to quickly identify appropriate technical documentation for machinery in error and maintenance situations.
2. Smart factory is the vision of a production environment with increasingly self-organising and self-adapting machinery. Identifying appropriate technical documentation in error and maintenance situations will become even more important in the smart factory.

The project ProDok 4.0 – Process-Oriented Technical Documentation for Industry 4.0 – is funded by the German Ministry of Education and Research (BMBF) within the framework of the Services 2010 action plan under funding no. 02K14A110.

U. Beez (✉) · L. Kaupp · T. Deuschel · B. G. Humm
Hochschule Darmstadt, Darmstadt, Germany
e-mail: Ulrich.Beez@h-da.de; lukas.kaupp@h-da.de; tilman.deuschel@h-da.de; bernhard.humm@h-da.de

F. Schumann
dictaJet Ingenieurgesellschaft mbH, Wiesbaden, Germany
e-mail: fabienne.schumann@dictajet.de

J. Bock
KUKA Roboter GmbH, Augsburg, Germany
e-mail: Juergen.Bock@kuka.com

J. Hülsmann
ISRA Surface Vision GmbH, Herten, Germany
e-mail: jhuelsmann@isravision.com

T. Hoppe et al. (eds.), *Semantic Applications*,
https://doi.org/10.1007/978-3-662-55433-3_12

3. In order to identify appropriate documentation, the semantic context of the error or maintenance situation needs to be taken into account. The semantic context needs to be extracted and inferred from low-level machine data.
4. The ProDok 4.0 application allows identifying appropriate documentation in error and maintenance situations for two use cases: robotics application development and maintenance of industrial inspection machines.

12.1 Introduction

In every factory environment, errors and maintenance situations may occur. They must be handled quickly and accurately. Highly experienced and skilled experts for maintenance and repair know exactly what they have to do in most situations. However, the larger and more complex the factory environment and the more specific the error situation, the more likely even experts need to consult technical documentation, not to mention less skilled workers who depend on accurate, easy-to-use technical documentation. But how is it possible quickly and easily identify appropriate technical documentation in an error or maintenance situation?

Technical documentation informs the user of a machine about how to operate it safely and in the intended way, about error situations, maintenance procedures, and proper disposal. Depending on the region, there are different regulations concerning technical documentation. Inside the European Union, technical documentation has to be delivered to the customer in accordance with the Directive 2006/42/EC of the European Parliament and of the Council on machinery [5], such as user manuals, installation and assembly instructions, and maintenance manuals. Chapter 1.7 of ANNEX I – Essential health and safety requirements relating to the design and construction of machinery – deals with information and warnings on machinery, and with information devices connected to the machinery. It is stipulated that “the information needed to control machinery must be provided in a form that is unambiguous and easily understood. It must not be excessive to the extent of overloading the operator. Visual display units or any other interactive means of communication between the operator and the machine must be easily understood and easy to use” [5].

However, first identifying and then searching for the appropriate technical documentation in a given error or maintenance situation is difficult and requires much more than a full-text search. What does the highly skilled and experienced expert do when being faced with an error? First, he will analyse the situation, taking into account the symptoms he observes. Using those observations, the expert will form hypotheses on the error cause based on this experience and will generally know what to do in order to solve the problem.

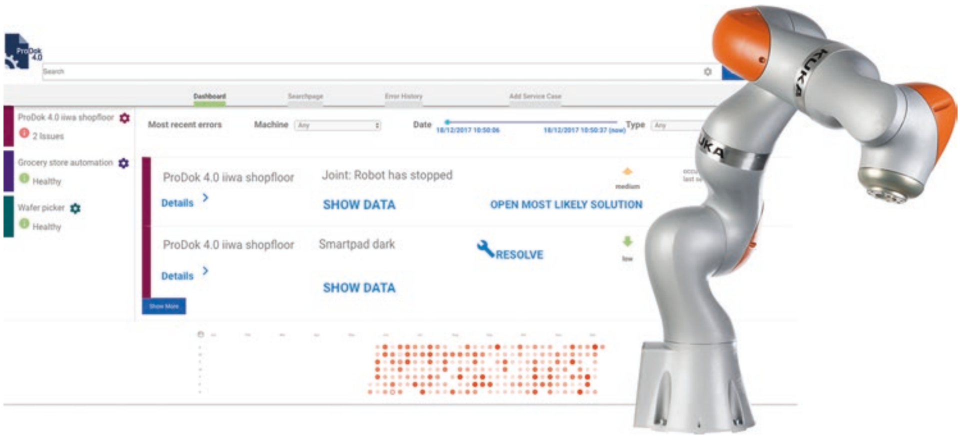
For the less skilled worker, support in performing these steps can be provided through a semantic software application. To achieve this, the application needs to extract low-level machine data in order to infer semantic context information (symptoms), link symptoms to causes and solutions semantically, and present appropriate solutions to the user in an easy-to-use way.

In this chapter, we describe such a semantic application for identifying appropriate technical documentation in a given error or maintenance situation in a factory. We call this application ProDok 4.0. We will demonstrate its use through two use cases: robotic application development and maintenance of industrial inspection machines. Although the use cases are very different, the underlying semantic application has a common software architecture.

12.2 Use Case 1: Robotics Application Development

State-of-the-art robots such as the KUKA LBR iiwa (Fig. 12.1) are designed to be deployed in a wide range of application domains. A key feature to enable this flexibility is the capability to sense forces and torques, and thus, to react to haptic user interaction as well as physical contact or collisions with its environment. The ability to measure external forces and torques allows for the development of force-controlled robot applications, where the teach-in of exact positions is no longer required. Instead, an application could, for example, move the end effector towards a surface until it detects physical contact and then move along the surface while constantly applying a certain force. This allows for the development of sensitive joining or peg-in-hole applications without knowing the exact position of the surface, the hole, the parts to be joined, etc., while achieving as high a precision as could possibly be realised using a position control mode only.

These new features cause an increase in complexity when developing robotic applications. This is because force/torque measurements and the corresponding conditional application control mechanisms need to be mastered in addition to a correct and more



ProDok 4.0 iiwa shopfloor
2 Issues
Grocery store automation
Healthy
Wafer picker
Healthy

Most recent errors Machine Any Date 18/12/2017 10:50:06 18/12/2017 10:50:37 (now) Type Any

ProDok 4.0 iiwa shopfloor Joint: Robot has stopped medium
Details SHOW DATA OPEN MOST LIKELY SOLUTION

ProDok 4.0 iiwa shopfloor Smartpad dark RESOLVE low
Details SHOW DATA

Show More

Fig. 12.1 LBR iiwa and ProDok 4.0 web interface

comprehensive configuration of the robot, end effector, workpiece, etc. Matching both the machine configuration and status as well as the developers knowledge level, our ProDok 4.0 application shall support the process of developing robotic applications by allowing access to helpful documentation for error situations with ease.

