Representing and executing a Medical Guideline using Prova

Gerhard Kober[1] and Adrian Paschke[2]

[1] Tiani "Spirit" Gmbh, Vienna, Austria
gerhard[DT]kober[AT]tiani-spirit.com
[2] Fraunhofer FOKUS and Freie Universitaet Berlin, Berlin, Germany
adrian[DT]paschke[AT]fokus.fraunhofer.de

Abstract. Patient assessment and finding their critical issues is a crucial aspect of patient care, which is addressed by a medical workflow called the ABCDE approach. However, the ABCDE guidelines are only described in a textual way and hence are not automated. Human errors in practical use might occur, e.g. by persons new into the domain. In this paper, we propose a technical solution to support medical staff during the ABCDE assessment tasks by helping to decide if a patient is critically ill or injured and prioritize particular treatments for better and faster therapy. The solution is built upon a set of semantic rules and automated using the rule-engine "Prova".

Keywords: Medical guideline · Prova · Ruleengine.

1 Introduction

Medical staff in emergency rooms and paramedics in emergency scenes are usually confronted with patients they do not know. This means they are unaware of their current medical status, what has happened to them, and where or whom to ask about details. Emergency cases have a unique characteristic for the persons who need to treat patients - the focus during processing is on high important body functions while having in mind, some body functions are less important for surviving the illness or accident [1]. For example, a person must have a free airway - otherwise, breathing is impossible, and life is at high risk. In medical science, a so-called 'ABCDE' approach (Airway, Breathing, Circulation, Disability, Exposure) has been introduced, which takes care of life-threatening causes, evaluating and treating them step by step [18].

The ABCDE approach is a basis in many training formats (e.g., ERC (European Resuscitation Council), NAEMT (National Association of Emergency Medical Technicians), AHA (American Heart Association)) [8]. For a paramedic in training, the entire ABCDE approach is hard to learn and needs permanent training to incorporate knowledge for quick evaluation about the patient's needs. In daily routine, including all the details when evaluating the scene and the single steps in the ABCDE approach, and take correct decisions for treatment or calling for additional help and finally to lower lethality of patients [6].

The paper describes a technical approach to support the decision and action-taking concerning the ABCDE approach for paramedics in training or emergency physicians. The goal is not only to provide support for the five steps (as ABCDE suggests), but our solution also details every step and supports the decision whether a patient is critically ill and needs treatment within a hospital (e.g., a trauma-center or intensive care unit), or the patient is not critically ill and therefore care at a physician's office is sufficient. This technical support also helps emergency doctors who do not complete the entire ABCDE approach and tend to forget essential points [11].

Personal health-tracking tools and more and more available personal medical data [15] might lead to the idea to take a decision about critical illness already during the emergency call, so that the emergency dispatcher can decide early to send specialists to the emergency scene and already book intensive-care-beds in advance. However, several papers that analyse the ABCDE approach advise paramedics and medical teams to do a best effort in the initial patient assessment and it seems there is no technical approach yet to support this sort of workflow, even if there are still issues in performing the five steps.

In the following section, we describe the ABCDE approach from a practitioner's viewpoint and which technical solutions might be possible to apply to it. In section 3 we describe our approach to deliver an at most flexible solution without the need to change software deployments. In the results-section (4) we evaluate our solution with practical examples and then we discuss it in section 5. Finally, we conclude and mention future work in section 6.

This paper contributes by delivering a technical solution for the currently non-technical ABCDE approach in order to support medical staff in making faster decisions.

2 Related work

2.1 The ABCDE approach

The ABCDE approach is used in patient assessment for finding critical illnesses or injuries and in treatment [20]. This approach was developed by experts in the emergency field and is applicable in any emergency case to any patient group. ABCDE is an abbreviation for Airway, Breathing, Circulation, Disability, and Exposure.

