dHealth 2020 – Biomedical Informatics for Health and Care
G. Schreier et al. (Eds.)

doi:10.3233/SHTI200096

Exploring Methods of Implementing Arden Syntax for CDS Hooks

Jeroen S. de BRUIN[a,1], Andrea RAPPELSBERGER[b], Klaus-Peter ADLASSNIG[b,c] and Jakub GAWRYLKOWICZ[c,d]
[a] *Institute eHealth, Department of Applied Computer Sciences, FH Joanneum University of Applied Sciences GmbH, Graz, Austria*
[b] *Section for Artificial Intelligence and Decision Support, Center for Medical Statistics, Informatics, and Intelligent Systems, Medical University of Vienna, Austria*
[c] *Medexter Healthcare GmbH, Vienna, Austria*
[d] *University of Vienna, Informatics Studies, Vienna, Austria*

Abstract. Background: Specifications for Arden Syntax lack provisions for the standardized access of clinical decision support (CDS) services. The CDS Hooks standard provides such access. Objectives: To extend an ArdenSuite reference implementation of the Arden Syntax by providing a CDS-Hooks-compatible interface. Methods: With the use case Hepaxpert, an Arden-Syntax-based expert system for the interpretation of hepatitis serology test results, a needs analysis was performed to identify changes required in the ArdenSuite reference implementation to support the CDS Hooks API. Arden Syntax language support for CDS Hooks was also assessed. Results: The needs assessment was performed in three phases: hook assessment, hook context definition, and Card definition. For the use case, the ArdenSuite was modified to include a new hook and hook context, which defines the type of CDS service as well its input parameters. Card definitions were created in the ArdenSuite. Examples of Arden Syntax support for the use case are presented for all three phases. Conclusion: Minor changes in the ArdenSuite made it compatible with the CDS Hooks specification.

Keywords. decision support systems, clinical; health information interoperability; hepatitis serology

1. Introduction

The Arden Syntax for Medical Logic Modules, in short Arden Syntax, is a Health Level Seven (HL7) standard for the electronic representation and sharing of computerized health knowledge bases among healthcare personnel, information systems, and institutions [1]. Arden Syntax is meant to be used by medical experts as well as medical knowledge engineers, which is reflected in its design. Despite being a programming language the syntax resembles natural language to a great extent, which makes implementations easier to read, understand, and verify [2].

Despite these benefits, the Arden Syntax suffers from a variety of problems that restrict the ability of Arden Syntax knowledge bases and clinical decision support (CDS) systems to be shared on a large scale. The most frequently reported difficulty is the "curly

[1] Corresponding Author: Drs. Dr. Jeroen S. de Bruin, FH Joanneum, eHealth, Eckertstrasse 30i, 8020 Graz, Austria, E-Mail: jeroen.debruin@fh-joanneum.at.

braces problem" [3,4]. Unlike newer standards such as the HL7 Clinical Document Architecture [5] and HL7 Fast Healthcare Interoperability Resources (FHIR) [6], Arden Syntax has no underlying data or reference information model as part of its specification. Instead, curly braces are used to demarcate institution-specific code segments that interoperate with the institution's underlying health information system. As underlying systems vary from institution to institution, CDS solutions may break when shared between institutions.

Another problem that hinders the wide dissemination and use of Arden Syntax knowledge bases is the lack of an interfacing standard. Again, with newer standards such as FHIR, the method of interfacing is part of the standard [in the case of FHIR the method of interfacing is Representational State Transfer (REST)]. However, such provisions are not available in Arden Syntax and are left to the standard implementer, with varying results [3,7,8]. Furthermore, other initiatives for the improvement of interoperability in healthcare systems, such as Integrating the Healthcare Enterprise (IHE) have addressed the problem of standardized CDS access and interaction in a rudimentary fashion [9].

Curly braces may be described as a problem of semantic interoperability indigenous to the Arden Syntax specification, which can truly be solved only by changing the specification, although other solutions have been proposed [10,11]. The problem of uniformly accessing Arden Syntax implementations, however, is a problem of technical interoperability. It can be resolved by the extension of Arden Syntax reference implementations to include a standardized interfacing mechanism. Ideally, such a mechanism is tailored to CDS, and has been approved or adopted by at least a large part of the healthcare industry. One such standard is CDS Hooks [12], a standard specification for the integration of CDS services in health information systems.