12.3 Use Case 2: Maintenance of Industrial Inspection Machines

In the production of glass, defects may occur. A glass inspection machine uses computer vision techniques for detecting such defects and for rating material quality [2]. The glass inspection machine consists of cameras and lights as well as servers with inspection software [12]. In Fig. 12.2, a typical glass inspection machine setup is shown.

An error may occur in every component of the setup, including the interconnections, and on both the software level and the hardware level. Errors in the inspection machine may lead to uninspected glass or glass of unknown quality and thus impact the plant yield. With today's quality requirements for glass products, only quality-inspected glass can be sold to customers.

Common inspection machine errors are camera failures, network errors, lighting issues, or incorrect configuration of external system parameters. The machine itself detects and reports known issues. Typically, the machine on its own cannot solve these issues, e.g. a lost communication signal with another node due to physical damage of the cable.

Less obvious or even previously unknown problems can cause an altered behaviour in the defects detection (over-detection or under-detection). Often, hints for these kinds of problems can be found provided that the distributed information the system generates at runtime is considered as a whole. Our approach aims at detecting errors within the inspection machine based on different indicators from this runtime information. After having identified an error, the system shall give a statement on its cause and, if possible, point to a solution to fix the problem.

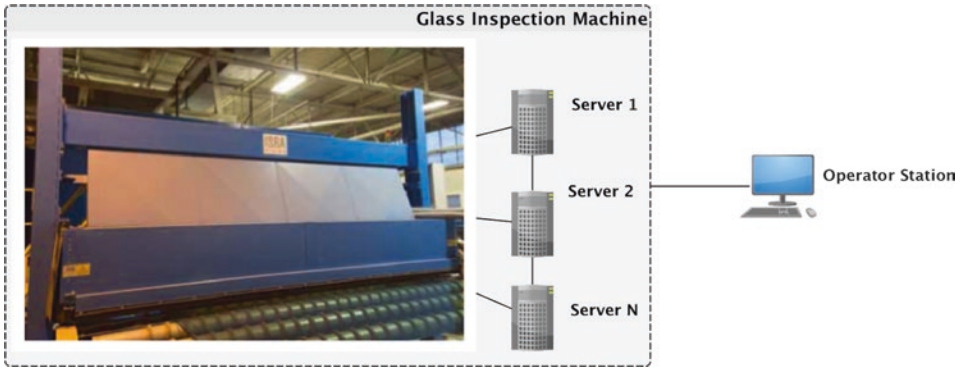


Fig. 12.2 Glass inspection machine setup

12.4 Requirements

Based on expert interviews conducted with personnel of manufacturers for smart factory equipment (robot application developers, inspection machine support engineers), the following requirements have been collected [1]:

R1) In case a machine error occurs in a smart factory, personnel shall be enabled to *resolve the error quickly and with little effort*.
R2) *Appropriate technical documentation* shall be provided, indicating causes and solutions of machine errors.
R3) The provided technical documentation shall *match the machine context* in which the error occurred.
R4) The technical documentation shall be provided *automatically*, alerting the user as soon as the machine error occurs.
R5) *Devices* shall be supported which support the smart factory workflow, e.g., desktop computer, tablet PC, smartphone, or smartwatch.
R6) *Usability* of the user interface shall be high.
R7) *User interaction* shall be *fast* so as not to disturb the personnel's workflow.

12.5 Architecture

12.5.1 Information Architecture

We explain the interaction concept with the robotics example use case previously outlined. When the robot stops during hand guiding, the user is alerted. This alert may be pushed to a suitable device, e.g., the development workstation, a tablet PC, or even a smartwatch. The alert indicates the symptom of the machine error, i.e. 'Joint: the robot has stopped'. See Fig. 12.3 for a screenshot of a dashboard view on the development workstation.

An important aspect of the screen design is the clarity of information presentation, following the ISO Standards 9241-110 [10] and 9241-210 [11]. The user-centred design

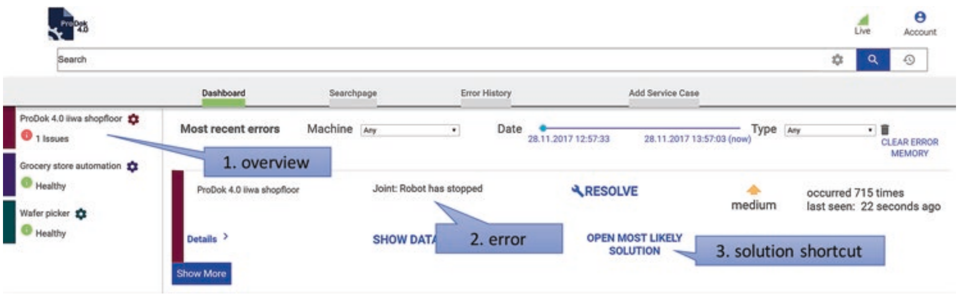


Fig. 12.3 Dashboard view. (Adapted from [1])

approach is applied, which includes close contact with the end user to define requirements collaboratively and test intermediate prototypes iteratively [6].

The dashboard component implements the interaction design pattern of sequence-of-use, which follows the mental model of spatial alignment of semantically related objects [14]. Elements that share a semantic relationship are grouped and the interaction design provides subsequent interaction steps for the main tasks.

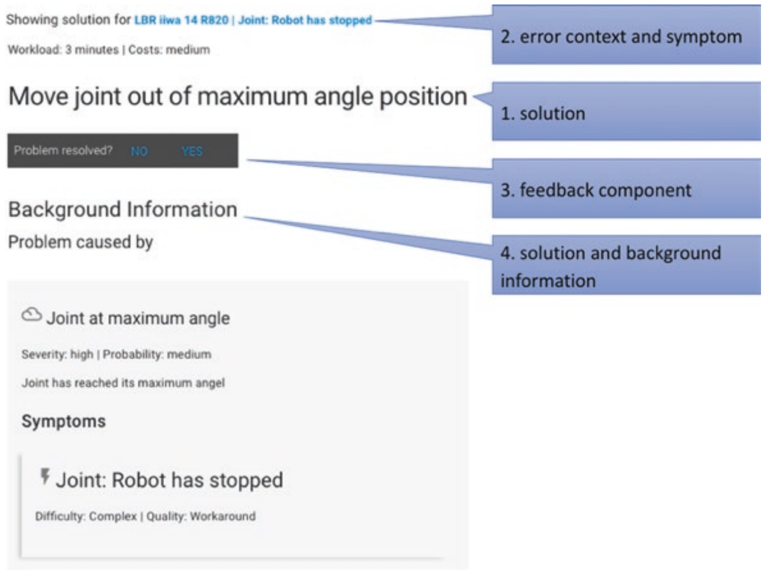
The dashboard's most important components are:

(a) The overview functionality of all connected devices as a list (Fig. 12.3, Mark 1).
(b) A table containing the most recent errors for all connected devices. Each column displays the error symptom alongside a short cut towards the solution, e.g. 'Joint: the robot has stopped' (Fig. 12.3, Mark 2) and a corresponding navigation component reducing the effort a user has to invest for finding a solution ('open most likely solution') (Fig. 12.3, Mark 3).

