A probabilistic ontology-based platform for self-learning context-aware healthcare applications

Femke Ongenae[a,*], Maxim Claeys[a], Thomas Dupont[a], Wannes Kerckhove[a], Piet Verhoeve[b], Tom Dhaene[a], Filip De Turck[a]

[a]*Department of Information Technology (INTEC), Ghent University - iMinds, Gaston Crommenlaan 8 bus 201, B-9050 Ghent, Belgium*
[b]*iMinds VZW, Gaston Crommenlaan 8 bus 102, B-9050 Ghent, Belgium*

Abstract

Context-aware platforms consist of dynamic algorithms that take the context information into account to adapt the behavior of the applications. The relevant context information is modeled in a context model. Recently, a trend has emerged towards capturing the context in an ontology, which formally models the concepts within a certain domain, their relations and properties.

Although much research has been done on the subject, the adoption of context-aware services in healthcare is lagging behind what could be expected. The main complaint made by users is that they had to significantly alter workflow patterns to accommodate the system. When new technology is introduced, the behavior of the users changes to adapt to it. Moreover, small differences in user requirements often occur between different environments where the application is deployed. However, it is difficult to foresee these

*Corresponding author: Tel.: +32 9 331 49 38, Fax: +32 9 331 48 99
Email addresses: Femke.Ongenae@intec.UGent.be (Femke Ongenae), Maxim.Claeys@intec.UGent.be (Maxim Claeys), Thomas.Dupont@intec.UGent.be (Thomas Dupont), Wannes.Kerckhove@intec.UGent.be (Wannes Kerckhove), Piet.Verhoeve@iminds.be (Piet Verhoeve), Tom.Dhaene@intec.UGent.be (Tom Dhaene), Filip.DeTurck@intec.UGent.be (Filip De Turck)

changes in workflow patterns and requirements at development time. Consequently, the context-aware applications are not tuned towards the needs of the users and they are required to change their behavior to accommodate the technology instead of the other way around.

To tackle this issue, a self-learning, probabilistic, ontology-based framework is proposed, which allows context-aware applications to adapt their behavior at run-time. It exploits the context information gathered in the ontology to mine for trends and patterns in the behavior of the users. These trends are then prioritized and filtered by associating probabilities, which express their reliability. This new knowledge and their associated probabilities are then integrated into the context model and dynamic algorithms. Finally, the probabilities are in- or decreased, according to context and behavioral information gathered about the usage of the learned information.

A use case is presented to illustrate the applicability of the framework, namely mining the reasons for patients' nurse call light use to automatically launch calls. Detecting Systemic Inflammatory Response Syndrome (SIRS) as a reason for nurse calls is used as a realistic scenario to evaluate the correctness and performance of the proposed framework. It is shown that correct results are achieved when the dataset contains at least 1,000 instances and the amount of noise is lower than 5%. The execution time and memory usage are also negligible for a realistic dataset, i.e., below 100 ms and 10 MB.
Keywords:
Context-aware, Self-learning, Ontology, Probability, eHealth, Nurse call system

1. Introduction

Computerized tools, health monitoring devices and sensors are being actively adopted in modern healthcare settings, especially to support administrative tasks, data management and patient monitoring (Orwat et al., 2008; Colpaert et al., 2009). Today, caregivers are directly faced with these technologies, which increases the complexity of their daily activities (Tentori et al., 2009). The caregiver has to use several devices to manually consult, insert and combine data, even when carrying out a single task. This is very time-consuming. Due to this inadequate integration of the technology, as well as the large amount of data being generated by the devices and the heavy workload of staff members, it is not rare for important events to be missed, e.g., early indications of worsening condition of a patient. To resolve this issue, context-aware techniques are often proposed to automatically exploit the medical information available to improve continuous care and personalize healthcare (Burgelman and Punie, 2006).

Although much research has been done on the subject, the adoption of context-aware services is lagging behind what could be expected. Most of the projects are prototypes and real applications are still difficult to find. Whereas the healthcare industry is quick to exploit the latest medical technology, they are reluctant adopters of modern health information systems (Chin, 2004). Half of all computer-based information systems fail due to user resistance and staff interference (Anderston and Aydin, 1997). The main complaint made against mobile, context-aware systems is that users had to significantly alter workflow patterns to accommodate the system (Jahnke et al., 2004). This is due to inadequate techniques for personalization of the services, a lack of focus on the soft aspects of interaction, e.g., automated and personalized alerts, and the lack of tackling problems such as the need of the users for control (Criel and Claeys, 2008).

The context-aware platforms use dynamic algorithms, which take the context information into account, to adapt the behavior of the applications according to the context and offer personalized services to the users. However, these algorithms are defined at development time. When new technology is introduced, the behavior of the users changes to adapt to it. Moreover, different environments in which the application is deployed, e.g., different nursing units or hospital departments, might have slightly different requirements pertaining to how the context information is taken into account. It is difficult to foresee these changes in behavior and small nuances in workflows at development time. This means that the context model might be incomplete or the algorithms of the applications built on it may no longer apply. As the applications do not adapt to the requirements and workflow patterns of the users, they feel less in control of the technology and have to adapt their behavior to accommodate the technology instead of the other way around.

To tackle this issue, this paper proposes a self-learning framework, which allows the context-aware applications to adapt their behavior at run-time to accommodate the changing requirements of the users. The proposed framework consist of the following techniques. First, an ontology-based context model with accompanying rule-based context-aware algorithms is used to capture the behavior of the user and the context in which it is exhibited. This captured information is then filtered, cleaned and structured so that it can be used as input for data mining techniques. The results of these data mining techniques are then prioritized and filtered by associating probabilities with the obtained results expressing how reliable or accurate they are. These results and their associated probabilities are then integrated into the context model and dynamic algorithms. These probabilities clarify to the stakeholders that this new knowledge has not been confirmed by rigorous evaluation. Finally, the probabilities are adapted, i.e., in- or decreased, according to context and behavioral information gathered about the usage of the learned information.

The remainder of this article is organized as follows. In Section 2 the relevant related work is discussed and our contribution is highlighted. Section 3 presents the architecture of the proposed probabilistic ontology-based framework for self-learning context-aware healthcare applications. Section 4 discusses the generic implementation of the framework, i.e., the classes that can be extended to implement the specific use cases. The implementation of a specific use case, namely mining the reasons for patients' call light use to automatically launch calls, is presented in Section 5. Finally, the main conclusions of this research are highlighted and the future work is discussed in Section 6.

2. Related work

2.1. Context-aware systems

Dey and Abowd (2000) refer to context as "any information that can be used to characterize the situation of entities (i.e., whether a person, place or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves". A system may be labeled as "context-aware" if it can acquire, interpret and use context information to adapt its behavior to the current context in use (Byun and Cheverst, 2004). A number of generic context platforms have been developed to relieve application developers from the aggregation and abstraction of context information and the derivation of high-level contexts (Hong et al., 2009a; Baldauf et al., 2007; Xue and Pung, 2012; Yilmaz and Erdur, 2012). Unorganized, unprocessed raw data can be voluminous, but has no meaning on itself as it has no relationships or context. Information is data that has been given meaning by defining relational connections. The proposed platforms employ several techniques to model this context information, i.e., key-value, markup scheme, graphical, object-oriented, logic-based and ontology-based models (Strang and Linnhoff-Popien, 2004). A notable trend is emerging towards ontology-based context-aware platforms (Gu et al., 2005; Chen, 2004; Santos et al., 2007; Román et al., 2002).

To write the dynamic algorithms, which take the context information captured in the ontology into account to achieve personalized and context-aware applications, two approaches are commonly used, namely rules or machine learning techniques (Tsang and Clarke, 2008). Rules are manually constructed at development time and thus require developers to foresee all possible situations that can occur at runtime and define the appropriate corresponding actions. Rules are difficult to modify, maintain and scale (Prentzas and Hatzilygeroudis, 2007). Machine learning techniques, e.g., Bayesian networks and neural networks, are also trained at development time. Bayesian networks suffer from similar maintenance and scalability problems as the rule-based approach and acquiring accurate probabilities is a tedious job (Russell and Norvig, 2003). Neural Networks require a lot of processing power and have consequently only been sparsely applied in context-aware applications. Their black-box nature also makes it difficult to gain insight into relations between context and actions, increasing the fear of technology and loss of control from the users. Consequently, with each of these approaches, the context-aware system is only able to cope with a fixed set of context changes that were taken into accounted during the design of the system.

As mentioned previously, run-time adaptation of the dynamic algorithms is needed to adapt to changing behavior of the stakeholders and to truly offer personalized services tuned to the work practices of the specific environment where the application is deployed. A couple of context-aware systems exist that try to tackle this problem by mining historical information (Tsang and Clarke, 2008; Baralis et al., 2011; Strobbe et al., 2012a; Hong et al., 2009b). However, most of the research focusses on the development of data mining techniques, which can be used to learn the patterns and requirements, or use a black-box approach. Litte research has been done on the development of a complete framework for self-learning, context-aware applications and on how the learned knowledge should be integrated in an ontology-based platform.

2.2. Context-aware systems in healthcare

The use of context and context-awareness in healthcare is an active research area (Bricon-Souf and Newman, 2007; Varshney, 2009). First, there is a large amount of available information, specific healthcare situations and related tasks, which create a potentential for cognitive overload amongst the caregivers. Second, the patients, healthcare professionals and some equipment are fairly mobile, which requires accurate localization and adaptation of the healthcare services to the environment. Third, the financial and human resources are limited. This implies a need to cut cost while improving the quality of service to an increased number of people. Context-aware and pervasive prototypes have been developed for a number of hospital (Bardram, 2004; Skov and Hoegh, 2006; Mitchell et al., 2000; Stanford, 2003; Munoz et al., 2003) and homecare & residential care (Fishkin et al., 2003; Floerkemeier and Siegemund, 2003; Korhonen et al., 2003; de Toledo et al., 2006; Hu et al., 2010; Mihailidis et al., 2003; Suzuki and Doi, 2001; Jansen and Deklerck, 2006) use cases. Examples of context-aware healthcare systems based on ontologies can also be found in literature (Fook et al., 2006; Zhang et al., 2005; Paganelli and Giuli, 2011; Ongenae et al., 2011d).

2.3. eHealth ontologies

An ontology (Gruber, 1993) is a semantic model that formally describes the concepts in a certain domain, their relationships and attributes. In this way, an ontology encourages re-use and integration. By managing the data about the current context in an ontology, intelligent algorithms that take advantage of this information to optimize and personalize the context-aware applications, can more easily be defined. The Web Ontology Language (OWL) (McGuinness and Harmelen, 2004) is the leading language for encoding these ontologies. Because of the foundation of OWL in Description Logics (DLs) (Baader et al., 2003), which are a family of logics that are decidable fragments of first-order logic, the models and description of data in these models can be formally proved. It can also be used to detect inconsistencies in the model as well as infer new information out of the correlation of this data. This proofing and classification process is referred to as Reasoning. Reasoners are implemented as generic software-modules, independent of the domain-specific problem. Ontologies thus effectively separate the domain knowledge, which can be re-used across different applications, from the application logic, which can be written as rules on top of the ontology.

The definition and use of ontologies in the medical domain is an active research field, as it has been recognized that ontology-based systems can be used to improve the management of complex health systems (Valls et al., 2010). Most of the developed ontologies focus on biomedical research and are mainly employed to clearly define medical terminology (Ongenae et al., 2011b), e.g., Galen Common Reference Model (Rector et al., 2003), the Foundational Model of Anatomy Ontology (FMA) (Rosse and Jr, 2008) or the Gene Ontology (Blake and Harris, 2008). Little work has been done on developing high-level ontologies, which can be used to model context information and knowledge utilized across the various continuous care settings (Ongenae et al., 2011a). However, ontologies have been developed for specific subdomains of continuous care, e.g., ontologies for structuring organization knowledge in homecare assistance (Valls et al., 2010), representing the context of the activity in which the user is engaged (Rodríguez et al., 2011) and modeling chronic disease management in homecare settings (Paganelli and Giuli, 2011).

2.4. Our contribution

In this paper, we propose a self-learning and probabilistic framework to adapt the behavior of ontology-based, context-aware applications to the changing requirements of the users and their workflow patterns. To our knowledge, little previous research has been done on how discovered trends and patterns can be integrated into ontology-based platforms without making the existing model inconsistent. To tackle this issue, we use a probabilistic approach, which conveys the reliability of the learned knowledge to the users and ensures the compatibility with existing knowledge in the context model. Moreover, the existing research on self-learning, context-aware applications concentrates on exploring data mining techniques, which can be used to discover the trends and patterns. Our research focuses on the development of a complete framework to enable self-learning, context-aware healthcare applications.

3. Architecture of the self-learning, context-aware framework

The general architecture of the proposed self-learning, context-aware framework is visualized in Figure 1. The following subsections discuss the different components and modules of this framework in more detail.

3.1. Context-aware platform

The general architecture of a context-aware, ontology-based platform can be split up into five layers. The *Device Layer* includes all the devices and the software on those devices that deliver context information. The modern healthcare settings contains a plethora of computerized medical equipment to convey the condition of a patient, e.g., monitoring equipment, electronic patient records and laboratory results stored in a database, and support the caregivers in their daily activities, e.g., nurse call systems and task management and planning tools.