This acronym is used to remind medical staff or paramedics about the order of the checks. They are ordered by importance, and the purpose is to treat first what kills first [23]. Every step has an outcome - either a "everything fine, proceed"-result or a "stop here - apply treatment, take a decision for immediate transport or call for specialist". All these tasks contain more factual information: the particular checks to be done, which treatment to apply, upper and lower limits for vital signs. The approach allows starting from the very beginning, once an issue is fixed. For example, if the airway is blocked, a “stop and treat”-result will occur. Once the paramedic is able to resolve the issue, the paramedic is forced to restart the entire ABCDE approach and find subsequent issues.

The overall result of the entire ABCDE approach is either *critical* or *not critical*. These two impact patient treatment, priority within a treating facility (e.g., hospital, the ambulance car), and the following urgent procedures for medical staff.

From a more generic viewpoint, the ABCDE approach can be considered as a medical workflow. The ABCDE approach displayed in figure 1 (without exceptions) maps to a sequence-workflow-pattern [21]. This is because every

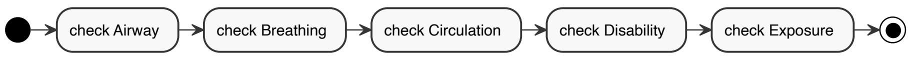

Fig. 1. ABCDE-workflow

step in the ABCDE approach is a single task, but the next one can only proceed if the previous is successfully completed. The start event involves a medical person’s involvement, and the end of the task sequence is the final result. This sequence (workflow) pattern, when taking into account the subsequent tasks, of the entire ABCDE approach maps technically to an ’exclusive-choice-workflow-pattern’ [21]. The expansion is needed because if a medical check (e.g., on the Airway) is not successful, another workflow branch proceeds. Each sub-branch includes manual interventions, instructions, and decisions that need to be taken.

BPMN (Business Process Model and Notation) [5] is a widely used approach to describe Clinical Guidelines defining each step and the different path to follow. In [2] a BPMN model was used to transform guidelines into a formal model called ’labeled event structures (LES)’ for finding conflicts between two guidelines applied at the same time. However, the LES mapping is not appropriate to use it within a rule engine.

From a technical perspective, the decisions taken within the medical workflow are rules [10]. The ”check-result” of any decision-point is depending on multiple parameters. So the rules decide on the path to be followed in the sequence. For example, the ”check Breathing”-decision-point depends on whether the patient breathes normally and on oxygen saturation greater than 95%. Therefore the rule decides for true (=yes) only if the two mentioned functions evaluate to true; otherwise, the rule concludes false (=no).

The entire ABCDE approach is shown in figure 2. Since many publications provide a textual description of the ABCDE approach, these descriptions are still unclear, and they differ in detail, depending on the expert (-group) being involved. However, for our solution, the detail itself is not of importance, since it is just another parameter.

2.2 Medical Rule Engines

Research in clinical/medical decision support systems and the related workflows has a long history. One of the main contributors in the field of clinical decision support is the *Arden Syntax* [17]. Another standard is the “Guideline Interchange Format” (GLIF3), which intention is about guideline-sharing. However, there are many more Clinical Decision Support Systems, but these two are essential because Arden Syntax is productively used, and GLIF3 takes care about sharing guidelines.

The structure of Arden Syntax follows so-called Medical Logic Modules (MLMs), which are collected out of different components. Namely they are “Maintainance”, “Library”, “Knowledge” and “Resources”. All these components together are needed to build an appropriate knowledge base, to provide enough logic for taking clinical decisions [9]. From a technical perspective, a technician always needs to implement the MLMs, since the encoding of these modules is done for programmers and is not fully intuitive for an end-user. Furthermore, the provided examples [19] in the literature show that huge workflows contain a lot of *if- else- clauses*, in order to capture the entire workflow. This is because there is a need to take care about any detail in the conceptualization of the entire decision-path. However, the Arden syntax with the MLM structure is an established tool that is powerful in clinical decision support but seems too complicated for simple workflows like the ABCDE approach and needs an exact definition of the particular workflow and possible outcomes (e.g., there is no mechanism for “negation as a failure” as in logic programming rules).