With a single interaction, i.e., a click on 'open most likely solution' (Fig. 12.3, Mark 3), the user is provided with a solution to the most likely cause of this machine error. See Fig. 12.4.

The conceptual ideas of the Solution View (Fig. 12.4) are as follows:

(a) Present the solution to the machine error, e.g. the solution text 'Move joint out of maximum angle position' (Fig. 12.4, Mark 1).
(b) Provide a quick view to the user regarding error context and symptom, e.g. the blue headline 'LBR iiwa 14 R820 | Joint: the robot has stopped' (Fig. 12.4, Mark 2).


Showing solution for LBR iiwa 14 R820 | Joint: Robot has stopped
Workload: 3 minutes | Costs: medium

Move joint out of maximum angle position

Problem resolved? NO YES

Background Information
Problem caused by

Joint at maximum angle
Severity: high | Probability: medium
Joint has reached its maximum angel

Symptoms

Joint: Robot has stopped
Difficulty: Complex | Quality: Workaround
2. error context and symptom
1. solution
3. feedback component
4. solution and background information

Fig. 12.4 Solution view. (Adapted from [1])

(c) Collect user feedback for a solution, e.g. ‘Problem resolved?’ (Fig. 12.4, Mark 3).
(d) Display solution background information to the user, matching both error context and error, e.g. the cause and symptom (Fig. 12.4, Mark 4).
(e) In case several causes exist (here there is only a single cause identified: ‘Joint at maximum angle’), the less likely solutions are sorted by likelihood descending, and displayed on the solution view following the most likely solution.

12.5.2 Ontology

An *ontology* specifies concepts and their relationships. One purpose of an ontology is to bridge terminology across different domains [3]. In this case, the event data and machine data, both originating from the machine, are bridged to the technical documentation. The ontology can be queried to retrieve *symptoms*, *causes,* and *solutions (SCS)*, matching both product and error data. See Fig. 12.5 for an example following the W3C recommendation for concepts and abstract syntax [17].

Within the ontology, different concepts are modelled:

(a) Hierarchies of products and errors, both interlinked, e.g.:
‘LBR iiwa R820’ ‘isA’ ‘LBR iiwa’
‘LBR iiwa’ ‘has Error’ Joint: maximum angle reached, robot stopped’
(b) Technical documentation split into symptoms, their causes and solutions (SCS) with linkages to (a), e.g.:
‘Joint: maximum angle reached, robot stopped’ ‘hasSymptom’ ‘Joint: Robot has stopped’
‘Joint: Robot has stopped’ ‘hasCause’ ‘Joint at maximum angle’
‘Joint at maximum angle’ ‘hasSolution’ ‘Rotate joint’
all linked to ‘LBR iiwa’ via ‘hasProduct’

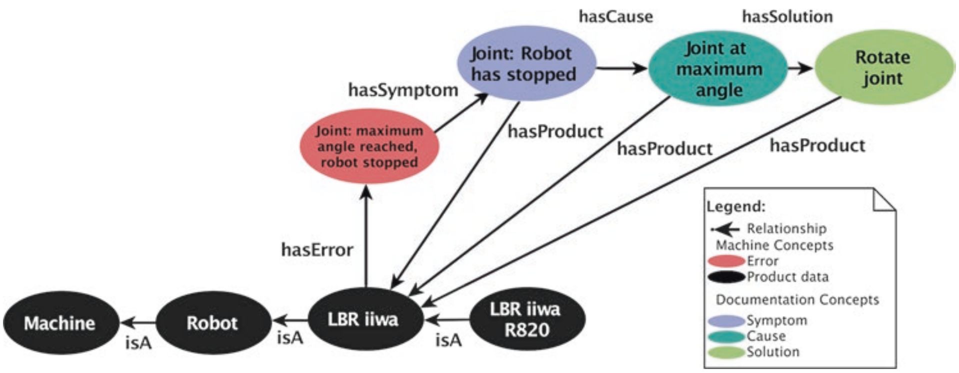


Fig. 12.5 Ontology example [1]

The ontology enables modelling transitive relationships like ‘isA’. ‘LBR iiwa R820’ has no direct relationship with any error, symptom, cause, or solution. However, due to the ‘isA’ relationship with ‘LBR iiwa’, the relationships to the respective error, symptom, cause, and solution can be inferred.

In addition the concepts and relationships shown in Fig. 12.5, may have additional attributes. For example, a solution may have an attribute containing a detailed description on how to apply the solution. An additional attribute for SCS may contain information about the target user group.

12.5.3 Software Architecture

The software architecture is shown below in Fig. 12.6 as an UML class diagram. It consists of three *layers* [16]: *Presentation*, *Logic,* and *Data.* Each layer encompasses different modules. Following Fig. 12.6, the purpose of each module is described.

Presentation Layer: Contains the Graphical User Interface (*GUI*) which serves as an entry point for the user.

Logic Layer: This layer encompasses two modules: (a) *Semantic Knowledge Retrieval* for providing accurately matching documentation and (b) *User Feedback Adapter* for handling user feedback.

Data Layer: Three modules are located here. (a) The *Machine* sends out event and context information, (b) the *Ontology* contains the hierarchies of products and errors, both interlinked, as well as the modularised technical documentation and (c) The User Feedback Store contains collected user feedback.