The *Context Provider Layer* takes care of the acquisition of specific context information, e.g., location or presence information, and translates it to

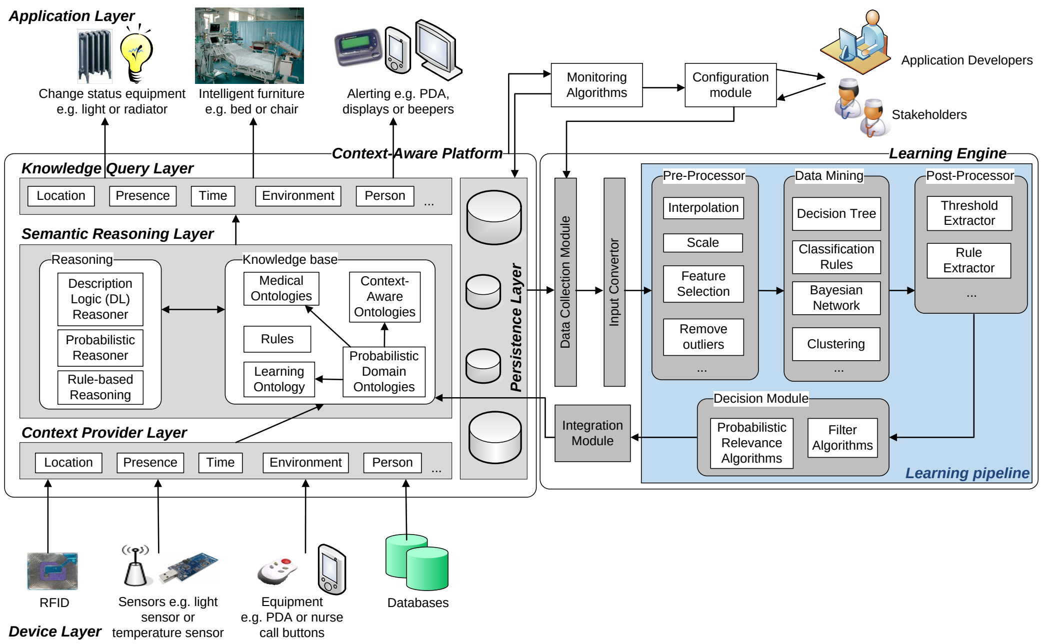


Figure 1: General architecture of the self-learning, context-aware framework

ontology instances. These ontology instances are then added to the *Knowledge Base* in the *Semantic Reasoning Layer*. This *Knowledge Base* aggregates all the relevant context information into a formal context model, i.e., an ontology. Existing *Medical and Context-Aware Ontologies* are integrated into the platform and extended with *Domain Ontologies* which model the information specific to a particular healthcare setting, e.g., the specific roles and competences of the caregivers and how they map on each other, the available monitoring equipment and their threshold values and specific tasks that need to be performed. These *Domain Ontologies* can also contain probabilistic information, e.g., a call made by patient with a heart disease has 25% chance of being urgent.

Reasoning components are then used to derive new, high-level knowledge from the information aggregated in the *Knowledge Base*. Due to the foundation of ontologies in *Description Logics* (*DL*), the models can be formally proofed by using a *DL Reasoner*. This *DL Reasoner* is used to detect inconsistencies in the model as well as infer new information from the correlation of the data. For example, a concept *Fever* is created in the ontology, which automatically detects patients with a temperature above 38 °C. More complex logic is expressed by defining *Rules* on top of this ontology and performing *Rule-based Reasoning*.

The *Knowledge Query Layer* facilitates the retrieval of context information such that it can be used by the different applications and services. The *Application Layer* includes all the devices and the software on those devices that use the (derived) context information to adapt their behavior.

Finally, the *Persistence Layer* ensures the persistence of context information. Static contextual information about users, devices and the environment can be easily obtained from these databases. More importantly, the *Persistence Layer* can also be used to store more dynamic information, such as previous locations of caregivers and patients or actions taken by the users.

The *Semantic Reasoning* and *Persistence Layers* are the most important layers to facilitate a self-learning *Context-Aware Platform*. As the *Knowledge Base* integrates all the context information, it gives insight into the behavior and changing requirements of the users. All the collected context information and the knowledge derived from it is then persisted in the databases from the *Persistence Layer*. This lets the *Learning Engine* exploit this history of context information to derive trends and patterns and adapt the information in the ontology and accompanying rules accordingly.

3.2. Monitoring algorithms and configuration module

Monitoring Algorithms determine missing or inaccurate knowledge in the ontology. An example: situations are logged where a suggestion is given by the system to the staff to do an action, but under certain circumstances the caregivers consistently execute a different action. The *Monitoring Algorithms* constantly monitor the ontology for interesting situations. They gather these situations and store them collectively in the *Persistence Layer*. The results of the *Monitoring Algorithms* can intermediately be shown to *Stakeholders*, i.e., domain experts such as nurses, doctors and professionals working for the healthcare industry, and *Application Developers*. When enough data has been collected, the *Learning Engine* can be initiated. The amount of data that should be gathered depends on the specific use case and the used data mining technique. The input parameters are specified in the *Configuration Module* and the *Data Collection Module* automatically extracts the appropriate data from the *Persistence Layer*. The *Configuration Module* is also responsible for configuring the pipeline. A default pipeline can be used or a specific configuration can be indicated by the *Stakeholders* or *Application Developer*s.

Note, that the *Configuration Module* can be configured both by the *Monitoring Algorithms* themselves and by the *Stakeholders & Application Developers*. It can thus be regulated how much autonomy the *Learning Engine* has. Moreover, the possibility of human intervention avoids unnecessary learning steps in case the new knowledge, which should be added to the ontology based on the observation from the *Monitoring Algorithms*, is straightforward. Finally, the results of the *Monitoring Algorithms* give the *Stakeholders & Application Developers* insight into the behavior and requirements of the users.

3.3. Learning engine

The Pipes-and-Filters architectural design pattern (Bass et al., 2003) was used to design the *Learning Engine*. This data-driven pattern divides a larger processing task into a sequence of smaller, independent processing steps, called filters, that are connected by channels, called pipes. Each filter provides a simple interface, namely it receives messages on the incoming pipe, processes them and provides the results to the outgoing pipe. A filter is thus unaware of its position in the pipeline and which filter precedes and follows it. Because all the filters use similar interfaces they can be combined into different pipelines. Filters can thus easily be added, omitted or rearranged. As a result, the architecture becomes very modular, extensible, re-usable and flexible.

3.3.1. Data collection & input conversion

To be able to use a flexible Pipes-and-Filters architecture, the data exchanged between the filters needs to be expressed in the same format. A format was developed, which allows expressing both the information which is used as input and the knowledge that is obtained as output, e.g., rules. The format is largely based on the Attribute-Relation File Format (ARFF), which is the text file format used by WEKA (Witten et al., 2011).

The Data Collection Module is responsible for gathering the necessary input information for the *Learning Engine* from the *Persistence Layer*. The *Input Convertor* converts this data to the data format used by the *Learning Pipeline*. The *Data Collection Module* and *Input Convertor* cannot be considered as actual filters for two reasons. First, for any use case scenario they will always appear as the first two steps of the pipeline. Second, the input and output format of these modules is dependent on the source from which the information is collected, e.g., a triple store.

3.3.2. Learning pipeline

The *Pre-Processor* contains several modules to clean up the data. For example, the *Remove Outliers* component removes unrealistic entries from the input data, e.g., impossible sensor values. The *Scale* component centers the input values at zero. This is often beneficial for the learning algorithms of various machine learning techniques. *Feature Selection* can be used to reduce the size of the input data set and thus speed up the data mining. Other examples of pre-processing techniques can easily be integrated into the pipeline as new Filters.

The cleaned data is then passed to the *Data Mining* component that provides several techniques to discover trends, e.g., classification rules, decision trees, Bayesian networks or clustering. The results of the *Data Mining* are then processed by the *Post-Processor* to derive the actual information which can be added to the ontology, e.g., rules or thresholds can be derived from a decision tree by the *Rule* or *Threshold Extractor*.

The conclusions of the *Post-Processor* are studied further by the *Decision Module*. To ensure that the *Knowledge Base* does not become inconsistent when the new knowledge is added, i.e., because it contradicts with already defined knowledge, probabilistic relations are defined between the new and existing knowledge. Moreover, this probability also makes clear to the *Stakeholders* that the new knowledge has not been confirmed by rigorous evaluation yet. The *Probabilistic Relevance Algorithms* are used to determine the initial probability that should be associated with this new knowledge. For

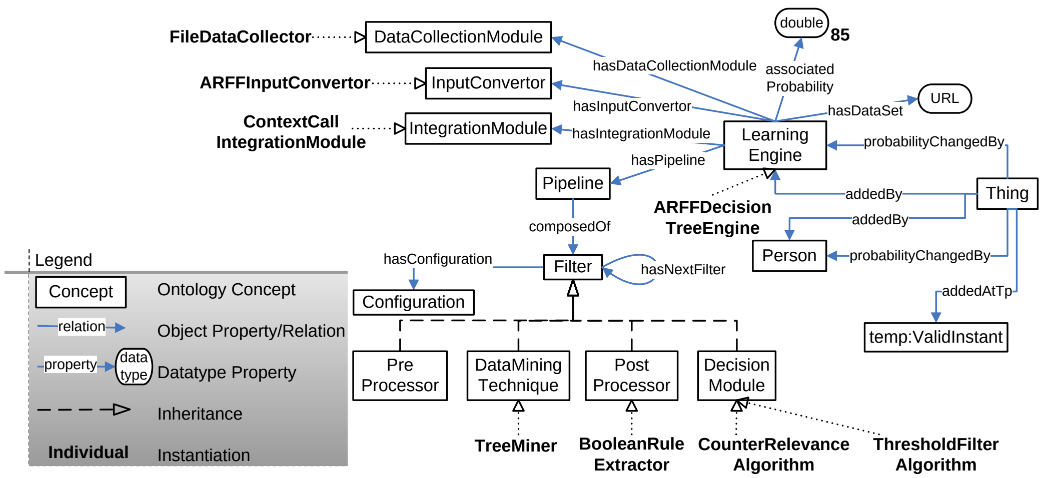


Figure 2: The Learning Ontology

example, it can be calculated how many times a derived rule occured in the data set on which the data mining technique was trained. However, wrong trends can easily be detected because of skewed or too small data sets. It is also important to only include trends that reflect good and general work practices. Wrong information could clutter the *Knowledge Base* and make the context-aware platform less useable. The *Filter Algorithms* are responsible for detecting and removing these anomolies, e.g., by removing knowledge that received a too low probability by the *Probabilistic Relevance Algorithms*.

The *Learning Pipeline* cannot only be used to learn new information, but also to reassess knowledge that has been previously added to the *Knowledge Base*. In this case, the *Probabilistic Relevance Algorithms* are responsible for in- or decreasing the probability depending on the new information that becomes available about the usage of this knowledge.

3.3.3. Integration module & adapting the probabilities

Finally, the *Integration Module* is responsible for defining the probabilistic relations that connect the new knowledge to the existing knowledge in the *Knowledge Base*. For the same reasons as were already explained in Section 3.3.1 for the *Data Integration* and *Input Convertor Modules*, this module cannot really be considered a filter.

For new knowledge, the probability calculated by the *Probabilistic Relevance Algorithms* is used. When the *Stakeholders* are confronted with a probabilistic decision in their daily work practices, they might be interested in the origin of the information, i.e., how the information was learned, before deciding to follow the recommendation of the context-aware platform or not. Therefore, the *Learning Ontology* was created, which allows associating the learned knowledge with its origin. The most important concepts of this ontology are visualized in Figure 2. This ontology also allows *Application Developers* to easily identify learned knowledge. This enables them to treat this knowledge differently if needed, e.g., ignore it in reliability critical applications or highlight it for the users.

For reassessed knowledge, two thresholds are checked. If the probability calculated by the *Probabilistic Relevance Algorithms* falls below the lowest threshold, the knowledge is removed from the *Knowledge Base* as it is clearly not being used or confirmed by the stakeholders. If the probability exceeds the highest threshold, the knowledge is added to the ontology as generally accepted knowledge, i.e., without an associated probability. Finally, if the probability lies between the two thresholds, the probabilitiy of the reassessed knowledge is updated to this probability to reflect its changed reliability. As

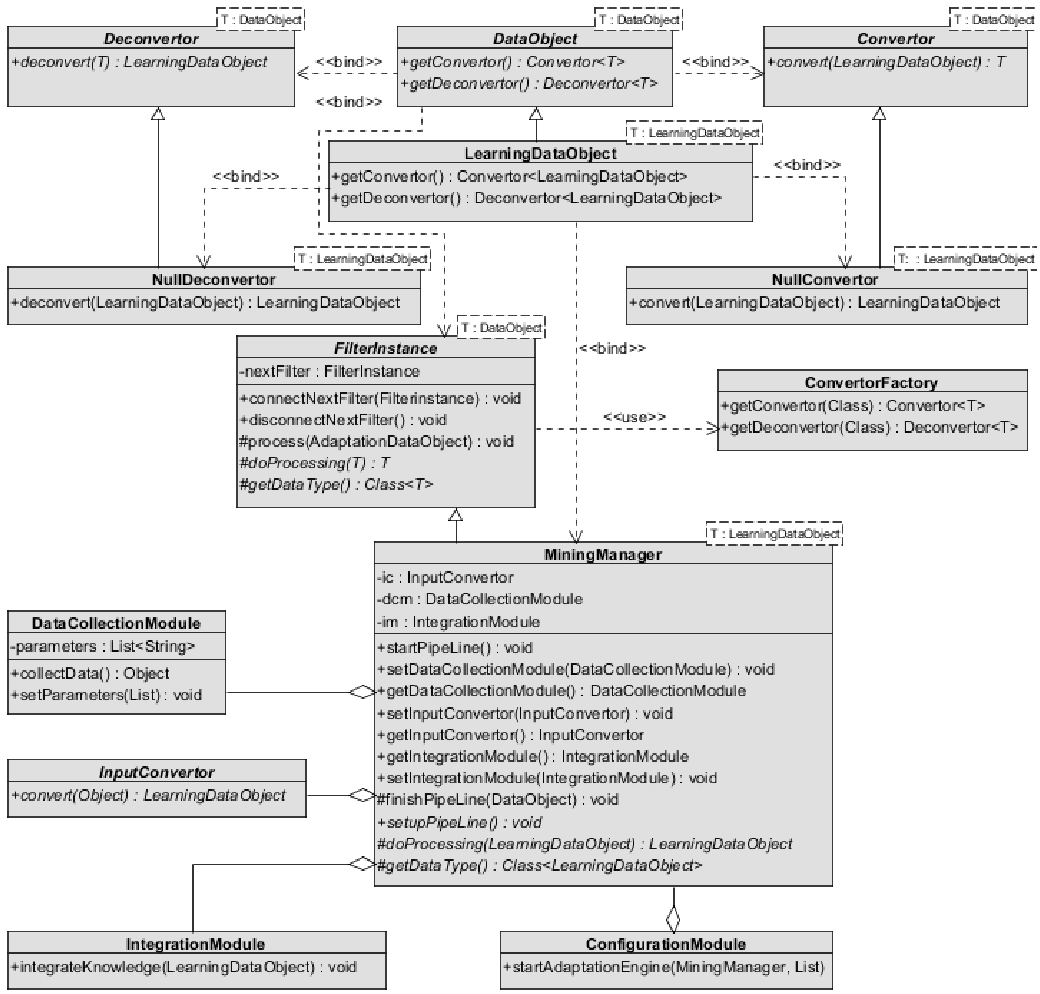


Figure 3: The class diagram of the *Learning Engine* and *Configuration Module*

such, a self-learning, context-aware platform is obtained in which knowledge can be added and removed on the fly based on historical information.