In this paper – which is a continuation of previous work [13,14] – we describe how a reference implementation of the Arden Syntax, the ArdenSuite [8], was extended with the CDS Hooks standard. We illustrate our efforts with the help of a use case known as Hepaxpert [15], an Arden-Syntax-based CDS system for the automated interpretation of hepatitis A, B, and C serology test results. Moreover, we assessed how constructs (objects, data types, etc.) and metadata supported by the Arden Syntax specification could be used to implement support of the CDS Hooks standard in the ArdenSuite, which we illustrate with the use case as well.

2. Methods

2.1. The Arden Syntax and ArdenSuite Reference Implementation

The Arden Syntax is a digital standard for the implementation, management, and sharing of medical knowledge [16,17]. As the Arden Syntax was developed for the representation (and application) of medical knowledge, it embodies several properties that are beneficial for the creation of knowledge bases in a medical domain. These include a program code that resembles natural language, the inclusion of data types that match the needs of medical documentation, and the ability to represent decision support logic independent of the implementation environment [2].

Arden Syntax knowledge bases are constructed in a modular fashion; each module is known as a medical logic module (MLM). MLMs are split into three required categories, each comprising slots that are filled with metadata or medical logic. The maintenance category contains metadata for the identification of the MLM and the institution responsible for its conception and maintenance. The library category provides medical background information and information pertaining to the source of the implemented logic. Electronic medical knowledge and the relationships between its entities are defined in the knowledge category. This category includes the definition of the *data* slot, wherein variables can be declared and assigned values. These include MLM arguments that are passed via the *Argument* array; variables may also be assigned institution-specific *event* definitions by the use of curly braces. These variables are then used in the category's *evoke* slot as trigger definitions, for which an MLM is automatically triggered.

We used the ArdenSuite designed by Medexter Healthcare for implementing the knowledge base of the application [8]. The ArdenSuite is a commercial Java-based CDS technology platform that comprises an integrated development and test environment for the creation and compilation of MLMs, as well as a server for the execution of compiled MLMs via REST interfaces. Several connectors and extensions for the server provide access to external data sources, such as SQL databases or FHIR servers.

2.2. CDS Hooks

CDS Hooks [12] is a specification for the standardized integration of CDS services in healthcare systems. The standard describes the (RESTful) APIs and interactions between CDS clients and CDS services, whereby data is exchanged using JSON structures and transmitted through HTTPS channels.

As the name suggests, the CDS Hooks standard specifies how a hook-based design pattern should be implemented to discover and invoke CDS services. In such a pattern, the behavior of software can be changed or extended by providing a code that handles intercepted function calls, messages, or events. The code is known as a hook.

The CDS Hooks specification defines two primary APIs, namely Discovery and Service. The Discovery service allows for the runtime discovery of CDS services through an HTTP GET request over a REST endpoint named */cds-services.* An array of CDS Service objects are returned in response to this call. Each object (if any) describes the available services with at least a hook, a description, and the id portion of the service endpoint URL under which the service is available. To call a service endpoint */cds-services/{id}*, the client needs to specify at least the hook, a hook instance, and the service context. An array of Card objects are returned in response.

For CDS services to be compatible with the CDS Hooks specification, two basic components need to be implemented: Hooks and Cards. A Hook is a denominator for common use cases in medical workflows that describes an activity being performed by the CDS Client in this workflow. Some predefined hooks exist at the present time, but the set is basically open. Anyone may define new hooks and propose these to the community. At a specification level, each hook defines a hook context that contains contextual information (parameters) to be provided to the CDS Service. Each CDS Service advertises the hooks it supports and the prefetch data (information optionally needed by the CDS Service) it requires.

Cards serve as the return values of a CDS Service that present the user with information (informational cards), suggestions for the user to accept or decline (suggestion cards), or links to an app with which the CDS Client can further interact with the system (app link card). Although suggestion cards and app link cards contain additional attributes, all card types have a set of required attributes in common, i.e., a summary message for display to the user, an indicator that conveys the urgency or importance of the card's contents, and the source of the information displayed on the card [12].

2.3. Hepaxpert

Hepaxpert [15] is an expert system developed cooperatively by clinical hepatologists and health IT specialists for the interpretation of serology test results for hepatitis A, B, and C. For each of the included serology test results, the application provides a classification using one of four classes: positive, negative, borderline, or not tested. Depending on the overall combination of test results, the user is provided with different interpretive texts. These interpretive texts cover all combinations of results, including frequent, rare, as well as inconsistent combinations.