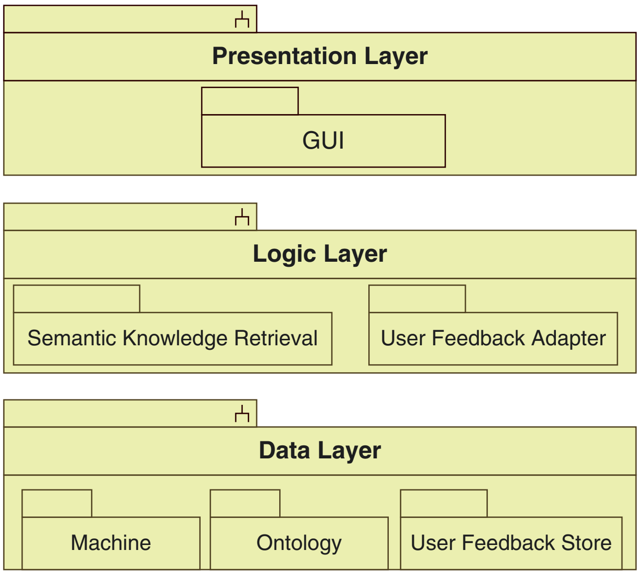


Fig. 12.6 Software architecture [1]

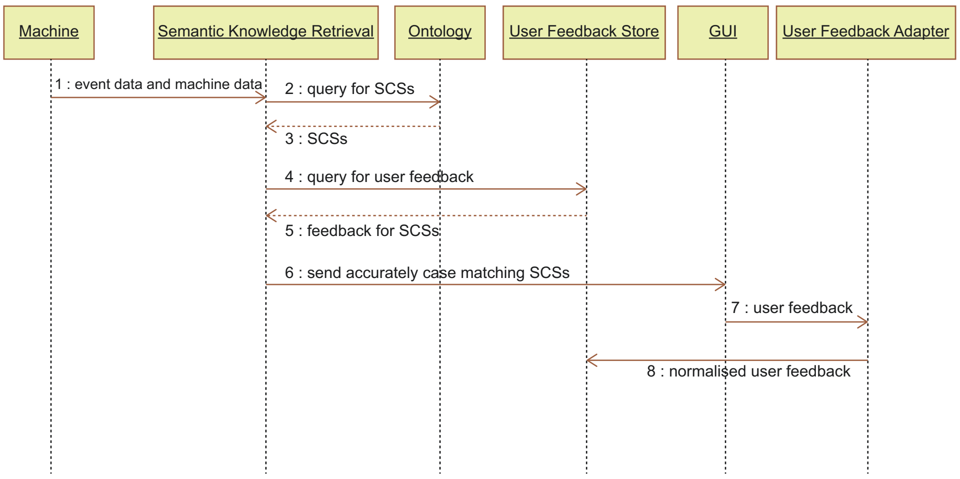

Machine
Semantic Knowledge Retrieval
Ontology
User Feedback Store
GUI
User Feedback Adapter
1 : event data and machine data
2 : query for SCSs
3 : SCSs
4 : query for user feedback
5 : feedback for SCSs
6 : send accurately case matching SCSs
7 : user feedback
8 : normalised user feedback

Fig. 12.7 Communication between components [1]

Figure 12.7 provides an inside view of the communication between components as an UML sequence diagram.

Step 1: When an error occurs, the machine sends *event data* and *machine data* to the *'Semantic Knowledge Retrieval'* component. *Event dat*a describes the machine error in detail, e.g. 'Joint 3 maximum angle reached, robot stopped' on 13:45:17. *Machine data* contains context information, e.g. the robots' digital identification plate with manufacturer and type, e.g.,'KUKA' and 'LBR iiwa 14 R820'.

Steps 2,3: Modularised technical documentation is retrieved from the ontology by querying for the event data in combination with the machine data. The modularised technical documentation consists of the documentation fragments *symptom*, *cause*, and *solution (SCS)*.

Steps 4,5: User feedback is queried for each of the previously retrieved SCSs.

Step 6: Technical documentation accurately matching both the machine event and the personnel's preference is forwarded to the 'Graphical User Interface' (GUI) component.

Steps 7,8: When the GUI sends *user feedback*, it is normalised and saved in the feedback store.

12.6 From Raw Data to Semantic Context

In some situations, machine errors are less obvious than in the example above, where the machine sends an error message like, e.g., 'Joint 3 maximum angle reached, robot stopped'. For example, consider a communication failure between two components of the inspection machine. Such an error can only be detected by observing that regular messages have been missing for an unusual amount of time.

In such situations, raw data needs to be semantically enriched to get a semantic context. We call the process from raw data to semantic context the Semantic Fusion Process (SFP) [13]. Figure 12.8 shows the SFP as a BPMN diagram. In the following, the SFP is explained employing the use case of 'Maintenance of Industrial Inspection Machines' for glass production.

Any internal state changes, exceptions, sensor data, and communication between components (software or hardware components) inside the glass inspection machine is saved into log files. Each line in a log file corresponds to a log event (raw event). Raw events are

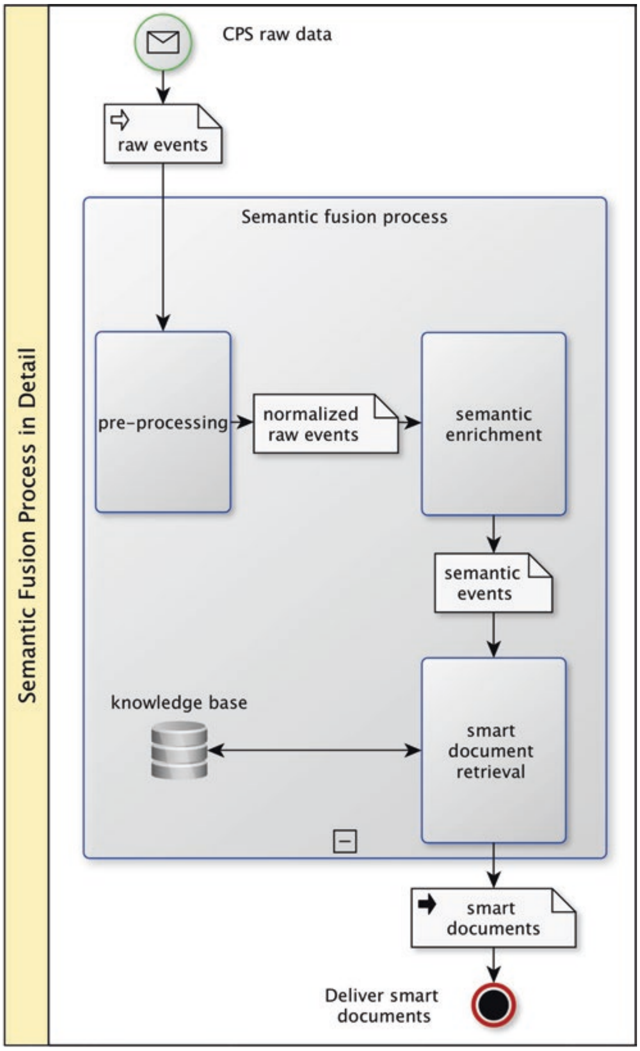


Fig. 12.8 Semantic fusion process. (Adapted from [13])