4. Implementation details

The implementation details of the *Context-Aware Platform* are described in Strobbe et al. (2007, 2012b). The platform uses OWL (McGuinness and Harmelen, 2004) as ontology language, Pellet (Sirin et al., 2007) as *DL Reasoner*, Jena Rules (Carroll et al., 2004) and SWRL (Horrocks et al., 2004) to express the *Rules* and SPARQL (Prud'hommeaux and Seaborne, 2008) to query the context information. The platform was extended with the *Probabilistic Reasoner* Pronto (Klinov, 2008) to enable probabilistic reasoning on the ontologies. Jena is used to manage and persist the ontologies.

The *Learning Engine*, *Monitoring Algorithms* and *Configuration Module* were implemented in Java. The class diagram of the *Learning Engine* is visualized in Figure 3. These are the (abstract) classes, which can be used for any scenario. To implement a specific use case, subclasses can be created that implement the specific requirements of the scenario, e.g., a specific pipeline configuration or a specific input convertor. An example of how a specific use case can be implemented is thoroughly explained in Section 5. How these classes can be used to construct and use a specific *Learning Pipeline* with associated *Data Collection Module*, *Input Convertor* and *Integration Module* is visualized with a sequence diagram in Figure 4.

As can be seen, the different filters in the *Learning Pipeline* are respresented by `FilterInstance` objects. Specific filters, e.g., pre- and post-processors, filter algorithms and data mining techniques, are created as subclasses of this `FilterInstance` class by implementing the `doProcessing` method. This method specifies how the data is processed by the specific filter, e.g., a `ScalingFilter` that scales the data or a `ClusterFilter` that

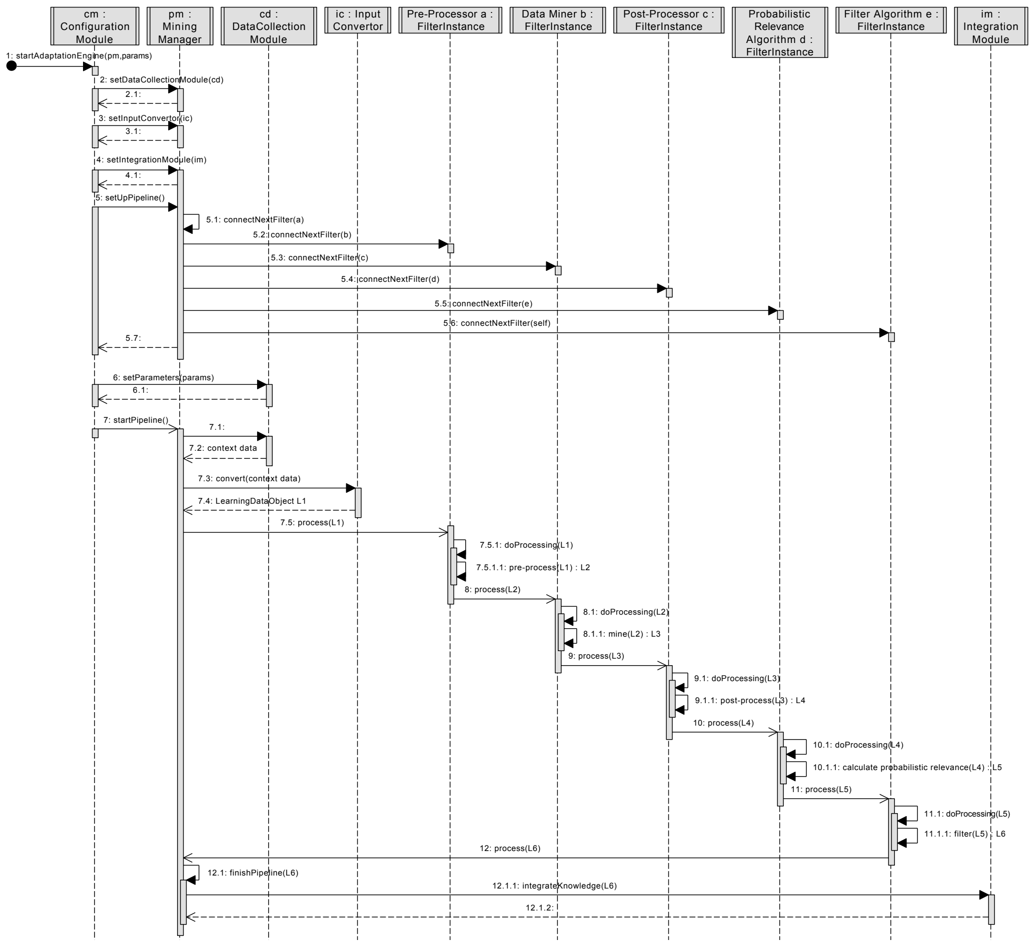


Figure 4: Sequence diagram illustrating the construction and usage of a *Learning Pipeline* with associated *Data Collection Module*, *Input Convertor* and *Integration Module*

clusters it.

As mentioned previously, the data exchanged between the filters in the pipeline uses the same data format, which is represented by the `Learning-DataObject` Java-Object. This object contains the information about the different attributes, i.e., ontology concepts, which will be mined, and their data instances. However, to enable logging of the data at any point during the pipeline, this object can easily be serialized to XML.

As can be seen, (de)convertors can be used to translate the specific data format to other formats. This is not only necessary to convert the context data gathered by the *Context-Aware Platform* to the data format used by the pipeline, but also to allow the usage of external libraries, e.g., WEKA for data mining. The (de)convertors allow to transform the `LearningDataObject` to the format used by the external libraries, e.g., the ARFF format used by WEKA. Each `FilterInstance` indicates which datatype it employs to process the data by using '*generic types*'. Based on the indicated type, the framework is able to automatically find the appropriate `Convertor` and `Deconvertor`. This eases the development of specific use cases and the usage of external libraries. The *Application Developers* only have to develop `Convertor` and `Deconvertor` subclasses that implement the conversion to the specific file format used by the `FilterInstance`.

To manage the complete pipeline, a special type of `FilterInstance` was created, namely the `MiningManager`. This class is responsible for constructing the *Learning Pipeline* out of the separate filters, starting it and processing the results. To implement a specific *Learning Pipeline*, a subclass of the `MiningManager` needs to be constructed that implements the `setupPipeline` method. This method initializes the different filters of the pipeline and connects them to each other. Each `FilterInstance` is connected to the next `FilterInstance` in the pipeline by using the `connectNextFilter` method. The first `FilterInstance` is connected to the `MiningManager`, while the last `FilterInstance` indicates the `MiningManager` as next filter to ensure proper processing of the result of the *Learning Pipeline*.

The `ConfigurationModule` is notified of which data should be collected for the mining process, either by the *Stakeholders* and *Application Developers* or by the *Monitoring Algorithms*. It configures the `MiningManager` to use the appropriate `DataCollectionModule`, `InputConvertor` and `IntegrationModule` that suits this type of data. It also passes the correct parameters to the `DataCollectionModule`, which are needed to retrieve the data from the *Persistency Layer*. Next, the `ConfigurationModule` calls the `setupPipeline` and `startPipeline` methods of the `MiningManager` to create the pipeline and start the learning process. The latter method first collects the necessary data by using the associated `DataCollectionModule` and converts it to the `LearningDataObject` format with the `InputConvertor`. Next, the `MiningManager` calls the `process` method of the first `FilterInstance` in the pipeline. This `FilterInstance` processes the data with its `doProcessing` method and then calls the `process` method of the next `FilterInstance` in the pipeline. This continues until the last `FilterInstance` calls the `process` method of the `MiningManager`. The `MiningManger` then finishes the learning process by calling the `IntegrationModule` to integrate the knowledge in the *Knowledge Base*.

It can be noted that the implemented framework is very extensible, modular and flexible, which allows easy adoption for any use case, as illustrated in the following section.

5. Use case: Mining the reasons for patients' call light use to automatically launch calls

5.1. Scenario description

Nurse call systems are a fundamental technology in continuous care as they are used by caregivers to coordinate work, be alerted of patients' needs, communicate with them through intercoms and request help from other staff members. When patients feel unwell they push a button. The nurses then receive a message with the room number on a beeper. This brings up the question: which nurse goes to the room? The closest one? the one on call, etc.? Current systems often have a very static nature as call buttons have fixed locations, e.g., on the wall next to the bed. There is an increased risk when patients become unwell inside a hallway, staircase or outside as they cannot use the nurse call system. Additionally, the current nurse call algorithms consist of predefined links between beeper numbers and rooms. Consequently, the system presently does not take into account the various factors specific to a given situation, such as the pathology of a patient, e.g., heart patient or confused, nor the competences of the staff, e.g., nurse or caregiver.

The increased introduction of electronic devices in continuous care settings facilitated the development of the ontology-based Nurse Call System (oNCS), which allows patients to walk around freely and use wireless nurse call buttons. Additionally, this platform manages the profiles of staff members and patients in an ontology. A sophisticated nurse call algorithm was developed by the authors. It first determines the priority of the call using probabilistic reasoning algorithms, which take into account the origin of the call and the pathology of the patient. Next, the algorithm finds the most appropriate staff member to handle the call. It dynamically adapts to the situation at hand by taking into account the context, e.g., location of the staff members and patients, the priority of the call and the competence of the different caregivers. The oNCS was implemented according to the *Context-Aware Platform* architecture discussed in Section 3 and visualized in Figure 1. A detailed description of this platform can be found in Ongenae et al. (2011d).

The oNCS is also able to automatically launch context calls based on the data generated by the electronic equipment and sensors in the environment, e.g., when a patient spikes a fever or when the light intensity is too high in the room of a patient with a concussion. It is however very difficult for developers to determine in advance all the risky situations for which a context call should be launched. These parameters and their thresholds are very dependent on the specific environment where the oNCS is deployed. Moreover, some of the relations between parameter measurements and calls made by the patient might not even be directly apparent to the caregivers as these relations are not rigorously studied.

To detect relations between the parameter measurements and the calls made by patients, the oNCS was extended with *Monitoring Algorithms*, the *Configuration Module* and *Learning Engine*. To evaluate this extension, a relation was simulated and it was investigated whether the *Learning Engine* was able to detect this trend and add it to the *Knowledge Base*. The trend that patients make a call when they exhibit symptoms for Systemic Inflammatory Response Syndrome (SIRS) (Davies and Hagen, 1997; Nyström, 1998) was chosen as simulated relation. This medically relevant use case could be easily generated, but is challenging for the *Learning Engine* to detect. SIRS is a generalized inflammatory reaction of the organism to a severe medical condition such as acute pancreatitis, severe burn injury, trauma, surgical procedure or infection. If SIRS is the response to an infection, the patient is diagnosed with sepsis. Sepsis has a high mortality rate (30%-40%). The criteria for diagnosing a patient with SIRS are:

- Tachycardia: heart rate > 90 beats per minute (bpm)
- Fever or hypothermia: body temperature > 38 °C or < 36 °C
- Tachypnea: arterial partial pressure of carbon dioxide ($PaCO_2$) < 32 mmHg
- White Blood Cell (WBC) count < 4,000 cells/mm^3 or > 12,000 cells/mm^3

For the diagnosis of SIRS, two or more of these criteria must be fulfilled. This is a challenging scenario for the *Learning Engine* as it involves both parameters measured at regular intervals by sensors, i.e., the heart rate and body temperature, as well as parameters obtained through the analysis of a blood sample by the laboratory, i.e., WBC and $PaCO_2$. Moreover, a combination of conditions needs to be fulfiled before the call should be launched.

The following sections illustrate how the *Learning Engine* was implemented and the *Learning Pipeline* was constructed, using the (abstract) classes discussed in Section 4, to detect this relation and add it to the *Knowledge Base*. The resulting pipeline is visualized in Figure 5.

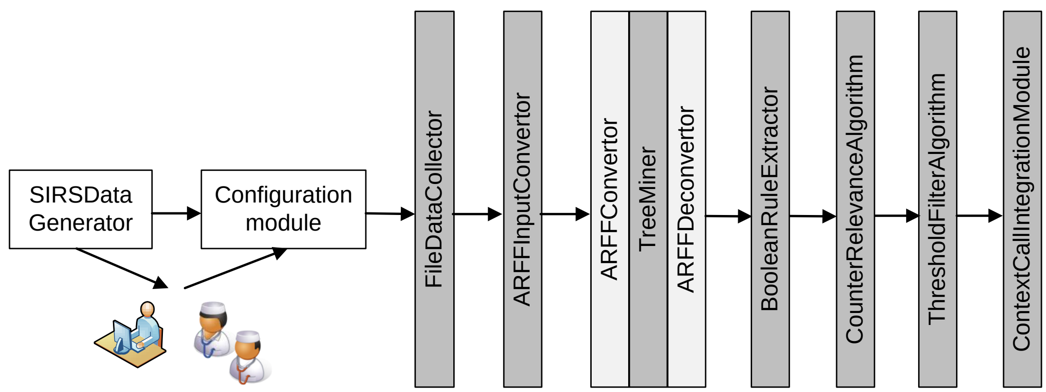


Figure 5: The pipeline used by the *Learning Engine* to tackle the SIRS use case

5.2. *Scenario implementation*

5.2.1. *Generating the SIRS data*

To realize the scenario, a dataset needs to be generated in which the trend can be detected that patients make calls when they exhibit SIRS symptoms. This dataset consists of a set of instances, each consisting of five data values, namely a value for the four SIRS parameters and whether or not a call was made. A *SIRS Instance* is defined as an instance, which consists of a combination of the four SIRS parameters that fulfills two or more SIRS criteria. Logically, a *Non-SIRS Instance* is defined as an instance, which fulfills at most one SIRS criterion at a time.