The approach using medical guidelines, with a focus on sharing, is the *Guideline Interchange Format (GLIF3)* [22]. It was based on Arden Syntax and is meant to be used for exchanging guidelines among different systems. GLIF3 creates flowcharts of medical workflows containing an object-oriented representation for decision tasks of the guideline [3]. There is a user-frontend available for creating the workflows and transforming them to an XML syntax. For the formalization of the guideline expressions itself, Arden Syntax was incorporated into GLIF, but due to incompatibilities, the GLIF3 (version 3) removed Arden Syntax and developed their own language called GELLO (Guideline Expression Language, object-orientated). To allow standardized decision-taking and equal treatment procedures even in different locations, sharing medical guidelines is an essential task. To achieve the goal of reproducible results beyond hospital boundaries, the same workflows and rules need to be applied. To avoid re-implementing the same guidelines several times, the approach of creating it once and sharing seems appropriate. Encoding the guideline to an XML representation for exchanging among systems is standard practice and widely used. This way of exchanging information is also used in RuleML[4].

[4] http://wiki.ruleml.org/index.php/Mission

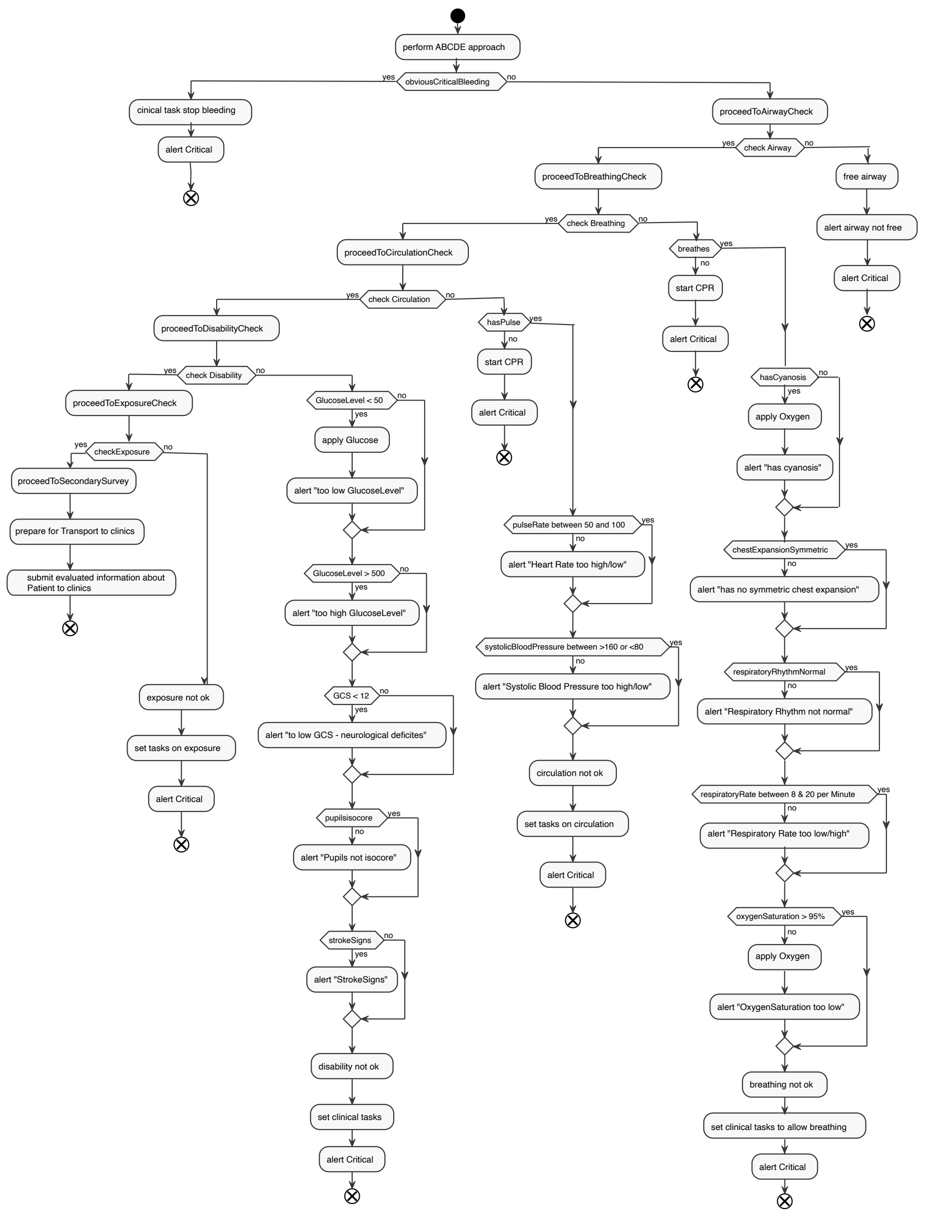

Fig. 2. ABCDE approach

3 Representing the Rules

In this section, we represent the ABCDE workflow as logical rules, defining the overall sequence and the expected outcomes of each rule. Secondly, we describe a proof of concept implementation using Prova-rule-engine. Thirdly, we outline a "hard-coded"-version of the workflow.

The ABCDE workflow is rigorous in the procedure. Therefore, the workflow does not allow any change in the order since the tasks are depending on each other. Furthermore, is it a *must* for the sequence to stop if any task is evaluated as “not normal”. From a practitioner’s perspective, these stops are crucial to deliver treatment to the patient at the right point of time, call for additional help, or prepare rapid transportation.

3.1 Representing the ABCDE approach in formal rules

As a first step in transforming the medical workflow from a textual representation to a technical support solution, there is the need for a formal definition of the applied rules. These rules can be represented as a set of horn-clauses [4]. These clauses only hold if the entire body of the rule holds; otherwise, the rule’s head results in *false*. Furthermore, this formal representation can be used for the implementation task.