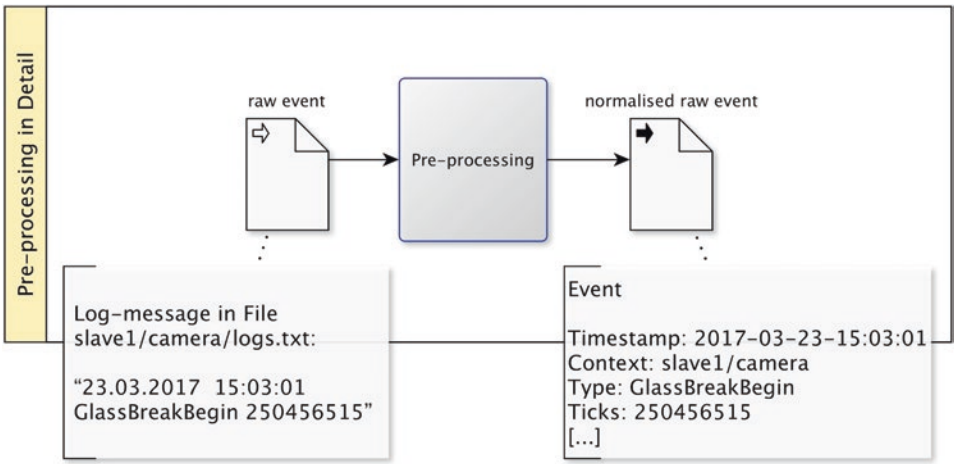
the inbound data of the SFP. Multiple log events are collected within a *data stream* and delivered in real-time into the process.

The *Semantic Fusion Process* consists of three steps. In the *pre-processing* step, differently formatted log events are normalised to a defined schema. In the subsequent *semantic enrichment* step, normalised raw events are refined using analysis methods, generating semantic events. In the *smart document retrieval* step, these semantic events enable a fine-grained query to the knowledge base, thus leading to a composition of smart documents.

12.6.1 Pre-processing

In order to support multiple log events with different encodings and formats from different software and hardware modules, a pre-processing step is needed. Implementing the idea of [15] on the input data, log events are normalised using a defined schema, hence decoupling the SFP from the inbound data format. Figure 12.9 shows the pre-processing step with the example event *GlassBreakBegin*.

GlassBreakBegin indicates the detection of a break within the glass layer. It is reported by the glass inspection machine as a formatted log message as shown in Fig. 12.9. Each event has general attributes such as timestamp, context, and type. The 'timestamp' attribute indicates the time at which the error occurred. The 'context' attribute reflects the origin within the machine, e.g., for a camera event on slave 1 '/slave1/camera'. The 'type' attribute classifies the event, here *GlassBreakBegin*. In addition, an event may have specific event attributes, e.g., ticks (conveyor belt position). The formatted log message of the raw event gets parsed and the extracted data are stored in a normalized raw event object.



```
Log-message in File
slave1/camera/logs.txt:

"23.03.2017  15:03:01
GlassBreakBegin 250456515"
```

```
Event

Timestamp: 2017-03-23-15:03:01
Context: slave1/camera
Type: GlassBreakBegin
Ticks: 250456515
[...]
```

Fig. 12.9 Pre-processing of the event GlassBreakBegin [13]

12.6.2 Semantic Enrichment

Each normalised raw event alone may not be sufficient to identify the machine problem. Consequently, in a second step, the normalised log events get lifted semantically in the semantic enrichment process. Figure 12.10 shows the semantic enrichment process with four subprocesses. Normalised raw events are input data for the semantic enrichment process. The process can apply *filters, pattern matching, value progression analysis, and time progression analysis* to generate semantic events.

In addition, semantic events can be used as input data for semantic enrichment. In the case of an inbound semantic event, higher semantic events can be generated.

Figure 12.11 displays a filtering operation on a normalised event stream, here filtering *GlassBreak* events.

On the left side, the unfiltered data stream is shown containing multiple normalised events. On the right side, only filtered events are shown. For the filter operation, we use pseudocode similar to common complex event processing (CEP) languages as used in CEP tools such as Apache Flink.

Figure 12.12 shows an example for pattern matching operation, identifying corresponding *GlassBreakBegin* and *GlassBreakEnd* events.

The pseudo code specifies a pattern in which a "*GlassBreakBegin*-event-immediately followed-by-an-*GlassBreakEnd*-event" is detected in the data stream. Each pattern recognised can be used to generate a semantic *GlassBreakDetected* event.

Figure 12.13 shows an example of a value progression analysis for generating the semantic *SpeedChanged* event. The speed of the inspection process may vary, due to the versatility of the glass production process. A speed change may affect defect detection and, therefore, is important semantic information.

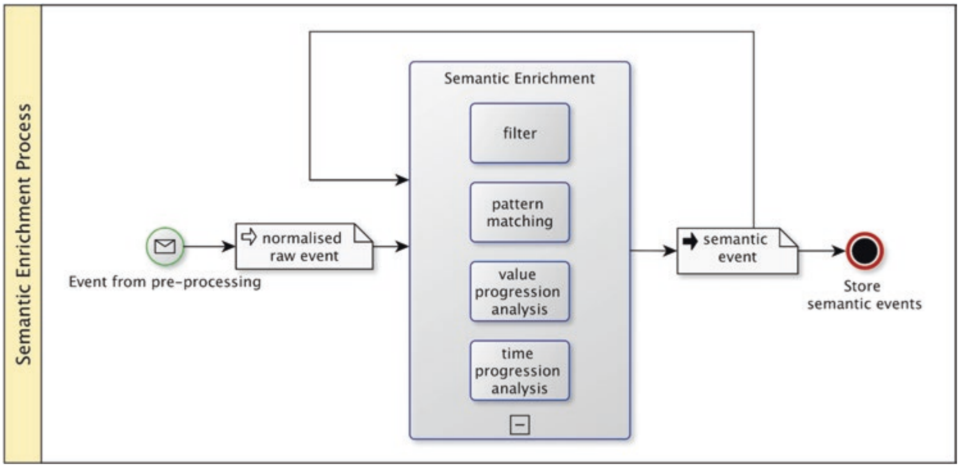

Semantic Enrichment Process
Semantic Enrichment
filter
pattern matching
value progression analysis
time progression analysis
normalised raw event
Event from pre-processing
semantic event
Store semantic events