When the different instances are generated, each instance has 15% chance of being a *SIRS Instance*. The parameter values are randomly generated, while ensuring that at least two parameters fulfill the SIRS criteria for *SIRS Instances* and at most one criterion is fulfilled for *Non-SIRS Instances*. The values are generated within realistic bounds, e.g., temperature must be lower than 43 °C. Whether the *SIRS Instance* fulfills two, three or four criteria and whether the *Non-SIRS Instance* fulfills one criterion or none, is also randomly chosen.

Finally, each instance needs to be associated with a context call or not. To achieve a realistic dataset, noise is introduced by wrongly classifying the instances, i.e., associating *Non-SIRS Instances* with a call and vice versa. A noise percentage of x means that each *Non-SIRS Instance* has $x\%$ chance of being associated with a call and vice versa.

Some example instances are illustrated in Table 1. The first four instances are *Non-SIRS Instances*, while the latter four are *SIRS Instances*. The parameter values that fulfill SIRS criteria are indicated in italic. The calls marked with a *-symbol represent noise. A *Data Generator* was written to create the needed instances and provide them in the ARFF format, i.e., the data format used by WEKA. The resulting file is stored in the *Persistence Layer*.

5.2.2. The oNCS and continuous care ontologies

As mentioned in Section 2, little work has been done on the development of high-level ontologies, which can be used to model context information and knowledge utilized across the various continuous care settings, e.g., hospitals, homecare and residential care settings. Therefore we developed the *Continuous Care Ontology*, which models the generic context information gathered by the various sensors and devices, the different devices, the various staff members and patients and their profile information, medical conditions, roles and competences and the variety of tasks that need to be performed. A detailed description of this ontology can be found in Ongenae et al. (2011a). The

Heart rate	**Body temperature**	**$PaCO_2$**	**WBC count**	**Call**
61.42	*38.62*	34.54	4969	No
78.55	37.47	32.68	7746	No
88.37	*35.76*	46.53	7253	Yes*
67.92	36.10	42.53	*12096*	Yes*
66.63	*40.95*	*30.56*	*3740*	Yes
91.59	36.78	*29.94*	*12301*	No*
94.52	*40.67*	*28.89*	4866	Yes
95.23	*35.93*	*31.61*	8737	No*

Table 1: Some example instances of the SIRS dataset

most important classes of these ontologies pertaining to the use case are visualized in Figure 6. This ontology references the *Galen Common Reference Model* (Rector et al., 2003) as *Medical Ontology*. The concepts from the *Galen Common Reference Model* are preceded by the `galen` namespace in Figure 6. The concepts preceded with the `temporal` namespace are imported from the *SWRLTemporalOntology* (O'Connor and Das, 2010) to represent temporal information. Finally, the *Wireless Sensor Network (WSN) Ontology* (Verstichel et al., 2010) was imported, as shown by the concepts preceded by the `wsn` namespace, to represent the knowledge pertaining to observations made by sensors. The *Probabilistic Domain Ontology* then models the specific properties of the environment where the oNCS is deployed, e.g., the specific roles and competences of the staff members and how they map on

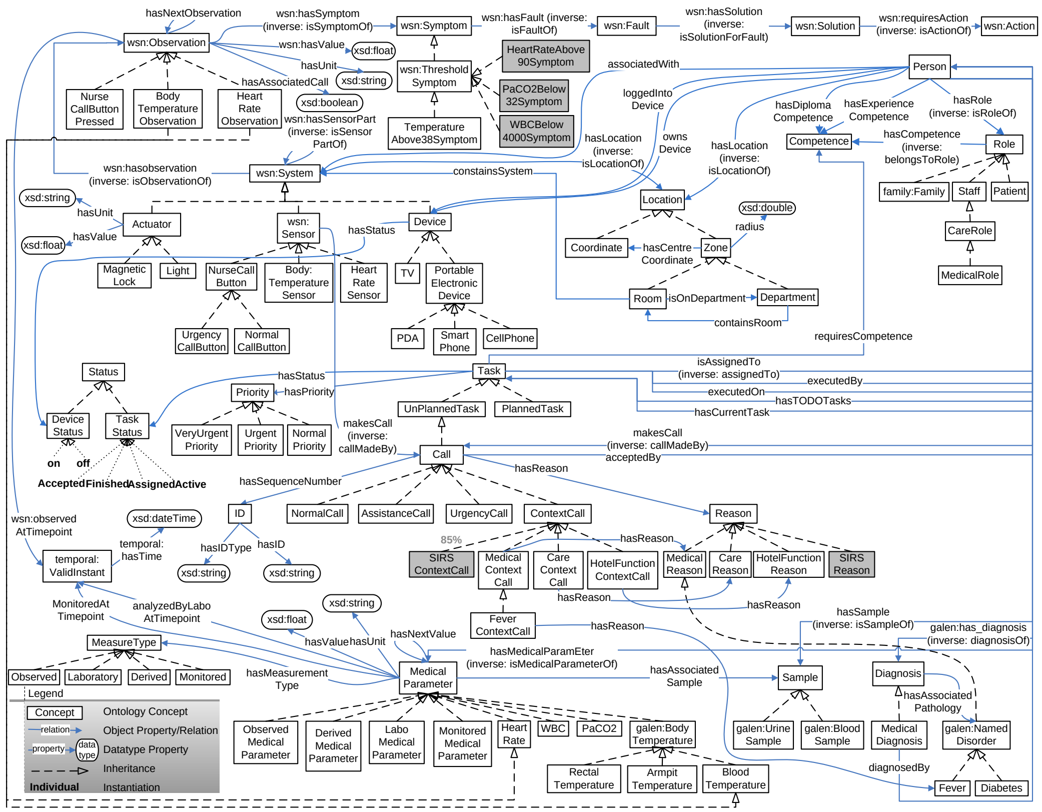


Figure 6: Prevalent classes of the *Continuous Care*, *Medical* and *Probabilistic Domain* ontologies pertaining to the SIRS use case

each other.

As can be seen, the model contains a `System` concept which models a system and its components. The ontology allows interpreting the data values monitored by the sensors. For this the ontology uses an observation pattern. A data value monitored by a system is modeled in the ontology as an `Observation`. Rules and axioms added to the ontology allow detecting specific phenomena in these observations, which are modeled as `Symptom` concepts. For example, the `TemperatureAbove38Symptom` class is defined as follows:

BodyTemperatureObservation AND $\exists$hasValue “ $>$ 38”

This axiom ensures that a `BodyTemperatureObservation` of more than 38 °C is reclassified as a `TemperatureAbove38Symptom`. Similarly, symptoms can also be reclassified as faults and even as solutions and actions that should be taken.

People are modelled through the `Person` concept and their roles and competences can be indicated. It can also be indicated with which person the sensors are associated through the `associatedWith` relationship. The medical parameters collected about a patient, either by sensors, the observations of staff members or the analysis of blood samples, are modelled as `MedicalParameters`. Similar to observations, these parameters can also be reclassified as symptoms. The medical condition of a person can also be modeled, e.g., `Fever`.

To model the daily activities of the caregivers and patients, the `Task` concept is used, which is further divided into planned and unplanned tasks. Each task can be assigned a `Status`, e.g., `Active` or `Finished`, a `Priority` and the competences which are needed to execute the task. People can be connected to the tasks through various relationships, e.g., `hasCurrentTask`, `isAssignedTo` or `executedBy`. A Call is modelled as an `unPlannedTask`. A call can be associated with a `Reason`, e.g., `Fever`. Four types of calls can be discerned. A `NormalCall` is a call made by a patient, while an `AssistanceCall` is launched by a caregiver to request help from another staff member. An `UrgencyCall` is only used for emergency situations, e.g., when a patient needs to be reanimated. Finally, a `ContextCall` is call that is automatically generated by the oNCS as a consequence of certain conditions being fulfilled. Consider for example the following Jena rule:

[FeverContextCall:

(?symp rdf:type oncs:TemperatureAbove38Symptom)

noValue(?symp task:hasAssociatedCall)

(?symp wsn:isObservationOf ?system)

(?kind rdf:type oncs:FeverContextCall)

→

createContextCall(?system, ?kind)

(?symp task:hasAssociatedCall ‘true’ˆˆxsd:boolean)]

The first line represents the name of the rule. First, it is sought if a body temperature of more than 38 °C was observed for which a call has not been launched yet. Next, the system that made the observation is retrieved. Finally, the type of call that should be created is specified. As a result, the functor `createContextCall` is called, which creates a `ContextCall` of type `FeverContextCall` and associates the system that made the observation with this call. The functor also assigns the status `Active` to the call. Moreover, the `hasAssociatedCall` relationship is set to `true` to make sure that the rule does not fire again.

The oNCS contains rules that fire when active calls are added to the ontology. Based on the context information, these rules assign the most appropriate staff member to the call. More information about these assignment rules can be found in Ongenae et al. (2011d).

Similar to how the fever example was modeled, the SIRS use case can be easily represented using these classes. Individuals of type `BodyTemperatureSensor` and `HeartRateSensor` are created to represent the sensors that measure the medical parameters of the patients. These sensors make observations of type `BodyTemperatureObservation` and `HeartRateObservation` respectively, which are associated with their sensors through the `hasObservation` relation. The measured value is indicated with the `hasValue` relation. Individuals of type `BloodSample` are created, that represent the blood samples analyzed by the laboratory to determine the WBC count and $PaCO_2$ of the patient. These results are captured in the ontology as medical parameters of type `WBC` and `PaCO2`. They are associated with their blood sample through the `hasAssociatedSample` relationship. Finally, when a patient makes a call by pressing a button, an individual of type `Call` is created in the ontology, which is connected through the `callMadeBy` relationship with the patient. Through reasoning, this call is reclassified as a `NormalCall` as it is made by a person with as role `Patient`.

A mobile nurse call application was also developed, which is used by the caregivers to receive, assess and accept, i.e., indicate that they are going to handle, calls. A nurse can also use the application to contact other staff members or the patient, e.g., to request the reason for the call or to give feedback. Before a nurse is able to indicate a call as finished, the reason for the call must be indicated either on the mobile application or the terminal next to the bed of the patient. This reason is also entered in the ontology. The mobile application is further explained in Ongenae et al. (2011c).

5.2.3. Collecting the data & input conversion

As the data is generated, no *Monitoring Algorithms* are needed. However, a *Monitoring Algorithm* could easily be written as follows. Relationships need to be found between medical parameters of patients and the calls that they make. The *Context Call Monitoring Algorithm*, monitors the ontology for calls of type `NormalCall`. When such a call is added to the ontology, the algorithm collects the most recent value for each medical parameter that is measured about the patient who made the call. This information can easily be retrieved using SPARQL queries. As not every medical parameter is measured for every patient, the dataset possibly contains missing values. When the call has been completely handled by the caregiver, the algorithm also retrieves the reason, which was attached to the call. As such, different data sets can be created, grouping calls together which have similar reasons. These datasets can differ in granularity of the reason. For example, a dataset could be created for all the calls with a `MedicalReason` or for all the calls with the more specific reason `Fever`. All calls of the second dataset would also be part of the first dataset, as `Fever` is a subclass of `MedicalReason`. Each of these datasets could be used as input for the *Learning Engine*. Other ways of grouping the data instances can also be employed, e.g., grouped per patient or grouping the instances of patients that have a similar pathology. The *Context Call Monitoring Algorithm* keeps track of how many instances have been collected for each dataset. When a representative amount has been gathered, the dataset is expanded with negative examples. For example, the medical parameters of the patients already present in the dataset can be collected at a timepoint when they have not seen a caregiver or made a call for a while or at a timepoint they made a call for a different reason. Finally, the *Monitoring Algorithms* invoke the *Configuration Module* to initiate the *Learning Engine*. The datasets can also be intermediately shown to the *Stakeholders* and *Application Developers* for inspection. In this use case, the *Data Generator* takes on the role of the *Monitoring Algorithm*.

The *Monitoring Algorithms* can store the datasets in the *Persistence Layer* in a format that best suits their needs. For the *Data Generator*, the ARFF format was chosen. Ontology individuals could also be directly stored in a triple store. The *Monitoring Algorithm* or the *Data Generator* indicates the location of the data and its format to the *Configuration Module*. They also indicate which `MiningManager` should be used to process the data. Different types of *Learning Pipelines*, which each consist of a combination of filters that suit the needs of a particular use case, can be created by implementing several subclasses of the `MiningManager`. This allows multiple *Monitoring Algorithms* to run at the same time and the collected data to be processed by the `MiningManager`, and thus *Learning Pipeline*, that matches with the goal of the algorithm.

The *Configuration Manager* configures the `MiningManager` to use the appropriate `DataCollectionModule` and `InputConvertor` that suits the format of the data. The subclass `FileDataCollector` of the `DataCollectionModule` class was implemented, which is able to read the data from a file at a specified location. The result is a `String`, which is provided to the appropriate `ARFFInputConvertor`. This subclass of `InputConvertor` is able to translate this `ARFF-String` to the `LearningDataObject` format, which is used by the *Learning Pipeline*. During the translation it also checks if the specified value for an attribute, e.g., 38 for the body temperature parameter, is compatible with the type of this attribute. For example, it is not allowed to assign a `String` to a numerical attribute. Illegal data instances are discarded.

5.2.4. Mining the sensor data using a C4.5 decision tree

A *Pre-Processing* filter was not implemented for this use case, as it works on generated data. If the previously discussed *Context Call Monitoring Algorithm* was used, several *Pre-Processing* filters could be used. For example, a `RemoveOutliers` filter could be employed to remove outliers or impossible parameter values in the dataset. Moreover, the number of features, i.e., measured medical parameters, in the dataset would be relatively high. A `FeatureSelection` filter could be used to select the most interesting features for the *Data Mining* step. Finally, a `MissingValues` filter would be able to deal with the missing values in the dataset.