Before defining the rules in more detail, it is necessary to mention the following: the over-all-workflow sequence defines the ABCDE workflow by itself. Secondly, each task of the sequence is also a rule that needs evaluation.

We define the check-results as follows (table 1):

Table 1. CheckResults

checkAirway(x) ≡ A	*checkBreathing*(x) ≡ B
checkCirculation(x) ≡ C	*checkDisability*(x) ≡ D
checkExposure(x) ≡ E	

The following rule applies for the particular ABCDE approach.

$$notCritical \leftarrow A \wedge B \wedge C \wedge D \wedge E \tag{1}$$

Using this information, we conclude every check results to be *true* if there is no critical issue in any of the single tasks. Therefore the entire result is *not critical*. If any of the checks resolves to false, the entire formula results in false therefore $\neg notCritical$, which means a *critical* outcome.

For the more detailed checks, some facts, which are essential for analysis, need to be defined. This definition is done on the patient’s "should-be" status; healthy and everything as expected. The patient’s vital signs, which are checked during the entire process and that need to be resolved as *true* are represented in table 2.

With this information available, the next step is in resolving the truth-values for *A, B, C, D, E*. These more detailed probes are the trigger for finding the critical values during the entire process.

For the checkAirway ($\equiv A$), the validation to *airwayIsFree* is done by using the following rule.

$$A \leftarrow checkAirwayFree(x) \tag{2}$$

Table 2. Patient's should-be values

airwayIsFree(yes) $\rightarrow$ true	*breathes*(yes) $\rightarrow$ true
cyanosis(no) $\rightarrow$ true	*chestExpansion*(normal) $\rightarrow$ true
respiratoryRythm(normal) $\rightarrow$ true	*chestExpansionSymmetric*(yes) $\rightarrow$ true
pulseCentral(yes) $\rightarrow$ true	*pulsePeripheral(*yes) $\rightarrow$ true
pupilsIsocore(yes) $\rightarrow$ true	*strokeSigns*(no) $\rightarrow$ true
hasBleeding(no) $\rightarrow$ true	*hasBrokenBone*(no) $\rightarrow$ true
hasAllergies(no) $\rightarrow$ true	*hasPain*(no) $\rightarrow$ true

This means if the airwayIsFree resolves to true, A resolves to true, otherwise, A returns false, and subsequently returns a false for the *notCritical*.