Fig. 12.10 Semantic enrichment process [13]

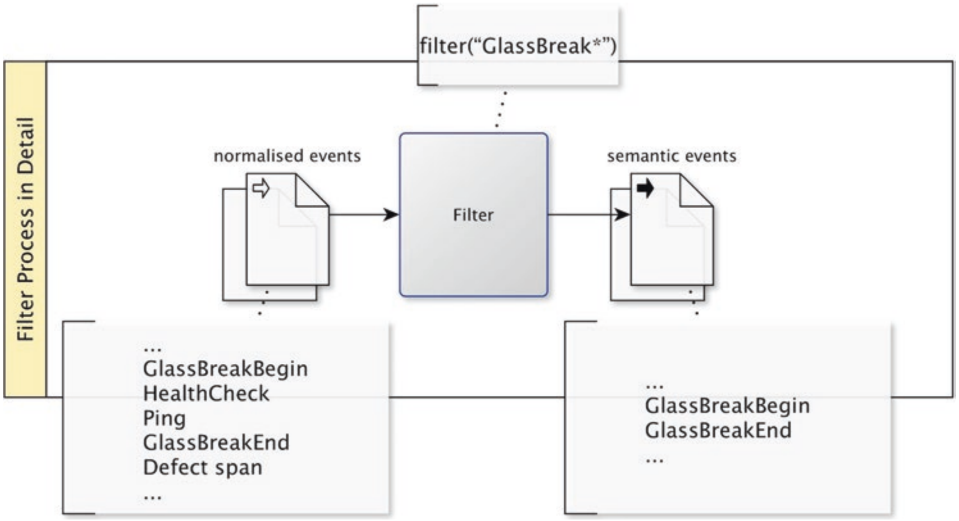


Fig. 12.11 Example of a filter process [13]

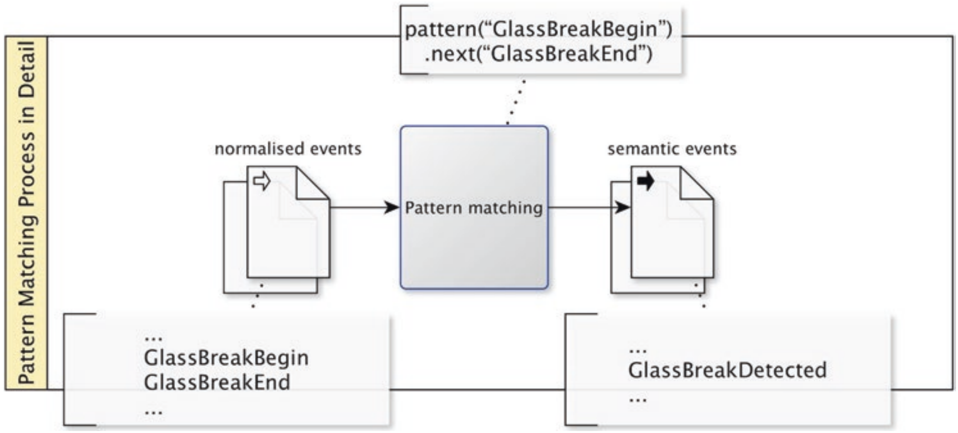


Fig. 12.12 Example of a pattern matching process [13]

Each *SpeedCheck* event delivers a snapshot of the speed of the conveyor belt. The pseudo code for implementing the value progression analysis uses sliding time windows of size 5000s with an overlap of 10s. For each time window, it is checked whether the difference in speed exceeds a threshold (here 0.5s). In this case, a new semantic event *SpeedChanged* is generated.

Figure 12.14 illustrates the time progression analysis with the example of the *SignalLost* event. The *SignalLost* event indicates a connectivity failure, e.g., between a camera and a server.

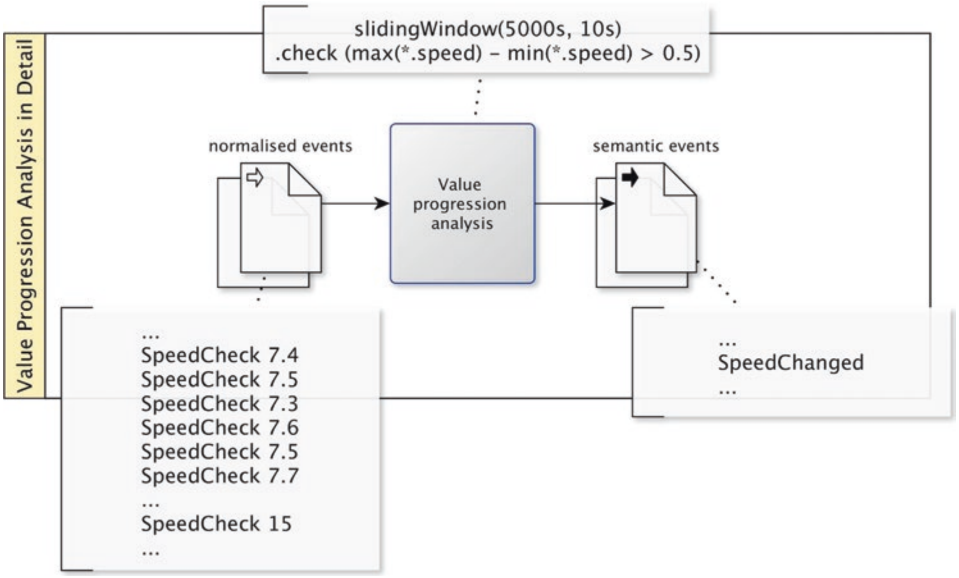


Fig. 12.13 Example of a value progression analysis process [13]

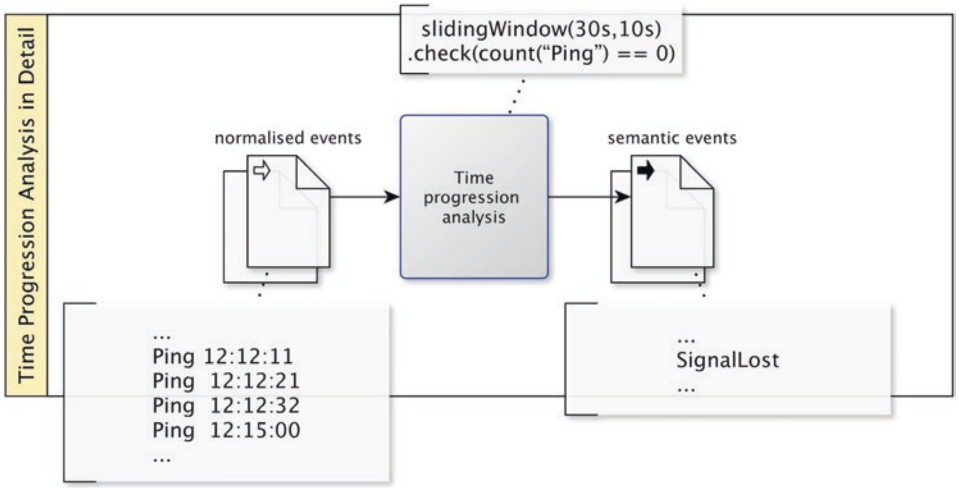


Fig. 12.14 Example of a time progression analysis [13]