The *Data Mining* filter needs to find relations in the generated dataset between the sensor measurements and the occurence of a call. Supervised (Witten et al., 2011) classification techniques (Kotsiantis, 2007) consider a set of input attributes, e.g., the different sensor types, and a output attribute, also called the class attribute or the label, e.g., whether a call was made or not. These techniques then try to build a model that fits this data set and derives relationships between the input attributes and the label. Building a decision tree (Kotsiantis, 2013) based on the information captured in the dataset is a well-known and easy supervised classification technique. A decision tree is a tree structure in which each leaf represents a value that the label can assume, e.g., Yes or No. The internal nodes of the tree represent the attributes

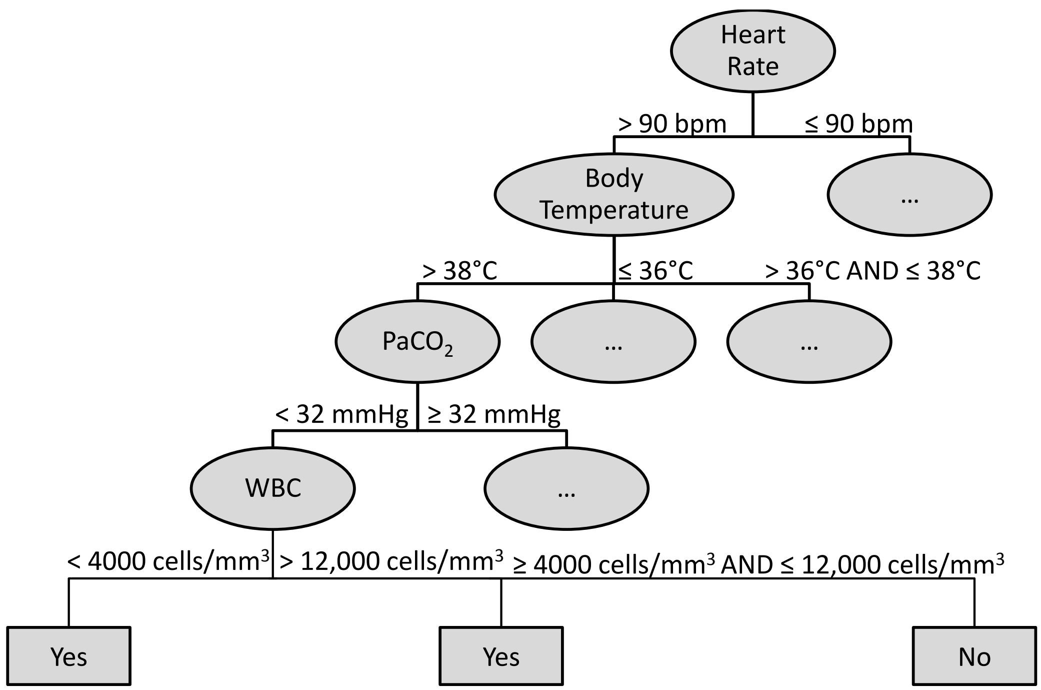


Figure 7: Part of the decision tree of the SIRS example

on which a decision is based, while the branches represent conditions that the attributes need to fulfill. As an example, a part of the decision tree of the SIRS example is shown in Figure 7. To determine the label of a certain data instance, one just needs to follow the tree from the root to the leaves along the branches for which the instance fulfills the conditions. Essentially, a decision tree forms a compact representation of classification rules. For example, the decision tree shown in Figure 7 contains the classification rule:

$$\begin{aligned}&\text{HeartRate} > 90 \text{ bpm AND BodyTemperature} > 38\,^{\circ}\text{C AND}\\ &\qquad\qquad < 32 \text{ mmHg AND} < 4000 \text{ cells/mm}^3 \rightarrow \text{Yes}\end{aligned}$$

Different techniques can be used to build such a decision tree out of a data set, e.g., the Iterative Dichotomiser 3 (ID3) (Quinlan, 1986) or C4.5 (Quinlan, 1993) algorithm. The latter is a more sophisticated algorithm as it allows that attributes have numeric values (Quinlan, 1996), is able to handle missing values and prunes the tree in order to make it more compact and avoid overfitting (Everitt and Skrondal, 2010).

To implement the C4.5 decision tree, an external library is used, namely WEKA. WEKA provides its own implementation of the C4.5 algorithm, namely J4.8, which was used in this research. A subclass of the `FilterInstance` abstract class was implemented, called `TreeMiner`. As previously mentioned, WEKA uses the ARFF data format to represent data. Therefore, an `ARFFDataObject` was created as a subclass of `DataObject` and (de)convertors were implemented that are able to translate the internal data format of the *Learning Engine*, i.e., `LearningDataObject`, to and from the ARFF data format. As mentioned in Section 4, it is enough to indicate in the `TreeMiner` that the filter uses the `ARFFDataObject` in the `getDataType` method and the framework will automatically use the correct (de)convertors to transform the data. Which attribute should be used as label can be indicated in the `TreeMiner` class. In case the label is not indicated, the `TreeMiner` assumes that the last attribute in the data format is the label. The `ARFFInputConvertor`, discussed in the previous section, makes sure that the last attribute is indeed the label. The `doProcessing` method then calls the Java API of WEKA to build the decision tree. However, the J4.8 algorithm does not allow to retrieve separates branches and nodes of the tree. Only a textual representation of the complete decision tree can be obtained. For example, the textual representation of the tree visualized in Figure 7 is:

N0 [label="HeartRate"]

N0 → N1 [label=" > 90"]

N1 [label="BodyTemperature"]

N1 → N2 [label=" > 38"]

N2 [label="PaCO2"]

N2 → N3 [label=" < 32"]

N3 [label="WBC"]

N3 → N4 [label=" < 4000"]

N4 [label="Yes"]

The nodes and branches are identified and translated to the `LearningDataObject` format such that the result can be forwarded to the next step in the pipeline. It is important to note that new results are always added to the data being exchanged, so that the original data set also stays available for the following steps in the pipeline.

The *Post-Processing* filter is responsible for deriving the rules out of the textual representation of the decision tree provided by the J4.8 algorithm. Therefore, the `BooleanRuleExtractor` subclass of the `FilterInstance` class was implemented. The implemented `doProcessing` method takes into account that the label has a boolean value, i.e., Yes or No. Only the branches that result in a positive leaf need to be translated into a rule, as only those rules will result in calls. The `doProcessing` method starts from a positive leaf and follows the branches until the root is reached. Each branch that is crossed is added as a condition in the rule. The iterative build-up of the rule according to the output of the J4.8 algorithm illustrated in Figure 7 is as follows:

Step 1: $\rightarrow Yes$

Step 2: WBC $<$ 4000 $\rightarrow$ Yes

Step 3: PaCO2 $<$ 32 AND WBC $<$ 4000 $\rightarrow$ Yes

Step 4: BodyTemperature $>$ 38 AND PaCO2 $<$ 32 AND
WBC $<$ 4000 $\rightarrow$ Yes

Step 5: HeartRate $>$ 90 AND BodyTemperature $>$ 38 AND
PaCO2 $<$ 32 AND WBC $<$ 4000 $\rightarrow$ Yes

The resulting rules are represented in the `LearningDataObject` format such that they can be processed by the *Decision Module*.

5.2.5. Filtering and integrating the rules

As mentioned in Section 3, probabilities are attached to the discovered rules to express their reliability to the users and to ensure that the *Knowledge Base* remains consistent, i.e., that the new knowledge does not contradict already existing knowledge.

To calculate the initial probability, the `CounterRelevanceAlgorithm` was implemented as a subclass of the `FilterInstance` class. This algorithm applies the rule to the original dataset, which is still included in the `LearningDataObject`. The percentage of times that the rule labels the data correctly, i.e., the conditions of the rule are fulfilled and the label is Yes, is used as probabilistic value. As the data for this use case was generated, this probability thus reflects the amount of noise in the dataset. For the remainder of the text, it assumed that the rule, which was presented in the previous section, receives a probability of 85%.

A simple filter algorithm, namely the `ThresholdFilterAlgorithm` was implemented as subclass of the `FilterInstance` class. This algorithm filters the rules for which the probability is lower than a specified probability, e.g., 50%. This rule is thus not added to the *Knowledge Base*. However, the rule and its associated probability is archived in the *Persistence Layer*.

Finally, the `ContextCallIntegrationModule`, a subclass of the `IntegrationModule` class, is responsible for integrating the rules and associated probabilities in the *Knowledge Base*. First, new subclasses of `ContextCall` and `Reason` are introduced in the ontology, with as name the condition of the rule added before the suffix `ContextCall` and `Reason` respectively. For brevity, `SIRSContextCall` and `SIRSReason` are used to refer to the concepts that are created for the rule, which is used as running example, i.e., the rule that fulfills each of the four criteria. These concepts are visualized in grey in Figure 6. Pronto is used to represent and reason on the probabilistic information in the ontology. To express generic probabilistic knowledge, Pronto uses Generic Conditional Constraints (GCCs) (Lukasiewicz, 2007). Generic means that the knowledge does not apply to any specific individual but rather to a fresh, randomly chosen one. A GCC is of the form (D—C)[l,u] where D and C are classes in the ontology and [l,u] is a closed subinterval of [0,1]. To represent these GCCs in the ontology, Pronto employs subsumption axiom annotations. For example, to express the fact that the `SIRSContextCall` is a `ContextCall` with only 85% probability, the following subsumption axiom annotation is added to the ontology:

< owl11:Axiom >

< rdf:subject rdf:resource="#SIRSContextCall" >

< rdf:predicate rdf:resource="&rdfs;subClassOf" >

< rdf:object rdf:resource="#ContextCall" >

< pronto:certainty > 0.85;1 < /pronto:certainty >

< owl11:Axiom >

Second, a `Symptom` concept is created for each parameter condition in the discovered rule, for example `HeartRateAbove90Symptom`, `BodyTemperature-Above38Symptom`, `PaCO2Below32Symptom` and `WBCBelow4000Symptom`. These classes are defined by axioms, for example the `HeartRateAbove90Symptom` is defined as:

HeartRateObservation AND ∃hasValue " > 90"

If a class with a similar definition already exists, the existing class is used. This can be checked by searching for equivalent classes in the ontology with a Reasoner. For example, `BodyTemperatureAbove38Symptom` is not added to the ontology, as `TemperatureAbove38Symptom` is an equivalent class. The newly created `Symptom` classes are visualized in grey in Figure 6.

Third, the rules are translated to a Jena Rule using the created classes and added to the *Knowledge Base*. For example, the rule from the previous section is translated to four Jena Rules. For example, the following Jena Rule launches when all the requirements are met and at least one of the symptoms does not have an associated call yet:

```
[SIRSContextCall:
(?symp1 rdf:type oncs:HeartRateAbove90Symptom)
noValue(?symp1 wsn:hasNextObservation)
(?symp2 rdf:type oncs:TemperatureAbove38Symptom)
noValue(?symp2 wsn:hasNextObservation)
(?symp3 rdf:type oncs:PaCO2Below32Symptom)
noValue(?symp3 medical:hasNextValue)
(?symp4 rdf:type oncs:WBCBelow4000Symptom)
noValue(?symp4 medical:hasNextValue)
noValue(?symp1 task:hasAssociatedCall)
(?symp1 wsn:isObservationOf ?system)
(?kind rdf:type oncs:SIRSContextCall)
→
createContextCall(?system, ?kind)
(?symp1 task:hasAssociatedCall ‘true’^^xsd:boolean)]
(?symp2 task:hasAssociatedCall ‘true’^^xsd:boolean)]
(?symp3 task:hasAssociatedCall ‘true’^^xsd:boolean)]
(?symp4 task:hasAssociatedCall ‘true’^^xsd:boolean)]
```

For each symptom a rule is created. The only difference between the rules is that the condition for an associated call is checked for a different symptom each time. This is because the different symptoms on their own might already have launched context calls for other reasons, e.g., the `TemperatureAbove38Symptom` might already have launched a `FeverContextCall`. Afterwards, all the symptoms are associated with a call to ensure that only one context call is launched. The rule also ensures that the most recent parameter values are taken into account by checking whether there are no next observations or parameter values through the `hasNextObservation` and `hasNextValue` relations.

When the rule is fulfilled, a new context call is added to the *Knowledge Base*. Consequently, the oNCS will detect the new context call and assign a staff member to it. The Pronto reasoner can then be used to retrieve the probabilistic information associated with the call. This information can then be conveyed to the assigned caregiver through the mobile application.

As only subclasses are added to the ontology and no knowledge is removed, it is unlikely that the ontology will become inconsistent. However, if the ontology does become inconsistent, the following solution can be employed. When new information is added to the ontology, the consistency is checked. If the ontology is no longer consistent, the information is identified with which the new knowledge conflicts. Pronto allows that different chunks of probabilistic information conflict with each other. For example, a bird is flying object with high probability and all penguins are birds, but a penguin has a low probability of flying. More specific probabilistic constraints are thus allowed to override more generic ones. The conflicting information is annotated with the probabilistic interval [1,1], which indicates that the knowledge is generally true. Consequently, we are now dealing with conflicting, probabilistic knowledge and the rule of increasing specificity can be employed to resolve the conflict. As such, we ensure that the ontology remains consistent.

Finally, the *Integration Module* also associates the learned knowledge with information about the *Learning Engine* that created it by using concepts from the *Learning Ontology*. The individuals, which are created to realize this goal, are visualized in bold in Figure 2.

Note that `ContextCall`, `Symptom` and `Reason` concepts and an associated probabilistic annotation axiom and Jena Rule are created for each discovered rule.

5.2.6. Adapting the probabilities

This step was not implemented as it requires the system to be deployed such that information about the usage of the new knowledge by the caregivers can be acquired. However, it is briefly discussed how this task of adapting the probabilities could be realized for this use case.

A *Monitoring Algorithm* could be implemented, which takes as parameter the newly created context call, e.g., in this case `SIRSContextCall`. The algorithm monitors the *Knowledge Base* and collects calls of this type, which have been launched by the system. For each call, its reason and the symptoms that caused the calls to be launched are retrieved. When nurses handle calls, they need to input the reason for the call. For context calls, they can affirm the reason, which was assigned by the framework, e.g., SIRS. They can also choose to change it, e.g., to false because the call was unnecessary. As such a dataset is created for each rule, which maps the values of the medical parameters on the associated reason. When a representative amount of data has been collected, this dataset can be retrieved by the `FileDataCollector` and converted by the `ARFFInputConvertor`. The output can then be processed by a *Learning Pipeline* consisting of only one filter. This filter is a *Probabilistic Relevance Algorithm*, which simply calculates the percentage of calls for each rule for which the reason was not changed. This means that caregivers deemed the reason to be correct. This percentage is than given to the *Integration Module*, which adapts the probability for this rule in the ontology to this calculated percentage. As explained in Section 3.3.3, if the calculated percentage exceeds or falls below the probability thresholds specified in the *Integration Module*, the knowledge is removed from the ontology or added as generally accepted knowledge without a probability.