Similar for the *checkBreathing* ($\equiv B$) which concludes to true if, and only if the patient breathes, has no cyanosis, the chest expands symmetric, a normal respiratory rhythm, a respiratory rate is between 8 and 15 per minute, and oxygen saturation of the blood higher than 95%. Formulating this as a rule, give the following representation:

$$\begin{aligned} B \leftarrow breathes(x) \wedge \neg cyanosis(x) \wedge symmetricChestExpansion(x) \\ \wedge respiratoryRhythm(x) \wedge respiratoryRate(x, y, z) \\ \wedge oxygenSaturation(x, y) \end{aligned} \tag{3}$$

If the check Breathing results to *true* the *circulationCheck* ($\equiv C$) is performed. It consists of checking if the patient has a pulse and if yes, the pulse rate and the systolic blood pressure are interesting. When formulating this as a rule representation, we get the following:

$$C \leftarrow pulseCentral(x) \wedge pulseRate(x, y, z) \wedge systolicBloodPressure(x, y, z) \tag{4}$$

The next in the sequence is the *checkDisability*($\equiv D$). It takes care of the blood glucose level, next to the GCS (Glasgow coma scale), if there is a symmetric reaction of the eyes and any stroke signs that are of high importance to take care of. Formulating this needs resolves to:

$$D \leftarrow glucoseLevel(x, y, z) \wedge GCS(x) \wedge pupilsisocore(x) \wedge strokeSigns(x) \tag{5}$$

The last step in the ABCDE approach is the *checkExposure* ($\equiv E$). This check results in yes/no-answers, where the "no" is the needed one, and therefore the true/false mechanism is flipped. For example, the *hasBleeding* resolves to *true* if there is no bleeding, but it is *false* if there is bleeding. The same applies to the other functions.

$$E \leftarrow hasBleeding(x) \wedge hasBrokenBone(x) \wedge hasAllergies(x) \wedge hasPain(x) \tag{6}$$

During the entire process, the solution needs to take care of outcomes. This means, no matter if a result is positive (*notCritical*, or an "Exception" is happening during the ABCDE sequence, this result needs to be communicated to the consumer of the workflow. Thus, we included an additional predicate to send messages by using *Prova* as a rule engine. These messages contain information about the outcome(s), which is dispatched to the initial caller of the Prova service.

3.2 Proof of concept implementation using Prova

Having a formal representation of the rules is an advantage for a proof of concept implementation. For our implementation we are using a rule engine. A rule-engine separates the business logic from program logic [12] in a declarative rule representation. Considering the ABCDE approach as “business logic”, we can easily change or extend the approach and the different checks without the need of changing program code, which takes care of the alerting mechanism itself. Even if different healthcare organizations need to have their individual procedures because of different levels of expertise, the same software stack can be applied by only changing the rule logic. Also, suppose there are new findings in medical research targeting changes in the logic. In that case, an alignment can quickly be done without the need to deliver new software to an organization or an individual.

The procedure in our solution: The rule-engine (in our particular case) Prova[5] [16], receives the message values, processes them and returns the outcome of the rule evaluation.

The Prova-syntax is prolog-style, and allows, on the one hand, to define *facts*, and on the other hand, the *rules* and *goals*. For our use case, we need to take care of all of the three elements. In more detail: From a user perspective, there is a user interface where the user can put all the information about a patient. If the user then hits the ”submit”-button, the entered values are sent (in the form of JSON-RESTful-queries) to a Java servlet. The Java servlet receives the submission and then calls the rule-engine, with the configured logic, and the ”to be evaluated goal” (in here the *notCritical*).