Each component within the glass inspection machine regularly sends ping events. If no ping event occurs within half a minute, it is considered as a loss of signal. The pseudocode shown in Fig. 12.14 uses a sliding time window of 30s with an overlap of 10s. If no ping event occurred within the time window, then a *SignalLost* event is generated.

As shown in Fig. 12.10, semantic events generated by semantic enrichment processes can be used as input for other semantic enrichment processes. So, a chain of successive enrichment processes can be established. For example, high conveyor belt speed typically implies a thinner glass layer. With the glass thickness, the properties of the defects change, and in the transition between different thicknesses many defects may occur. The semantic event *SpeedChanged* introduced above may then be used for generating a semantic *ThicknessChanged* event.

12.7 From Semantic Context to Appropriate Documentation

Technical documentation is stored in a *knowledge base*. Technical instructions are provided in the form of *smart documents*. A smart document is modularised, containing the building blocks symptom, cause, and solution. This structure forms the schema of the knowledge base. We call it the *symptom / cause /solution model (SCS model)*.

A *symptom* is a misbehaviour of any form as visual, physical or nonphysical (software-related) aspect [7]. A cause can be the origin of a symptom, but one cause can be linked to multiple symptoms and symptoms can have multiple causes. In addition, a solution covers one or more causes and a cause can be fixed by multiple solutions. On top of this, each solution can have a *semantic context* defining the scope of the solution, e.g., *server* or *camera* (see Fig. 12.15).

Through this modularised structure, it is possible to assemble a smart document consisting of a symptom, a cause, and solutions according to the semantic context for a given semantic event. The smart document is the final output of the SFP, semantically interlinking semantic events and documentation for a machine-specific problem.

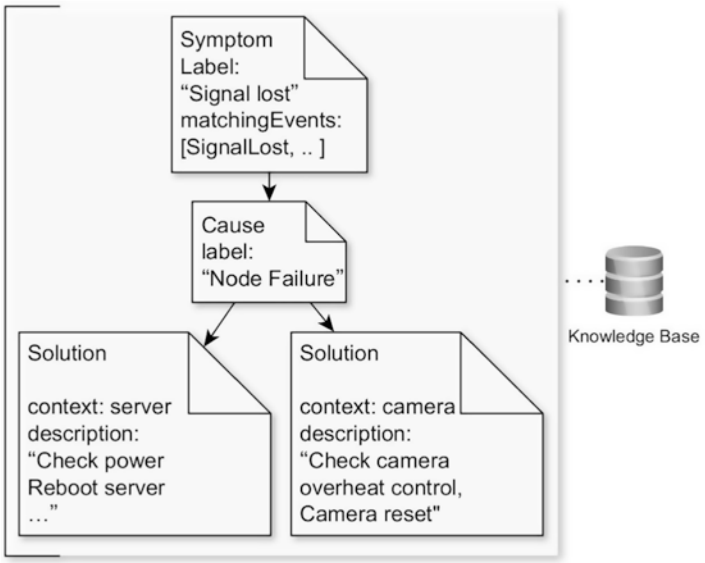


Fig. 12.15 SCS model with example data [13]

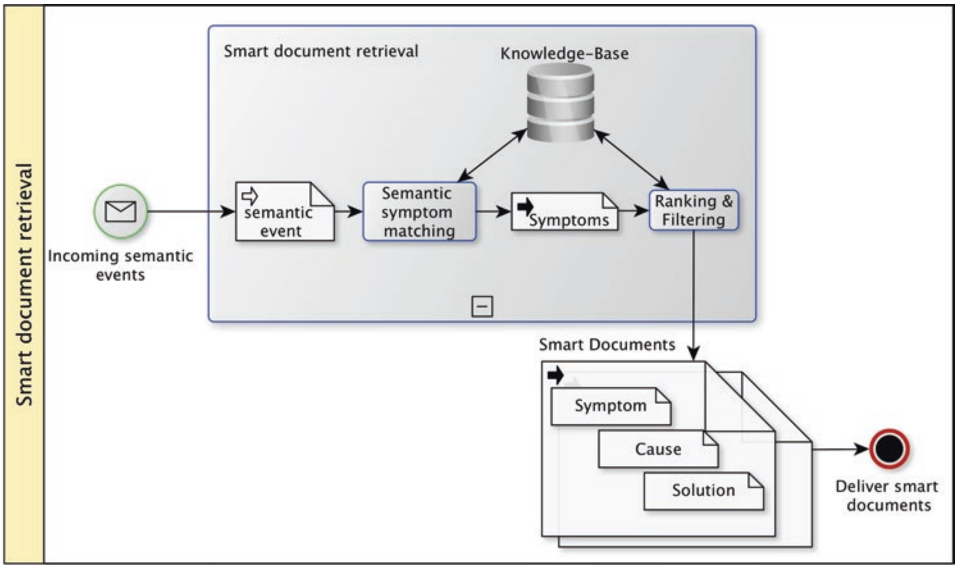


Fig. 12.16 Smart document retrieval process [13]

For instance, the previously generated semantic event *SignalLost* can be mapped onto the symptom ‘signal lost’, and the ‘signal lost’ symptom within the knowledge base has a ‘node failure’ cause with two different solutions.

In Fig. 12.16, the smart document retrieval process is shown, consisting of a ‘semantic symptom matching*’* process and a ‘ranking and filtering*’* process.

The ‘semantic symptom matching’ process queries the knowledge base, e.g., querying on a ‘matchingEvents’ attribute to determine the right symptom. Multiple symptoms can have the same ‘matchingEvent’ like *SignalLost.* In order to provide suitable smart documents, the symptoms get filtered by semantic context. A ranking can be applied according to filtered causes and solutions. For example, the frequency of occurrence of a problem-cause-solution triplet in the past may be used as a ranking criterion.