5.3. Evaluation set-up

To evaluate the applicability of the framework, it is important to assess the correctness of the derived rules. The correctness of the used data mining techniques is influenced by the size of the dataset and the amount of noise. To assess the influence of the latter, the *Learning Pipeline* was consecutively applied to datasets of the same size, but with an increasing amount of noise. The amount of noise is varied from 0% to 50% in steps of 1%. As mentioned in Section 5.2.1, a noise percentage of x means that each *Non-SIRS Instance* has x% chance of being associated with a call and vice versa. It is unnecessary to increase the noise percentage beyond 50% as a random label is assigned at this point and the dataset becomes meaningless. The amount of noise needs to be increased in a dataset of realistic size. The WBC and $PaCO_2$ parameters are derived by the laboratory by analyzing a blood sample. Consequently, it is unlikely that more than two different values for these parameters will be generated per patient per day. If we assume that a department contains on average 30 patients and that we want to wait at most 28 days before we run the self-learning framework for the first time, a realistic dataset contains 1,680 instances, i.e., 30 patients x 28 days x 2 entries per patient per day.

The influence of the size of the dataset on the correctness is evaluated by consecutively applying the *Learning Pipeline* to datasets of increasing size. The dataset sizes range from 100 to 2,000 instances in steps of 100 instances. It can be noted that this range also contains the size of the dataset used for the correctness tests that evaluate the influence of noise, i.e., 1,680 instances.

It is also important to evaluate the performance, i.e., execution time and memory usage, of the developed *Learning Engine*. Although, the learning process will mostly run in the background, it is important to assess the amount of resource usage. Most healthcare environments have a limited amount of resources and delegating the processing to the cloud is often difficult because of privacy issues. To evaluate the influence of noise on the performance, the same datasets were used as for the correctness tests. However, to assess the influence of the size of the dataset, datasets were generated with sizes ranging from 1,000 to 30,000 in steps of 1,000 instances. Bigger datasets were used as it is important to explore the limits of the proposed framework.

To achieve reliable results, each test was repeated 35 times, of which the first three and the last two were omitted during processing. For each run, a new dataset was generated. Finally, the averages across the 30 remaining runs are calculated and visualized in the form of graphs. The tests were performed on a computer with the following specifications: 4096 megabyte (MB) (2 x 2048 MB) 1067 megahertz (MHz) Double Data Rate Type Three

Synchronous Dynamic Random Access Memory (DDR3 SDRAM) and an Intel Core i5-430 Central Processing Unit (CPU) (2 cores, 4 threads, 2.26 gigahertz (GHz), 3 MB cache).

The term detection rate is introduced to assess the correctness. The SIRS use case is detected when the criteria for each of the four parameters of the SIRS use case are discovered. If one or more criteria is not learned, the SIRS use case is considered undetected. The detection rate of a dataset with a particular size is defined as the percentage of the 30 test runs for this size for which the SIRS use case was completely detected. For example, a detection rate of 50% for a dataset of 100 instances means that for 15 test runs of this dataset size the SIRS criteria were detected.

To assess the correctness, the relative error of the SIRS criteria is calculated. The relative error expresses how much the learned criterion deviates from the actual SIRS criterion. For example, a relative error of 5% for the "*heart rate* $>$ *90*" criterion indicates that the discovered threshold deviates from 90 by 5%. Note that the body temperature and WBC parameters have both an upper and lower threshold, while the heart rate and $PaCO_2$ have only one threshold.

5.4. Results

5.4.1. Correctness

Figure 8 depicts the detection rate as a function of the size of the dataset. The detection rate is relatively low for small datasets, but it quickly increases and reaches 100% for a dataset with 800 instances. When a dataset contains at least 1,000 instances, the detection rate is always 100%.

The detection rate is off course related to the relative error. In Figure 9

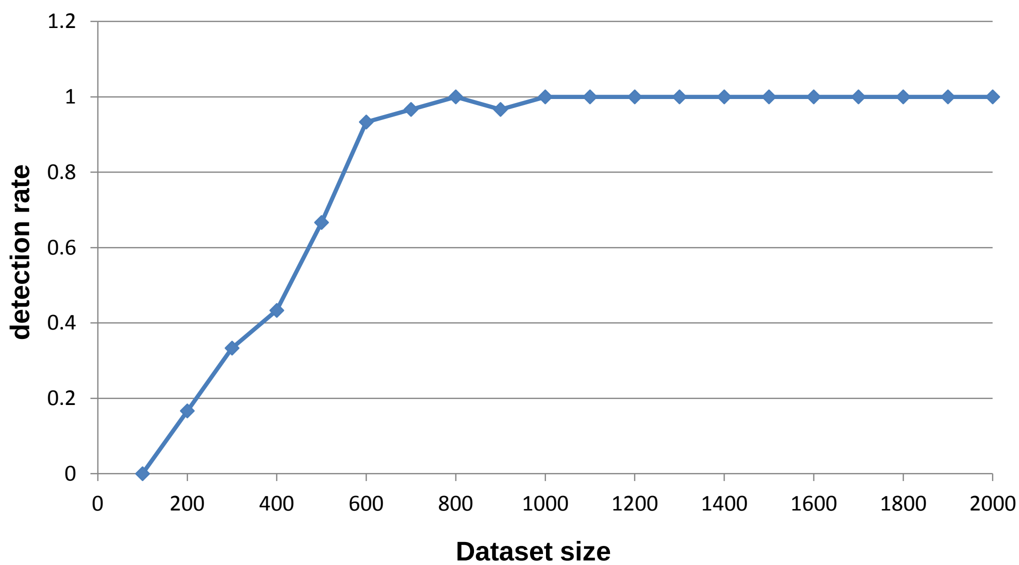


Figure 8: The detection rate of the SIRS use case as a function of the size of the dataset

the relative error is depicted for each of the SIRS criteria as a function of the size of the dataset. A missing value, i.e., the criterion was not learned, corresponds to a relative error of 100%. Consequently, a low detection rate corresponds to high relative error. When the dataset reaches a 1,000 instances and a detection rate of 100% is thus achieved, the relative error stays below 1%. This means that for a dataset of 1,000 instances, the threshold is discovered for each criterion and it only deviates from the actual threshold by at most 1%, which is a very good result. If we consider that the parameters are collected twice a day for each patient in a department with 30 patients, enough instances would be collected after 17 days.

Figure 10 visualizes the relative errors for each of the SIRS criteria as a function of the amount of noise in a realistically sized dataset of 1,680

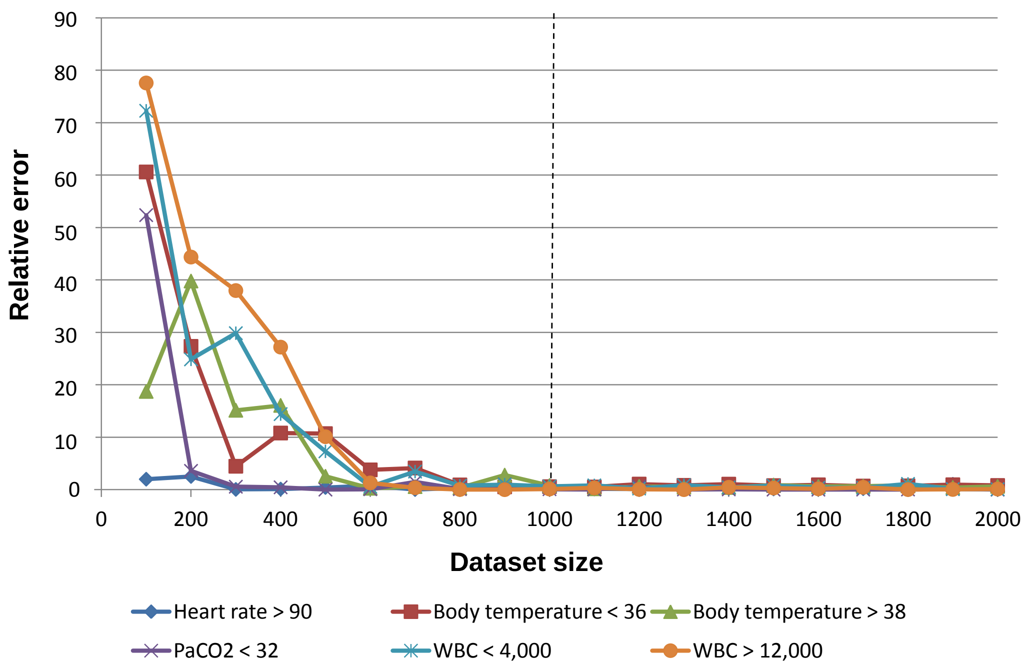


Figure 9: The relative errors for each of the SIRS criteria as a function of the size of the dataset

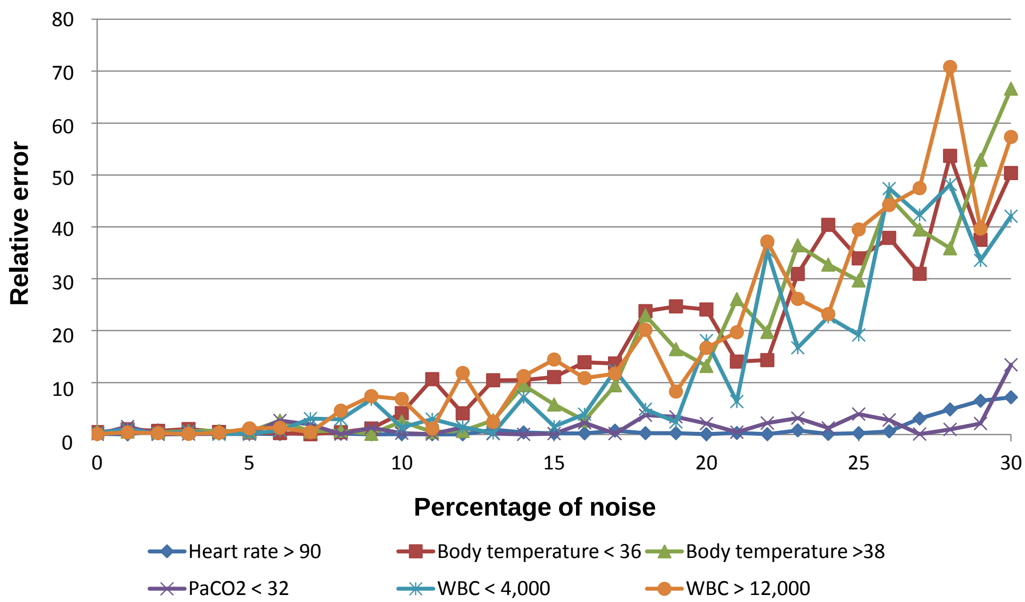


Figure 10: The relative errors for each of the SIRS criteria as a function of the amount of noise in the dataset

instances. It is clear that the *Learning Engine* is insensitive to a noise rate of less than 5%. If the amount of noise increases, the relative errors quickly rise to 10% and higher. A relative error of 10% on the lower threshold of the body temperature, already implies a difference of 3.6 °C. This is unacceptable. In contrast, a relative error of 10% on the lower bound of the WBC only indicates a difference of 400 cells/mm^3. The acceptability of the relative error thus depends on the kind and range of the parameter.

5.4.2. Performance

The execution time as a function of the size of the dataset is depicted in Figure 11. Only the execution time of the most relevant pipeline steps