Prova evaluates all the configured rules from section 3.1 and delivers the result back to the servlet. The resulting message from Prova back to the servlet is send using the sendMsg-function of Prova.

The sendMsg-function is a build-in-function of Prova that stops the workflow at the right point of procedure, with a correct error message. For example: if the airway is evaluated as “not free”, the workflow stops and returns that an error in the procedure occurred and needs re-evaluation or treatment of this particular issue.

We define the following *sendMsg*-functions within the rules. In general, in order to get a result returned, if no error occurred during processing, we send a completed message so that the user knows that the check is completed.

$$sendMsg(XID, osgi, ”FHIR”, inform, ”ABCDECheck” - > completed). \tag{7}$$

[5] https://github.com/prova/prova

For any errors there are also messages to be sent back, to provide the user with information about the issue, and what to treat. For example:

$$sendMsg(XID, osgi, "FHIR", inform, "AirwayNotFree" -> AW). \quad (8)$$

The Prova rulebase is configured to take care of the correct sequence of the ABCDE approach and stops if one is not successful. It also takes care of subsequent tasks for alerting. If everything is completed without any occurrences, the result is "completed" - no alerting, and a success message is send.

By implementing the workflow with Prova, i.e. a declarative rule representation, an advantage of our approach is the ease of extension. Any other workflow can be used by simply adding a new Prova-file containing the needed decisions. The generation of Prova files can be done by translating from the RuleML syntax. Due to the RuleML standard, this workflow is applicable to any other rule engine that supports RuleML or can convert from it, as well[6] [13].

3.3 Proof of concept implementation using Java

As an alternative to using a rule engine, the entire workflow can be coded in a Java program with imperative if-then rules following a predefined control flow. This is an approach that seems very obvious and has its legitimacy since, for a developer, it is an easy task to implement a method that covers the rules. However, this approach gets quickly confusing because of many decision points within the code.

In our proof of concept implementation for the hard-coded task, we used a small Java program containing a method, which receives the values from a JSON object and then runs the sequence. We had to take care of the different types that might be included in a JSON request and had to transform them to compare them. Concerning the Prova implementation, we did not face this problem, since Prova already handles types and comparison mechanisms. Furthermore, this approach is not flexible for an organization to operate and maintain, and it is pretty expensive since every change needs to be re-implemented. If deploying the software in different organizations, every customer is forced to use another version. This hard-coded solution becomes cumbersome even if a limit change or an additional parameter needs to be checked.

The code about the implementation can be found on Github[7]. There is the Java implementation as well as the Prova implementation and the rules.

4 Evaluation and Results

For the evaluation, we checked for the correct behavior of both implementation solutions. Since the evaluation focused on the correctness of the rule's results, we required the values submitted to be of the correct type or contained allowed

[6] https://github.com/RuleML/rule-translation-service
[7] https://github.com/gkober/MedicalRule

values. For example, a number-field contained a number, but not a text, and also the "yes/no"-questions had to contain *yes* or *no*. We sent 50 JSON queries containing different allowed values to both solutions and received the same results. A manual check of the target result was done, and the results from the manual checks matched the results from the implementation.

During implementation, we found a significant advantage of the Prova-rule-based version was the ease of extension of the ruleset. Adding the appropriate rules, saving the file, and re-running the development tests was comfortable in relation to implementing the comparison mechanisms in the Java-code-base.

The pure Java implementation hits some limits: so we need to use the correct values for the specific types. For instance, "airwayFree" can not be *yes* or *no* - since Java can not handle it out of the box as boolean values. So we needed to convert to make sure the values are *true* or *false*. For a potential customer, who does not have a technical background, and only wants to implement the support tool for his organization, this hard to understand and eventually too technical. Furthermore, if the implementation takes care of this, we need to cover all possible "positive" results and convert them.

Additionally, *failure processing* gets quite complex, due to the many combinations, if there are multiple "wrong" results for the alerting mechanism.