3. Results

Based on the CDS Hooks specification and the Hepaxpert implementation, a needs analysis was performed to model a situation wherein Hepaxpert and the ArdenSuite support the CDS Hooks API. Figure 1 shows the resulting workflow in which the Hepaxpert service endpoint is called. Comparing the target state analysis with the current situation, the modification process of the existing Hepaxpert CDS Service deployed on the ArdenSuite may be divided into three consecutive phases: hook assessment, hook context definition, and Card definition.

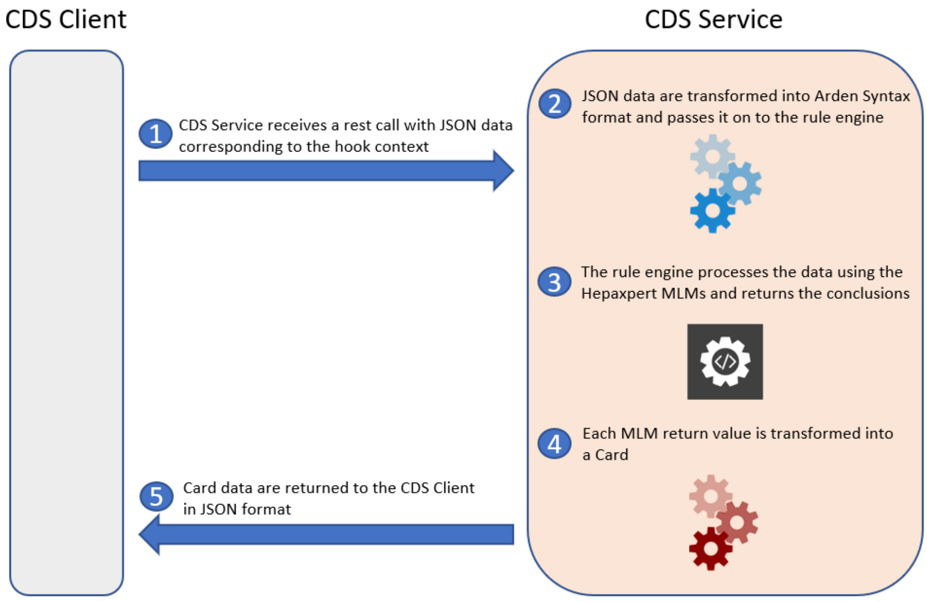
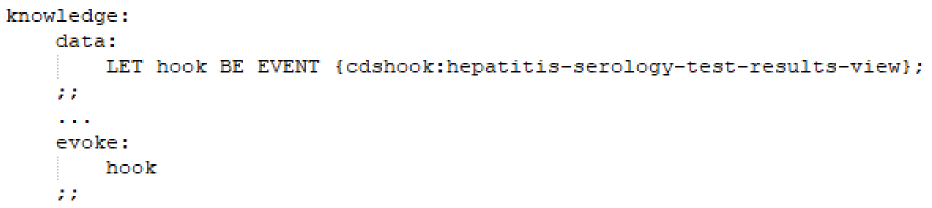

CDS Client
CDS Service
1 CDS Service receives a rest call with JSON data corresponding to the hook context
2 JSON data are transformed into Arden Syntax format and passes it on to the rule engine
3 The rule engine processes the data using the Hepaxpert MLMs and returns the conclusions
4 Each MLM return value is transformed into a Card
5 Card data are returned to the CDS Client in JSON format

Figure 1. Modeled workflow of the Hepaxpert service call after implementation of CDS Hooks API.

3.1. Hook Assessment

Hook assessment is performed to determine whether the collection of currently available hooks comprises one that fits the CDS Service, or whether a new hook needs to be defined. In our use case, no available hook was able to fit our needs. We had to define a new hook named the hepatitis-serology-test-results-view, which required a modification of the REST-API of the ArdenSuite implementation.

The searching and automated configuration of hooks, or the generation of new hooks, could be supported by the ArdenSuite through the Arden Syntax event mechanism. A CDS Hook call can be perceived as an MLM trigger if that MLM is associated with that particular hook and, as such, the ArdenSuite can be extended to handle a hook event. Consider the use case example illustrated in Figure 2. Owing to curly braces, the event definition is left to the reference implementation, i.e., ArdenSuite. As such, the MLM compiler may perform a CDS Hook lookup as a response to the event statement. If available, it can automatically configure the MLM service to be triggered by the defined hook, or generate an error if the hook does not exist.



```
knowledge:
    data:
        LET hook BE EVENT {cdshook:hepatitis-serology-test-results-view};
    ;;
    ...
    evoke:
        hook
    ;;
```

Figure 2. Hook definition in Arden Syntax using an event statement in the data slot and a simple trigger statement in the evoke slot.