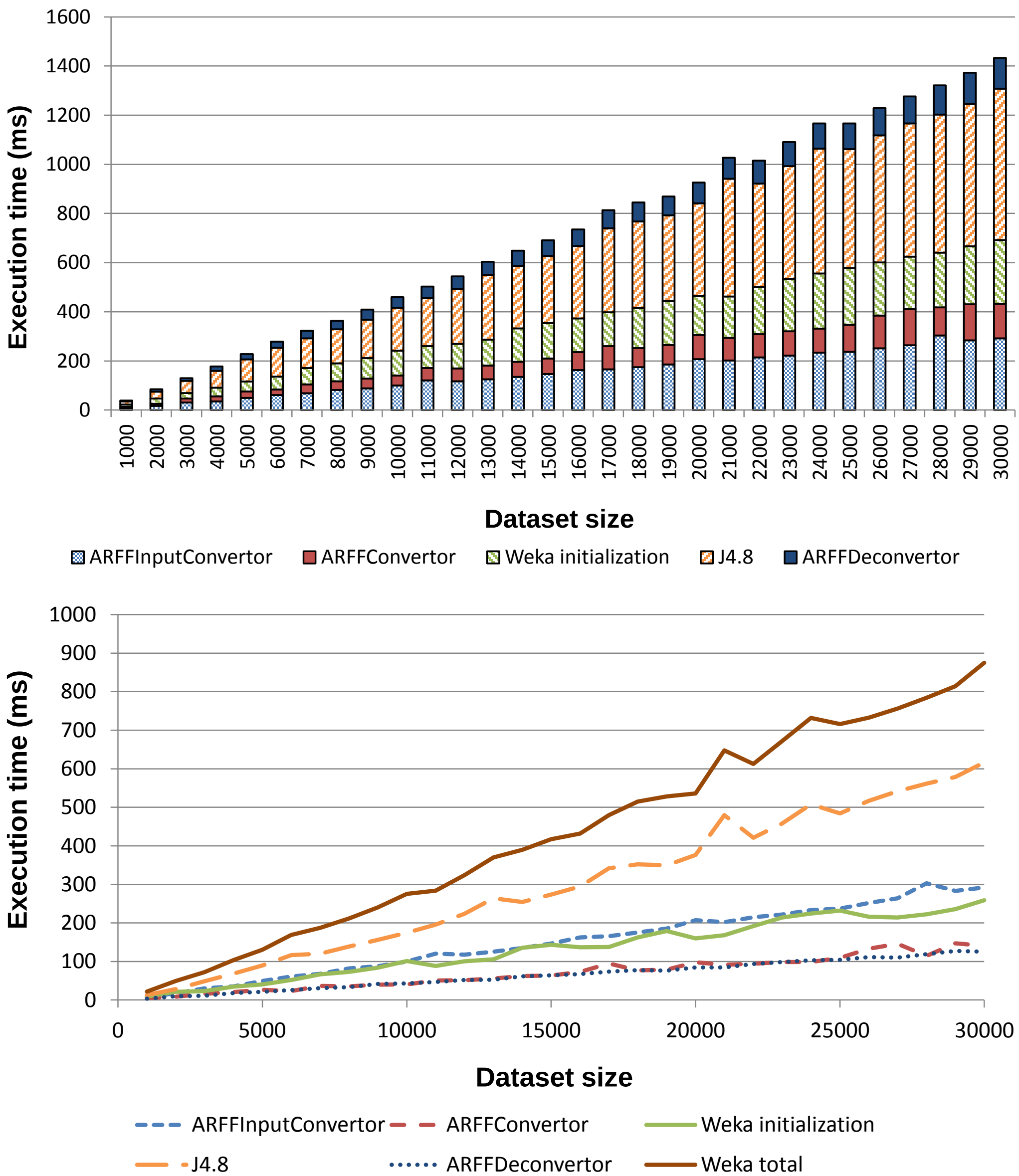


Figure 11: Execution time as a function of the dataset size

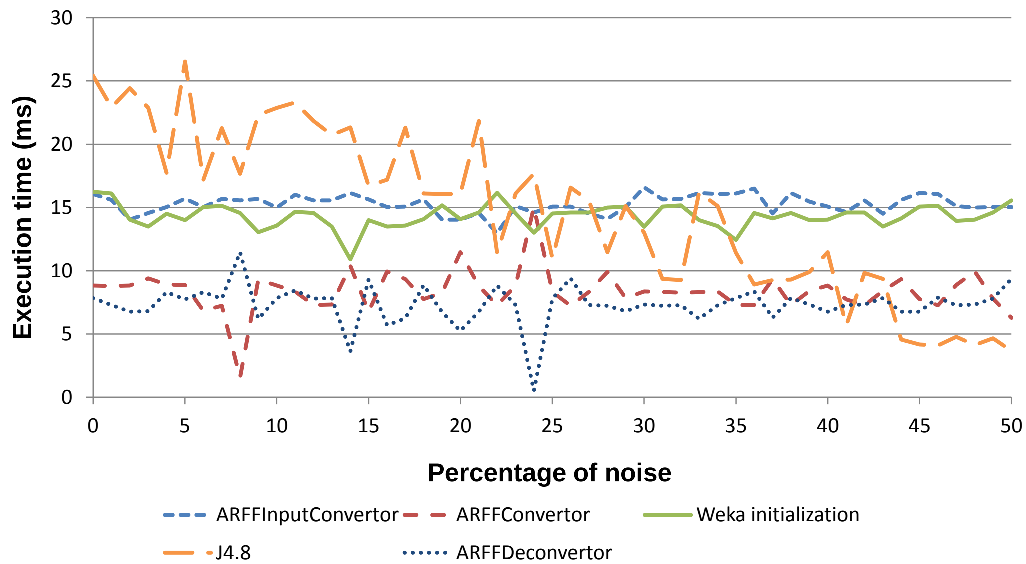
is shown. The execution time of the `CounterRelevanceAlgorithm` and `ThresholdFilterAlgorithm` is negligible compared to the execution times of the visualized modules. The execution time of the `ContextCallIntegrationModule` depends heavily on the complexity and the amount of data in the ontology. As the ontology was not initialized with a realistic data set, e.g., representing a realistic amount of staff members and patients, the execution time of this module is not shown. The size of the decision tree build by WEKA depends on the number of attributes in the dataset, but is independent of the number of instances, i.e., the size of the dataset. As the number of attributes, namely the four SIRS parameters and the label, stays constant and the Post-Processing step only processes the model build by WEKA, the execution time of this step is not influenced by the size of the dataset. Moreover, the execution time of the `BooleanRuleExtractor` was also negligible compared to the execution times of the depicted modules. The processing of the data by WEKA can be split up into two steps, namely transforming the ARFF format to Java Objects and the actual execution of the J4.8 algorithm to build the model. The execution times of both these steps are visualized.

It can be derived from Figure 11a that the execution time of the self-learning framework is linear as a function of the size of the dataset. The execution of the J4.8 algorithm by WEKA consumes the largest amount of execution time. It can also be noted that the `ARFFInputConvertor` consumes a considerable amount of execution time. This *InputConvertor* needs to translate a `String`-based representation of an ARFF-file to the internal data format used by the *Learning Pipeline*, namely `LearningDataObject`. Moreover, it needs to check if each value also fulfills the type requirements of the attribute, e.g., that a `String` is not provided where a numerical value is expected. The `ARFFConvertor` and `ARFFDeconvertor`, which are used by the Data Mining step to translate the internal data format to and from the ARFF format used by WEKA, are more performant. This is because these convertors translate to and from a Java Object representation of the internal format, which is more structured and is thus processed more easily.

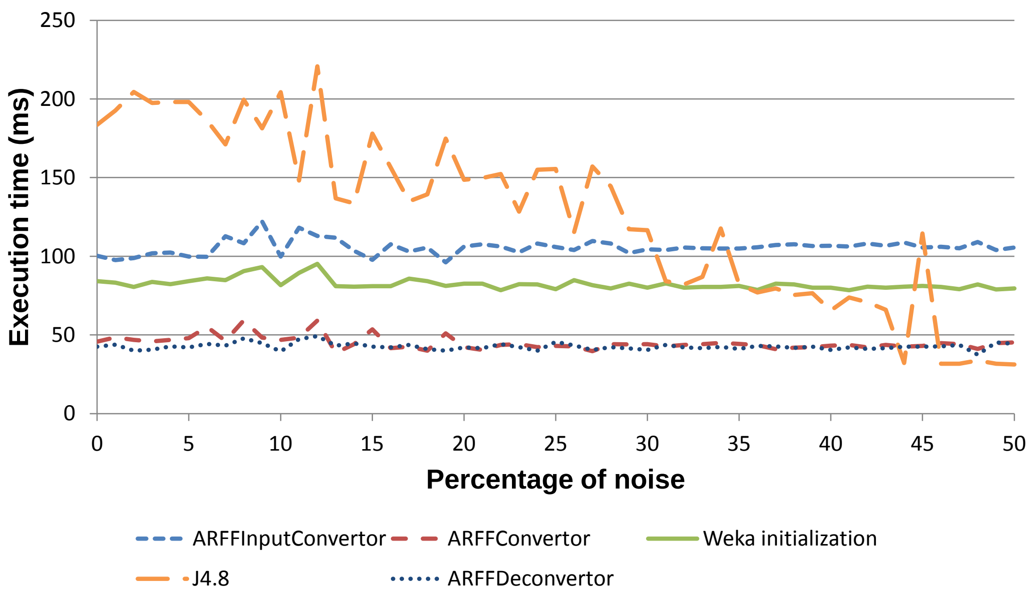
Figure 11b illustrates that the execution time of each of the visualized modules is also linear as a function of the size of the dataset. The complexity of the J4.8 algorithm is $O(m * n^2)$ for a dataset with m instances and n attributes (Su and Zhang, 2006). Since the number of attributes is constant in this use case, this reduces to a complexity, which is linear in the number of instances, i.e., $O(m)$. The `ARFFInputConvertor`, `ARFFConvertor` and `ARFFDeconvertor` are also linear in the size of the dataset, as they need to (de)convert all the instances in the dataset one by one.

The execution time needed to process the dataset of realistic size, i.e., 1,680 instances, is lower than 100 ms, which is a negligible delay. This means that the monthly patient data of a department with on average 30 patients can be processed very efficiently.

Figure 12a depicts the execution time as a function of the amount of noise for the realistic dataset containing 1,680 instances. As the measured execution times are quite small, i.e., lower than 30 ms, the graphs are quite erratic and unpredictable. To get a clear view on the underlying trends, the performance tests were repeated for a dataset consisting of 5,000 instances. The amount of noise in this dataset is also gradually increased. The resulting graph is visualized in Figure 12b. It can be noted that only the execution time of the J4.8 algorithm is influenced by the amount of noise in the dataset. The



(a) Dataset of 1,680 instances



(b) Dataset of 5,000 instances

Figure 12: Execution time as a function of the amount of noise in the dataset

Figure 13: The memory usage as a function of the size of the dataset

execution time of the J4.8 algorithm decreases as the amount of noise in the dataset increases. It can be derived that the execution time decreases faster when the percentage of noise is higher than 5%. As shown in the previous section, the relative error quickly increases once the amount of noise rises above 5%. This is because the J4.8 algorithm will more quickly decide that it is no longer useful to try to split up the decision tree. On the one hand, this leads to a lower detection rate as not all the criteria are discovered. On the other hand, this decreases the needed execution time of the algorithm.

Figure 13 illustrates the memory usage of the framework as a function of the size of the dataset. The fluctuating pattern of the graphs can be explained by the memory that is consumed by the *Garbage Collector* in Java. However, trend lines can clearly be discerned. It can be noted that the memory usage is linear as a function of the amount of instances. Moreover, the total amount of consumed memory stays quite low, i.e., at most 80 MB. For the realistic dataset of 1,680 instances the memory usage is negligible, namely about 10 MB.

It can be concluded that a dataset of realistic size for the SIRS use case can be processed by any modern PC or server and no cloud-based solutions are needed to run the framework.

6. Conclusions

In this paper a self-learning, probabilistic, ontology-based framework was presented, which allows context-aware applications to adapt their behavior at run-time. The proposed framework consists of the following steps. First, an ontology-based context model with accompanying rule-based context-aware algorithms is used to capture the behavior of the user and the context in which it is exhibited. Historical information is then gathered by algorithms that identify missing or inaccurate knowledge in the context-aware platform. This historical information is filtered, cleaned and structured so that it can be used as input for data mining techniques. The results of these data mining techniques are then prioritized and filtered by associating probabilities, which express how reliable or accurate they are. These results and the associated probabilities are then integrated into the context model and dynamic algorithms. These probabilities clarify to the stakeholders that this new knowledge has not been confirmed by rigorous evaluation. Finally, these probabilities are adapted, i.e., in- or decreased, according to context and behavioral information gathered about the usage of the learned information.

The pipeline architecture of the framework was presented and its implementation was detailed. Finally, a representative use case was presented to illustrate the applicability of the framework, namely mining the reasons for patients' nurse call light use to automatically launch calls. More specifically, detecting SIRS as a reason for nurse calls was used as a realistic scenario to evaluate the correctness and performance of the proposed framework. It is shown that correct results are achieved when the dataset contains at least 1,000 instances and the amount of noise is lower than 5%. The execution time and memory usage are also negligible for a realistic dataset, i.e., below 100 ms and 10 MB.

Future work will mainly focus on the development of more intricate monitoring, probabilistic relevance and filter algorithms. Moreover, a prototype of the proposed framework will be deployed and evaluated in a real-life setting.

Acknowledgements

We would like to acknowledge that part of this research was supported by the iMinds Project ACCIO co-funded by the IWT, iMinds and the following partners: Televic NV, Boone NV, Dominiek Savio Instituut and In-Ham. F. Ongenae and M. Claeys would like to thank the IWT for financial support through their Ph.D. grant. The authors thank F. De Backere for support in constructing the figures.

References

Anderston, J., Aydin, C., 1997. Evaluating the impact of health care information systems. International Journal Technology Assessment in Health

Care 13 (2), 380–393.

Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., Patel-Schneider, P., 2003. The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press.

Baldauf, M., Dustdar, S., Rosenberg, F., 2007. A survey on context-aware systems. International Journal of Ad Hoc and Ubiquitous Computing 2 (4), 263–277.

Baralis, E., Cagliero, L., Cerquitelli, T., Garza, P., Marchetti, M., 2011. Cas-mine: providing personalized services in context-aware applications by means of generalized rules. Knowledge and Information Systems 28 (2), 283–310.

Bardram, J., 14-17 March 2004. Applications of context-aware computing in hospital work - Examples and design principles. In: Proceedings of the Annual ACM Symposium on Applied Computing. New York, NY, USA: ACM Press;, Nicosia, Cyprus, pp. 1574–1579.

Bass, L., Clements, P., Kazman, R., 2003. Software Architecture in Practice, 2nd Edition. Addison-Wesley Professional.

Blake, J. A., Harris, M. A., 2008. The Gene Ontology (GO) project: structured vocabularies for molecular biology and their application to genome and expression analysis. Current Protocols in Bioinformatics 23 (7.2.1-7.2.9), 1472–6947, `http://www.geneontology.org/`.

Bricon-Souf, N., Newman, C. R., 2007. Context awareness in health care: A review. International Journal of Medical Informatics 76 (1), 2–12.

Burgelman, J.-C., Punie, Y., 2006. Close encounters of a different kind: ambient intelligence in Europe. True Vision: The Emergence of Ambient Intelligence, 19–35.

Byun, H., Cheverst, K., 2004. Utilizing context history to provide dynamic adaptations. Applied Artificial Intelligence 18 (6), 533–548.

Carroll, J. J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., Wilkinson, K., May 17–22 2004. Jena: implementing the semantic web recommendations. In: Proceedings of the 13th international conference on World Wide Web, Alternate track papers & posters. New York, NY, USA, pp. 74–83.

Chen, H., 2004. An intelligent broker architecture for pervasive context-aware systems. Ph.D. thesis, University of Maryland, Baltimore County.

Chin, T., 2004. Technology valued, but implementing it into practice is slow. American Medical News, http://www.ama-assn.org/amednews/2004/01/19/bisb0119.htm.

Colpaert, K., Belleghem, S. V., Benoit, D., Steurbaut, K., Turck, F. D., Decruyenaere, J., October 11-14 2009. Has information technology finally been adopted in intensive care units? In: 22nd Annual Congress of the European Society of Intensive Care Medicine. Vienna, Austria, p. 235.

Criel, J., Claeys, L., 2008. A transdisciplinary study design on context-aware applications and environments. a critical view on user participation within calm computing. Observatorio 2 (2), 57–77.