Another problematic limit is the exchange of the rules. Suppose several locations (organizations) run the processing engine, and all need to ensure the same behavior as the local medical experts need it. In that case, it might get problematic to exchange the rules to all participants simultaneously. Having a standard like RuleML, which allows the interoperability of rules, helps to have a stable code base from a deployment perspective and an easy exchangeable part in the rules.

5 Discussion

The objective of this paper was to find a technical solution to support medical staff during patient evaluation and finding the critically ill/injured people using the ABCDE approach, and deliver the appropriate therapy. This is highly relevant since physicians and paramedics in the field tend to forget essential checks, and patients are missing proper treatment [14][7].

We created a formal description of the rules applied during the ACBDE approach, also taking into account, there are exit rules that ensure stopping the algorithm at the correct point. A proof of concept implementation supports the general rule approach and compares different attempts of implementation. One implementation was done using the Prova-rule engine, while another implementation was done using only Java. Both methods were evaluated in terms of correctness. From our perspective, the more flexible solution is the declarative rule-based version since the exchange or extension of rules is more manageable and not depending on a developer changing the behavior. The idea of using such a tool in the daily routine of paramedics or physicians needs to be evaluated, and a study of acceptance and results needs to be done.

This unique workflow is minimal and very strict in *yes/no*, and has very well described criteria when a patient is assessed as critical or not-critical. It is possible to apply the idea of representing and executing medical workflows in a rule engine for decision support. Using RuleML as rule interchange standard supports a broad field of use, and is not focused too strict on the medical domain such as Arden syntax, or GLIF3, which e.g. might help in finding external reasons for illnesses.

6 Conclusion & Future work

In this work, we used a simple medical workflow called the ABCDE approach, which is well known amongst paramedics and medical staff, to represent it technically for supporting the medical personal during patient evaluation. Therefore, a formal representation of the workflow was done, including "second-level"-checks, to evaluate a workflow to critical or not-critical. This formal representation was then transformed into a proof of concept implementation to check for correctness. By now, the rule engine is relying on parameters that are provided. In the future, it is foreseen to include data queries (e.g., SPARQL-queries) to the rules and not to have the data in the incoming stream. This helps reduce the amount of transferred data if data is not necessary for evaluation.

References

1. The ABCDE Approach, https://www.resus.org.uk/library/abcde-approach, accessed: 2021-07-19
2. Bowles, J., Caminati, M., Cha, S., Mendoza, J.: A framework for automated conflict detection and resolution in medical guidelines. Science of Computer Programming **182**, 42–63 (2019). https://doi.org/https://doi.org/10.1016/j.scico.2019.07.002
3. Boxwala, A., Peleg, M., Tu, S., Ogunyemi, O., Zeng-Treitler, Q., Wang, D., Patel, V., Greenes, R., Shortliffe, E.: Glif3: A representation format for sharable computer-interpretable clinical practice. Journal of Biomedical Informatics - JBI (01 2004)
4. Chandra, A.K., Harel, D.: Horn clause queries and generalizations. The Journal of Logic Programming **2**(1), 1–15 (1985)
5. Chinosi, M., Trombetta, A.: Bpmn: An introduction to the standard. Computer Standards & Interfaces **34**(1), 124–134 (2012)
6. Culemann, U., et al.: Logistische materialvorhaltung der polytraumaversorgung im schockraum aus unfallchirurgischer sicht unter berücksichtigung des status der klinik: Überregionales traumazentrum akh celle (schwerpunktversorger). OP-JOURNAL **36**(01), 41–48 (2020)
7. Fernández-Méndez, F., Otero-Agra, M., Abelairas-Gómez, C., Sáez-Gallego, N.M., Rodríguez-Núñez, A., Barcala-Furelos, R.: ABCDE approach to victims by lifeguards: How do they manage a critical patient? A cross sectional simulation study. PLoS ONE **14**(4), 1–12 (2019). https://doi.org/10.1371/journal.pone.0212080
8. Häske, D., Gliwitzky, B., Münzberg, M.: Notfallmedizin–standardisierte kursformate. Lege artis-Das Magazin zur ärztlichen Weiterbildung **5**(02), 110–116 (2015)