12.8 Recommendations

We have successfully implemented an application for providing context-aware documentation in the smart factory. We summarize our main learnings from implementing and applying it to both use cases with the following recommendations:

1. Normalise raw data from machinery using a common data format and a (simple) ontology.
2. Queueing technology such as Apache Kafka is mature and scalable and well-suited for communicating events from machinery.

3. Complex Event Processing (CEP) technology such as Apache Flink is mature and scalable and well-suited for semantically enriching raw data from machinery. Functionality provided includes filtering, pattern matching, machine learning, value progression analysis, and time progression analysis.
4. Modular documentation is required when automatically matching technical documentation to machinery data. We recommend separating symptoms, causes, and solutions. The ontology links the machinery with their events and the events with their technical documentation.
5. Check for existing ontologies before developing a custom ontology. This includes product hierarchies and domain vocabularies, e.g. the eCl@ss product and service classification [4], IEEE Standard Ontologies for Robotics and Automation [8], IEEE Suggested Upper Merged Ontology (SUMO) [9], etc.
6. Emphasize the usability and user experience of the semantic application in order to increase acceptance by users.

12.9 Conclusion and Outlook

Documentation in the smart factory is a hot topic. In the project ProDok 4.0, we have regularly invited an open industry board to discuss the applicability of our solution in other corporate use cases. The enormous interest in the topic even surprised us. Many corporations, particularly in the manufacturing sector, face the constant challenge of finding appropriate documentation in error and maintenance situations. Our solution of semantically selecting appropriate documentation based on the current machine context has proven highly attractive to the industry board members.

Where is the road leading? Distilled from intense discussions, here are some suggestions for future work:

- *User-specific information delivery:* Context-awareness may also take into account the skill level and role of the user, providing even more appropriate documentation.
- *Documentation at the point of action:* This requires the use of mobile devices, or, more appropriately, augmented reality devices like Google Glass. Alternative user interaction mechanisms like voice control may be necessary in such settings.
- *Predictive maintenance*: Much better than fixing an error is avoiding it altogether. For certain situations, our solution can be extended to predict errors and maintenance situations based on common patterns and inform the user before issues occur.
- *Executable solutions*: the connectivity of machines in the smart factory allows not only to retrieve context for selecting appropriate documentation, but also to execute operations. Therefore, the user could be offered an additional option 'automatically apply solution in certain situations'.

Semantic applications can really make a difference.

References

1. Beez U, Bock J, Deuschel T, Humm BG, Kaupp L, Schumann F (2017) Context-aware documentation in the smart factory. In: Proceedings of the collaborative European research conference (CERC 2017), Karlsruhe
2. Beyerer J, León FP, Frese C (2016) Automatische Sichtprüfung: Grundlagen, Methoden und Praxis der Bildgewinnung und Bildauswertung. Springer, Berlin
3. Busse J, Humm B, Lubbert C, Moelter F, Reibold A, Rewald M, Schluter V, Seiler B, Tegtmeier E, Zeh T (2015) Actually, what does "ontology" mean?: a term coined by philosophy in the light of different scientific disciplines. J Comput Inf Technol 23(1):29. https://doi.org/10.2498/cit.1002508
4. eCl@ss (2017) Introduction to the eCl@ss standard [online]. Available at: https://www.eclass.eu/en/standard/introduction.html. Accessed 21 Sept 2017
5. EPC (2006) Directive 2006/42/EC of the European Parliament and of the Council of 17 May 2006 on machinery, amending Directive 95/16/EC (recast). Off J Eur Union [online]. Available at: http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=uriserv:OJ.L_.2006.157.01.0024.01.ENG&toc=OJ:L:2006:157:TOC. Accessed 21 Sept 2017
6. Garrett JJ (2011) The elements of user experience: user-centered design for the web and beyond, 2nd edn. New Riders, Berkeley
7. Hornung R, Urbanek H, Klodmann J, Osendorfer C, van der Smagt P (2014) Model-free robot anomaly detection. In: 2014 IEEE/RSJ International conference on intelligent robots and systems, pp 3676–3683. https://doi.org/10.1109/IROS.2014.6943078
8. IEEE (2015) 1872-2015 IEEE Standard ontologies for robotics and automation. IEEE Robot Autom Society [Online]. Availabe at https://standards.ieee.org/findstds/standard/1872-2015.html. Accessed 25 Sept 2017
9. IEEE, Adam Pease (2017) Suggested upper merged ontology (SUMO) [Online]. Available at http://www.adampease.org/OP/. Accessed 25 Sept 2017
10. ISO (2006) DIN ISO 9241-110:2006: 'DIN EN ISO 9241-110 Ergonomie der Mensch-System-Interaktion – Teil 110: Grundsätze der Dialoggestaltung (ISO 9241-110:2006)'. Deutsche Fassung EN ISO 9241-110:2006: Perinorm [Online]. Available at http://perinorm-s.redi-bw.de/volltexte/CD21DE04/1464024/1464024.pdf. Accessed 28 June 2013
11. ISO (2010) DIN ISO 9241-210:2010: 'Ergonomie der Mensch-System-Interaktion – Teil 210: Prozess zur Gestaltung gebrauchstauglicher interaktiver Systeme' [Online]. Available at http://perinorm-s.redi-bw.de/volltexte/CD21DE05/1728173/1728173.pdf. Accessed 29 Oct 2014
12. ISRA VISION AG (2015) The NEW Standard In Float Glass Inspection FLOATSCAN-5D Product Line, ISRA VISION AG [online]. Available at: http://www.isravision.com/media/public/prospekte2013/Brochure_Floatscan_5D_Product_Line_2013-05_EN_low.pdf. Accessed 13 June 2017
13. Kaupp L, Beez U, Humm BG, Hülsmann J (2017) From raw data to smart documentation: introducing a semantic fusion process. In: Proceedings of the collaborative European research conference (CERC 2017), Karlsruhe
14. Koffka K (2014) Principles of gestalt psychology. Mimesis Edizioni, Milan
15. Nuñez DL, Borsato M (2017) An Ontology-based model for prognostics and health management of machines. J Ind Inform Integr [online]. Available at: http://www.sciencedirect.com/science/article/pii/S2452414X16300814?via%3Dihub. Accessed 21 Sept 2017
16. Starke G (2015) Effektive Software-Architekturen: Ein praktischer Leitfaden, 7th edn. Hanser, München. https://doi.org/10.3139/9783446444065
17. W3C (2014) RDF: '1.1 Concepts and Abstract Syntax' [Online]. Available at https://www.w3.org/TR/rdf11-concepts/. Accessed 16 June 2017