Davies, M. G., Hagen, P. O., 1997. Systematic inflammatory response syndrome. The British Journal of Surgery 84 (7), 920–935.

de Toledo, P., Jimenez, S., del Pozo, F., Roca, J., Alonso, A., Hernandez, C., 2006. Telemedicine experience for chronic care in copd. IEEE Transactions on Information Technology in Biomedicine 10 (3), 567–573.

Dey, A. K., Abowd, G. D., 1-6 April 2000. Towards a better understanding of context and context-awareness. In: Morse, D. R., Dey, A. K. (Eds.), Proceedings of the CHI Workskhop on the What, Who, Where, When and How of Context-Awareness. New York, NY, USA: ACM Press, The Hague, The Netherlands, pp. 304–307.

Everitt, B. S., Skrondal, A., 2010. The Cambridge Dictionary of Statistics, 4th Edition. Cambridge University Press, New Yor, USA.

Fishkin, K., Wang, M., Fishkin, K. P., Wang, M., 2003. A flexible, low-overhead ubiquitous system for medication monitoring. Tech. rep., Intel Research Seattle, Technical Memo IRS-TR-03-011.

Floerkemeier, C., Siegemund, F., 25 October 2003. Improving the effectiveness of medical treatment with pervasive computing technologies. In: Proceedings of the 2nd International Workshop on Ubiquitous Computing for Pervasive Healthcare Applications at International Conference on Ubiquitous Intelligence and Computing. Seattle, Washington, USA.

Fook, V. F. S., Tay, S. C., Jayachandran, M., Biswas, J., Zhang, D., 13-17 March 2006. An ontology-based context model in monitoring and handling agitation behaviour for persons with dementia. In: Proceedings of the 4th

IEEE International Conference on Pervasive Computing and Commununications Workshops (PERCOMW). Washington, DC, USA: IEEE Computer Society;, Pisa, Italy, pp. 560–564.

Gruber, T., 1993. A translation approach to portable ontology specifications. Knowledge Acquisition 5 (2), 199–220.

Gu, T., Pung, H. K., Zhang, D. Q., 2005. A service-oriented middleware for building context-aware services. Journal of Network and Computer Applications (JNCA) 28 (1), 1–18.

Hong, J., Suh, E., Kim, S., 2009a. Context-aware systems: A literature review and classification. Expert Systems with Applications 36 (4), 8509–8522.

Hong, J., Suh, E.-H., Kim, J., Kim, S., 2009b. Context-aware system for proactive personalized service based on context history. Expert Systems with Applications 36 (4), 7448–7457.

Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B., Dean, M., 2004. SWRL: A semantic web rule language combining OWL and RuleML. Tech. rep., http://www.w3.org/Submission/SWRL/.

Hu, B., Hu, B., Wan, J., Dennis, M., Chen, H.-H., Li, L., Zhou, Q., 2010. Ontology-based ubiquitous monitoring and treatment against depression. Wireless Communications & Mobile Computing 10 (10), 1303–1319.

Jahnke, J. H., Bychkov, Y., Dahlem, D., Kawasme, L., 2004. Context-aware information services for health care. In: Proc. of the Workshop on Modeling and Retrieval of Context. pp. 73–84.

Jansen, B., Deklerck, R., November 29 2006. Context aware inactivity recognition for visual fall detection. In: Proceedings of the Pervasive Health Conference and Workshops. Innsbruck, Austria, pp. 1–4.

Klinov, P., June 1-5 2008. Pronto: A non-monotonic probabilistic description logic reasoner. In: Proceedings of the 5th European Semantic Web Conference. Tenerife, Spain, pp. 822–826.

Korhonen, I., Paavilainen, P., Särelä, A., 25 October 2003. Application of ubiquitous computing technologies for support of independent living of the elderly in real life settings. In: Proceedings of the 2nd International Workshop on Ubiquitous Computing for Pervasive Healthcare Applications at International Conference on Ubiquitous Intelligence and Computing. Seattle, Washington, USA.

Kotsiantis, S. B., 2007. Supervised machine learning: A review of classification techniques. Informatica 31 (3), 249–268.

Kotsiantis, S. B., 2013. Decision trees: a recent overview. Artificial Intelligence Review 39 (4), 261–283.

Lukasiewicz, T., 2007. Probabilistic description logics for the semantic web. Tech. rep., Technical University of Wien, Institute for Information Systems, Wien, Austria.

McGuinness, D. L., Harmelen, F. v., February 10 2004. Owl web ontology language overview. Tech. Rep. REC-owl-features-20040210, World Wide Web Consortium, `http://www.w3.org/TR/owl-features/`.

Mihailidis, A., Carmichael, B., Boger, J., Fernie, G., 25 October 2003. An intelligent environment to support aging-in-place, safety, and independence of older adults with dementia. In: Proceedings of the 2nd International Workshop on Ubiquitous Computing for Pervasive Healthcare Applications at International Conference on Ubiquitous Intelligence and Computing. Seattle, Washington, USA.

Mitchell, S., Spiteri, M. D., Bates, J., Coulouris, G., 17-20 September 2000. Context-aware multimedia computing in the intelligent hospital. In: Proceedings of the 9th workshop on ACM SIGOPS European workshop: beyond the PC: new challenges for the operating system. New York, NY, USA: ACM;, Kolding, Denmark, pp. 13–18.

Munoz, M. A., Rodríguez, M., Favela, J., Martinez-Garcia, A. I., González, V. M., 2003. Context-aware mobile communication in hospitals. Computer 36 (9), 38–46.

Nyström, P. O., 1998. The systemic inflammatory response syndrome: definitions and aetiology. Journal of Antimicrobial Chemotherapy 41, 1–7.

O'Connor, M. J., Das, A. K., 2010. A lightweight model for representing and reasoning with temporal information in biomedical ontologies. In: International Conference on Health Informatics (HEALTHINF). Valencia, Spain, pp. 90–97.

Ongenae, F., Bleumers, L., Sulmon, N., Verstraete, M., Jacobs, A., Van Gils, M., Ackaert, A., De Zutter, S., Verhoeve, P., De Turck, F., 26-29 October 2011a. Participatory design of a continuous care ontology: Towards a user-driven ontology engineering methodology. In: Filipe, J., Dietz, J. L. G. (Eds.), Proceedings of the International Conference on Knowledge Engineering and Ontology Development (KEOD). ScitePress Digital Library;, Paris, France, pp. 81–90.

Ongenae, F., De Backere, F., Steurbaut, K., Colpaert, K., Kerckhove, W., Decruyenaere, J., De Turck, F., 2011b. Appendix B: overview of the existing medical and natural language ontologies which can be used to support the trans-lation process. BMC Medical Informatics and Decision Making 10 (3), 4.

Ongenae, F., De Backere, F., Steurbaut, K., Colpaert, K., Kerckhove, W., Decruyenaere, J., De Turck, F., 2011c. Appendix B: overview of the existing medical and natural language ontologies which can be used to support the translation process. BMC Medical Informatics and Decision Making 10 (3), 4.

Ongenae, F., Myny, D., Dhaene, T., Defloor, T., Van Goubergen, D., Verhoeve, P., Decruyenaere, J., De Turck, F., 2011d. An ontology-based nurse call management system (oNCS) with probabilistic priority assessment. BMC Health Services Research 11 (26), 28.

Orwat, C., Graefe, A., Faulwasser, T., 2008. Towards pervasive computing in health care - a literature review. BMC Medical Informatics and Decsion Making 8 (26), 18.

Paganelli, F., Giuli, D., 2011. An ontology-based system for context-aware and configurable services to support home-based continuous care. IEEE Transactions on Information Technology in Biomedicine 15 (2), 324–333.

Prentzas, J., Hatzilygeroudis, I., 2007. Categorizing approaches combining rule-based and case-based reasoning. Expert Systems 24 (2), 97–122.

Prud'hommeaux, E., Seaborne, A., January 15 2008. Sparql query language for rdf. W3C Recommendation REC-rdf-sparql-query-20080115, http://www.w3.org/TR/rdf-sparql-query/.

Quinlan, J. R., 1986. Induction of decision trees. Machine Learning 1 (1), 81–106.

Quinlan, J. R., 1993. C4.5: Programs for Machine Learning. Morgan Kaufmann, San Francisco, CA, USA.

Quinlan, J. R., 1996. Improved use of continuous attributes in c4.5. Journal of Artificial Intelligence Research 4 (1), 77–90.

Rector, A. L., Rogers, J. E., Zanstra, P. E., van der Haring, E., 8-12 November 2003. OpenGALEN: Open source medical terminology and tools. In: Proceedings of the annual American Medical Informatics Association (AMIA) Symposium. American Medical Informatics Association (AMIA);, Washington, DC, USA, p. 982, http://www.opengalen.org/.

Rodríguez, M. D., Tentori, M., Favela, J., Saldaña, D., García, J.-P., 7-8 August 2011. Care: An ontology for representing context of activity-aware healthcare environments. In: Proceedings of the AAAI Workshop on Activity Context Representation: Techniques and Languages. Menlo Park, USA: AAAI Press;, Paris, San Francisco, CA, USA.

Román, M., Hess, C., Cerqueira, R., Campbell, R. H., Nahrstedt, K., 2002. Gaia: A middleware infrastructure to enable active spaces. IEEE Pervasive Computing 1, 74–83.

Rosse, C., Jr, J. L. V. M., 2008. The foundational model of anatomy ontology. In: Burger, A., Davidson, D., Baldock, R. (Eds.), Anatomy Ontologies for Bioinformatics: Principles and Practice. Springer;, London, UK, pp. 59–117.

Russell, S., Norvig, P., 2003. Artificial Intelligence, A Modern Approach, 2nd Edition. Prentice Hall.

Santos, L. O. B. S., Wijnen, R. P., Vink, P., 26-30 November 2007. A service-oriented middleware for context-aware applications. In: Proceedings of the 5th International Workshop on Middleware for Pervasive and Ad hoc Computing. New York, NY, USA: ACM Press;, Newport Beach, Orange County, CA, USA, pp. 37–42.

Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A., Katz, Y., 2007. Pellet: A practical owl-dl reasoner. Journal of Web Semantics 5 (2), 51–53.

Skov, M. B., Hoegh, R. T., 2006. Supporting information access in a hospital ward by a context-aware mobile electronic patient record. Personal and Ubiquitous Computing 10 (4), 205–214.

Stanford, V., 2003. Beam me up, doctor mccoy. IEEE Pervasive Computing 2 (3), 13–18.

Strang, T., Linnhoff-Popien, C., 7 September 2004. A context modeling survey. In: Indulska, J., Roure, D. D. (Eds.), Proceedings of the 6th International Conference on Ubiquitous Computing (UbiComp), Workshop on Advanced Context Modelling, Reasoning and Management. Nottingham, UK, pp. 31–41.

Strobbe, M., Hollez, J., Jans, G. D., Laere, O. V., Nelis, J., Turck, F. D., Dhoedt, B., Demeester, P., Janssens, N., Pollet, T., May 25 2007. Design of casp: an open enabling platform for context aware office and city services. In: Pfeifer, T., Strassner, J., Dobson, S. (Eds.), Proceedings of the 4th International Workshop on Managing Ubiquitous Communications and Services (MUCS 2007). Munich, Germany, pp. 123–142.

Strobbe, M., Laere, O. V., Dhoedt, B., Turck, F. D., Demeester, P., 2012a. Hybrid reasoning technique for improving context-aware applications. Knowledge and Information Systems 31 (3), 581–616.

Strobbe, M., Laere, O. V., Ongenae, F., Dauwe, S., Dhoedt, B., Turck, F. D., Demeester, P., Luyten, K., 2012b. Novel applications integrate location and context information. IEEE PERVASIVE COMPUTING 11 (2), 64–73.

Su, J., Zhang, H., 2006. A fast decision tree learning algorithm. In: Proceedings of the 21st National Conference on Artificial Intelligence. Boston, MA, USA, pp. 500–505.

Suzuki, T., Doi, M., 31 March - 5 April 2001. Lifeminder: an evidence-based wearable healthcare assistant. In: Beaudouin-Lafon, M., Jacob, R. J. K. (Eds.), Proceedings of the ACM Conference on Human Factors in Computing Systems. New York, NY, USA: ACM, Seattle, Washington, USA, pp. 127–128.

Tentori, M., Segura, D., Favela, J., 2009. Chapter VIII: Monitoring hospital patients using ambient displays. In: Olla, P., Tan, J. (Eds.), Mobile Health Solutions for Biomedical Applications, 1st Edition. Vol. 1. Medical Information Science Reference;, Hershey, New York, USA, pp. 143–158.

Tsang, S. L., Clarke, S., 2008. Mining user models for effective adaptation of context-aware applications. International Journal of Security and its Applications 2 (1), 53–62.

Valls, A., Gibert, K., Sánchez, D., , Bateta, M., 2010. Using ontologies for structuring organizational knowledge in home care assistance. Internation Journal of Medical Informatics 79 (5), 370–387.

Varshney, U., 2009. Chapter 11: Context-awareness in healthcare. In: Pervasive Healthcare Computing: EMR/EHR, Wireless and Health Monitoring, 1st Edition. Springer Science + Business Media, LLC;, New York, NY, USA, pp. 231–257.

Verstichel, S., De Poorter, E., De Pauw, T., Becue, P., Volckaert, B., De Turck, F., Moerman, I., Demeester, P., 2010. Distributed ontology-based monitoring on the IBBT WiLab.t infrastructure. In: Proceedings of the 6th International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TridentCom). Berlin, Germany, pp. 509–525.

Witten, I. H., Frank, E., Hall, M., 2011. Data Mining: Practical Machine Learning Tools and Techniques, 3rd Edition. Morgan-Kaufmann.

Xue, W., Pung, H. K., 2012. Chapter 8: Context-Aware middleware for supporting mobile applications and services. In: Kamur, A., Xie, B. (Eds.), Handbook of Mobile Systems Applications and Services, 1st Edition. Vol. 1. CRC Press;, Florida, USA, pp. 269–304.

Yilmaz, O., Erdur, R. C., 2012. iconawa - an intelligent context-aware system. Expert Systems with Applications 39 (3), 2907–2918.

Zhang, D., Yu, Z., Chin, C.-Y., 2005. Context-aware infrastructure for personalized healthcare. Studies in Health Technology and Informatics 117, 154–163.