9. Health Level Seven Arden Syntax: The Arden Syntax for Medical Logic Systems Version 2.9 (2013)
10. Lienhard, H., Künzi, U.M.: Workflow and business rules: a common approach. Workflow handbook pp. 129–140 (2005)
11. Olgers, T.J., Dijkstra, R.S., Drost-de Klerck, A.M., ter Maaten, J.C.: The ABCDE primary assessment in the emergency department in medically ill patients: An observational pilot study. Netherlands Journal of Medicine **75**(3), 106–111 (2017)
12. Paschke, A., Dietrich, J., Kuhla, K.: A logic based sla management framework. In: Iswc'05: Proceedings of the semantic web and policy workshop. pp. 68–83. Citeseer (2005)
13. Paschke, A., Könnecke, S.: Ruleml-dmn translator. In: RuleML (Supplement) (2016)
14. Peran, D., Kodet, J., Pekara, J., Mala, L., Truhlar, A., Cmorej, P.C., Lauridsen, K.G., Sari, F., Sykora, R.: ABCDE cognitive aid tool in patient assessment – development and validation in a multicenter pilot simulation study. BMC Emergency Medicine **20**(1), 1–8 (2020). https://doi.org/10.1186/s12873-020-00390-3
15. Piwek, L., Ellis, D.A., Andrews, S., Joinson, A.: The rise of consumer health wearables: Promises and barriers. PLOS Medicine **13**(2), 1–9 (02 2016). https://doi.org/10.1371/journal.pmed.1001953, https://doi.org/10.1371/journal.pmed.1001953
16. Prova rule language,
https://github.com/prova/prova, accessed: 2021-07-19
17. Samwald, M., Fehre, K., de Bruin, J., Adlassnig, K.P.: The Arden Syntax standard for clinical decision support: Experiences and directions. Journal of Biomedical Informatics **45**(4), 711–718 (2012). https://doi.org/10.1016/j.jbi.2012.02.001, http://dx.doi.org/10.1016/j.jbi.2012.02.001
18. Schoeber, N., Linders, M., Binkhorst, M., Draaisma, J., Fuijkschot, J., Morsink, M., Nusmeier, A., Van Riessen, C., Scheffer, G.J., Turner, N., et al.: Healthcare professionals' knowledge of the systematic abcde approach. Resuscitation **155**, S29 (2020)
19. Seitinger, A., Rappelsberger, A., Leitich, H., Binder, M., Adlassnig, K.P.: Executable medical guidelines with arden syntax—applications in dermatology and obstetrics. Artificial Intelligence in Medicine **92**, 71–81 (2018)
20. Thim, T., Krarup, N.H.V., Grove, E.L., Rohde, C.V., Lofgren, B.: Initial assessment and treatment with the Airway, Breathing, Circulation, Disability, Exposure (ABCDE) approach. International Journal of General Medicine **5**, 117–121 (2012). https://doi.org/10.2147/IJGM.S28478
21. Van der Aalst, W.M., Ter Hofstede, A.H., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distributed and Parallel Databases **14**(1), 5–51 (2003). https://doi.org/10.1023/A:1022883727209
22. Wang, D., Peleg, M., Tu, S.W., Boxwala, A.A., Ogunyemi, O., Zeng, Q., Greenes, R.A., Patel, V.L., Shortliffe, E.H.: Design and implementation of the GLIF3 guideline execution engine. Journal of Biomedical Informatics **37**(5), 305–318 (2004). https://doi.org/10.1016/j.jbi.2004.06.002
23. World Health Organization: The ABCDE and SAMPLE History Approach. World Health Organization p. 70 (2018), https://www.who.int/emergencycare/publications/BEC_ABCDE_Approach_2018a.pdf