3.2. Hook Context Definition

The input parameters for the CDS Service are specified in the hook context definition phase. In case an existing hook was available for the CDS Service, a context definition should also exist, in which case it can automatically be adopted. If a new hook was defined, such as with our use case, this definition needs to be created as well. Table 1 shows the hook context definition for Hepaxpert.

As each MLM in the ArdenSuite is deployed as a web service, program arguments are passed on to the MLMs using a custom JSON format, which had to be refactored in the CDS Hooks format.

Using a standardized format for hook context definition, the ArdenSuite reference implementation could automatically generate or refactor Arden Syntax code. Furthermore, using the names and sequence of variables in a hook context definition, the ArdenSuite could refactor code and assign MLM arguments to automatically generated variables corresponding to context definition names. An example for our use case is shown in Figure 3.

The reverse scenario – in which a hook context definition would be generated from an argument assignment – is only partially possible, because only field names could be determined automatically. As Arden Syntax is a weakly typed language that only supports rudimentary type checking, variable types cannot be determined in compile time. Furthermore, optionality and parameter descriptions also cannot be determined automatically.

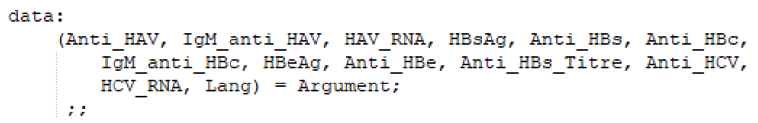
Table 1. Definition of the hepatitis-serology-test-results-view hook. The Field column specifies the name of the variable, the Optionality column states whether the variable must be specified or is optional, the Type column specifies the variable type, and the Description column provides a description of the variable to improve service understanding.

Field	Optionality	Type	Description
Anti_HAV	REQUIRED	Integer	Result of the test for antibodies to hepatitis A
IgM_anti_HAV	REQUIRED	Integer	Result of the test for IgM antibodies to hepatitis A
HAV_RNA	REQUIRED	Integer	Result of the test for hepatitis A virus in stool
HBsAg	REQUIRED	Integer	Result of the test for the surface antigen to hepatitis B
Anti_HBs	REQUIRED	Integer	Result of the test for the surface antibody to hepatitis B
Anti_HBc	REQUIRED	Integer	Result of the test for antibodies to the core antigen
IgM_anti_HBc	REQUIRED	Integer	Result of the test for IgM antibodies to the core antigen
HBeAg	REQUIRED	Integer	Result of the test for the hepatitis B envelope antigen
Anti_HBe	REQUIRED	Integer	Result of the test for the hepatitis B envelope antibody
Anti_HBs_Titre	REQUIRED	Integer	Quantitative measure of antibodies to the antigen of hepatitis B
Anti_HCV	REQUIRED	Integer	Result of the test for antibodies to hepatitis C
HCV_RNA	REQUIRED	Integer	Result of the test for the hepatitis C antigen
Lang	OPTIONAL	String	Contains a two-letter language code.

3.3. Card Definition

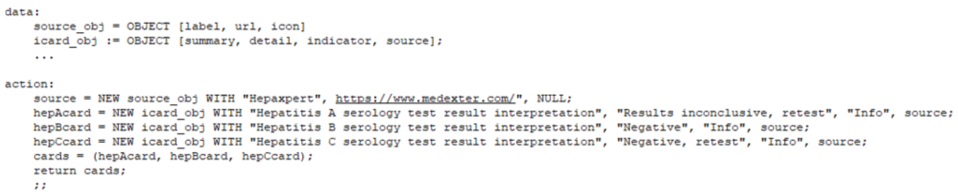
MLM outputs are analyzed in this final stage, and a Card is generated for each discrete output. With respect to our use case, Hepaxpert is a CDS service that provides three distinct interpretations (hepatitis A, B, and C). Thus, three Card definitions are required. These were all of the "information card" type because the MLM results were only of informational value, with no further need for suggestions or interaction. Transformation of these was partially implemented in Arden Syntax, and as such required only a minor modification of the ArdenSuite implementation.

As the data formats for the different Card types are defined by the CDS Hooks standard, these can be implemented using Arden Syntax data types grouped in objects. The only modification of the ArdenSuite implementation would then be data type and availability checking. Figure 4 shows the Arden Syntax object definition for the set of required attributes for an information card and their use in the action slot to create a list of card objects, such as Hepaxpert return results.



```
data:
    (Anti_HAV, IgM_anti_HAV, HAV_RNA, HBsAg, Anti_HBs, Anti_HBc,
        IgM_anti_HBc, HBeAg, Anti_HBe, Anti_HBs_Titre, Anti_HCV,
        HCV_RNA, Lang) = Argument;
    ;;
```

Figure 3. Assignment of MLM arguments to variables named consistently with hook context definitions for the hepatitis-serology-test-results-view hook.



```
data:
    source_obj = OBJECT [label, url, icon]
    icard_obj := OBJECT [summary, detail, indicator, source];
    ...

action:
    source = NEW source_obj WITH "Hepaxpert", https://www.medexter.com/", NULL;
    hepAcard = NEW icard_obj WITH "Hepatitis A serology test result interpretation", "Results inconclusive, retest", "Info", source;
    hepBcard = NEW icard_obj WITH "Hepatitis B serology test result interpretation", "Negative", "Info", source;
    hepCcard = NEW icard_obj WITH "Hepatitis C serology test result interpretation", "Negative, retest", "Info", source;
    cards = (hepAcard, hepBcard, hepCcard);
    return cards;
    ;;
```

Figure 4. Arden Syntax object definition and example assignment for information cards in the Hepaxpert CDS service.

4. Discussion

In the present report we discuss the extension of a reference implementation of the Arden Syntax, a standard for the definition of medical knowledge bases. The extension is CDS Hooks, which is a standard for the definition of CDS service interfaces. Medical knowledge in CDS applications can be uniformly defined and accessed by combining the two standards. This should result in higher-quality software that is widely accessible to a variety of clients.

This report is a continuation of previously published work, wherein we researched the implementation requirements of a CDS Hooks interface for the ArdenSuite [14] and discussed a subsequent prototype implementation [13]. The current work further refines this implementation for a specific use case, namely Hepaxpert. Furthermore, it assesses how CDS Hooks support could be assisted by Arden Syntax language constructs.

For the extension of the Hepaxpert use case to include a CDS Hooks interface, we first needed to define a new hook, which we tailored to the needs of Hepaxpert. Hepaxpert is the only application that was considered for this work. An adaptation for future versions of the hook to fit similar applications is possible and can be achieved with the community-based feedback mechanisms that CDS Hooks puts in place.

Once the new hook had been created, its associated metadata and context data scheme had to be defined. In this case we defined all hepatitis serology test results as required parameters for the CDS Hooks interface, because their encoding also includes the option of “not tested”. The language data element was kept optional because all CDS Clients are not required to select language.

Although we described how an MLM can be rendered compatible with CDS Hooks, not every MLM needs to be compatible. Rather, one CDS Service may consist of a collection of MLMs that cooperate to provide the decision for a specific hook. As such, only the primary MLM should be configured as a CDS Hooks Service, whilst all supporting MLMs would not need modification.

Although we discussed a use case for a specific reference implementation, the methods for achieving support for CDS Hooks in Arden Syntax are generalizable to other reference implementations, as they are supported by the Arden Syntax language itself. The mechanisms for Hook definitions as events involve curly braces, which need to be implemented by a reference implementation in any case. Hook context definition is an extension of data conversion to fill the *Argument* array, and Card definitions can be created by using Arden Syntax object definitions.

To the authors’ knowledge, no previous attempt has been made to combine Arden Syntax and CDS Hooks. Given that CDS Hooks is a foundational standard for SMART on FHIR, in most cases the standard has been implemented in combination with FHIR servers and profiles [18,19]. Nonetheless, as it is an open standard not dependent on underlying standards or representations, it could be combined with Arden Syntax without fundamental compatibility issues.

References

[1] Health Level Seven International, *Health Level Seven Arden Syntax for Medical Logic Systems, Version 2.10, November 2014*, Health Level Seven International, Ann Arbor, 2014. Available from: https://www.hl7.org/implement/standards/product_brief.cfm?product_id=372.

[2] M. Samwald, K. Fehre, J. de Bruin, K.-P. Adlassnig, The Arden Syntax standard for clinical decision support: experiences and directions, *Journal of Biomedical Informatics* **45**(4) (2012), 711–718. https://doi.org/10.1016/j.jbi.2012.02.001.

[3] H.C. Karadimas, C. Chailloleau, F. Hemery, J. Simonnet, E. Lepage, Arden/J: An architecture for MLM execution on the Java platform. *Journal of the American Medical Informatics Association* **9**(4) (2002), 359–368. https://doi.org/10.1197/jamia.M0985.

[4] J. Choi, S. Bakken, Y.A. Lussier, E.A. Mendonça, Improving the human readability of Arden Syntax medical logic modules using a concept-oriented terminology and object-oriented programming expressions, *Computers Informatics Nursing* **24**(4) (2006), 220–225. https://doi.org/10.1097/00024665-200607000-00009.

[5] Health Level Seven International, *Clinical Document Architecture, Release 2, 2007*, Health Level Seven International, Ann Arbor, 2007. Available from: https://www.hl7.org/implement/standards/product_brief.cfm?product_id=7.

[6] Index – FHIR v4.0.1, https://www.hl7.org/fhir/, last access: 21.3.2020.

[7] M. Gietzelt, U. Goltz, D. Grunwald, M. Lochau, M. Marschollek, B. Song, K.-H. Wolf, ARDEN2BYTECODE: A one-pass Arden Syntax compiler for service-oriented decision support systems based on the OSGi platform, *Computer Methods and Programs in Biomedicine* **106**(2) (2012), 114–125. https://doi.org/10.1016/j.cmpb.2011.11.003.

[8] Medexter Healthcare – ArdenSuite CDS Platform, https://www.medexter.com/products-and-services/ardensuite, last access: 21.3.2020.

[9] Clinical Decision Support – IHE Wiki, https://wiki.ihe.net/index.php/Clinical_Decision_Support#1._Proposed_Profile:_Clinical_Decision_Support, last access: 29.1.2020.

[10] M. Sordo, A.A. Boxwala, O. Ogunyemi, R.A. Greenes, *Description and Status Update on GELLO: A Proposed Standardized Object-Oriented Expression Language for Clinical Decision Support*, in: M. Fieschi, E. Coiera, Y.-C.J. Li (Eds.) MEDINFO 2004, Proceedings of the 11th World Congress on Medical Informatics, Studies in Health Technology and Informatics 107(Pt. 1), IOS Press, Amsterdam, 2004. pp. 164-168. https://doi.org/10.3233/978-1-60750-949-3-164.

[11] R.A. Jenders, R. Corman, B. Dasgupta, Making the standard more standard: a data and query model for knowledge representation in the Arden syntax, *AMIA Annual Symposium Proceedings* **2003** (2003), 323–330.

[12] CDS Hooks, https://cds-hooks.org/, last access: 21.3.2020.

[13] M. Spineth, A. Rappelsberger, K.-P. Adlassnig, *Implementing CDS Hooks Communication in an Arden-Syntax-Based Clinical Decision Support Platform*, in: J. Mantas, Z. Sonicki, M. Crişan-Vida, K. Fišter, M. Hägglund, A. Kolokathi, M. Hercigonja-Szekeres (Eds.) Decision Support Systems and Education – Help and Support in Healthcare, Studies in Health Technology and Informatics 255, IOS Press, Amsterdam, 2018. pp. 165-169. https://doi.org/10.3233/978-1-61499-921-8-165.

[14] M. Spineth, *Interoperability with a Clinical Decision Support Rule Engine,* Master Thesis, University of Applied Sciences Technikum Wien, Vienna, 2018.

[15] C. Chizzali-Bonfadin, K.-P. Adlassnig, M. Kreihsl, A. Hatvan, W. Horak, A WWW-accessible knowledge base for the interpretation of hepatitis serologic tests, *International Journal of Medical Informatics* **47**(1-2) (1997), 57–60. https://doi.org/10.1016/S1386-5056(97)00092-0.

[16] G. Hripcsak, Writing Arden Syntax medical logic modules, *Computers in Biology and Medicine* **24**(5) (1994), 331–363. https://doi.org/10.1016/0010-4825(94)90002-7.

[17] K.-P. Adlassnig, P. Haug, R.A. Jenders, Arden Syntax: Then, now, and in the future, *Artificial Intelligence in Medicine* **92** (2018), 1–6. https://doi.org/10.1016/j.artmed.2018.09.001.

[18] K.D. Mandl, I.S. Kohane, No small change for the health information economy, *New England Journal of Medicine* **360** (2009), 1278–1281. https://doi.org/10.1056/NEJMp0900411.

[19] SMART Health IT – Connecting health system data to innovators' apps, https://smarthealthit.org/, last access: 21.3.2020.