Integrated Medical Information and Decision-Support System development based on shared metamodel definition

Aliaksandr Kurachkin
faculty of radiophysics
and computer technologies
Belarusian State University
Minsk, Belarus
alex.v.kurochkin@gmail.com

Vasili Sadau
faculty of radiophysics
and computer technologies
Belarusian State University
Minsk, Belarus
sadov@bsu.by

Aliaksandr Halavatyi
faculty of radiophysics
and computer technologies
Belarusian State University
Minsk, Belarus
alex.halavatyi@gmail.com

***Abstract*—Developing integrated knowledge-based decision-support systems with persistent storage for medical use usually requires creating multiple program modules with a non-generic implementation that includes a lot of duplicate and non-generalized components for describing the dataset the system is designed to operate on. As a solution, this paper proposes a semantic metamodel definition for medical information systems based on extensible entity and attribute descriptor format, along with an integration framework that enables projecting external data to common format, simplifies data access by generating schema definitions and APIs for accessing persistant storage, and eliminates the need for manual user interface development by procedurally generating form-based and list-based views.**

***Keywords*—decision-support systems, medical information systems, intelligent diagnosis, expert systems**

I. Introduction

Medical information systems are an integral part of modern healthcare all around the globe. Generally, any kind of information system operating on patient or medical research data can be considered a medical information system; however, there are several problems specific to these kinds of systems: patient data handling, predictive and intelligent diagnosis assistance, and system integration [1].

Working with patient data generally requires isolated self-hosted database solutions with strong security measures to provide confidentiality, since healthcare data is considered sensitive and private. On the other hand, data sources themselves are generally numerous, loosely connected, non-standardized, denormalized and weakly structured, up to the point where it is possible to have duplicate or even conflicting information regarding the same patient.

Predictive and intelligent diagnosis assistance refers to the problem of using various decision support algorithms and predictive models. These methods are usually based either on rules and strict expect knowledge formalization, or on data analysis and supervised machine learning algorithms. Either way, both kinds of systems require a uniform way to access various kinds of data available regarding a specific type of medical research or specific pathology to evaluate performance relative to existing data, and to train the model in case of supervised machine learning.

Medical information system integration refers to the problem of adding new types of decision support models and handling data source changes in a way that is transparent, while retaining the simplicity of data access and generating predictions by medical staff using an integrated user interface. Ideally, adding a new kind of predictive model should be as simple as implementing a set of contracts within a predefined framework, in such a way that simple descriptive representation of the system is sufficient to automatically connect to existing data sources, provide necessary data access for predictive and intelligent diagnosis assistance, and present a user interface that can be used to both access relevant data and generate predictions based on it.

This paper proposes an integration framework for medical information systems based on descriptive semantic metamodel definition, consisting of 4 main parts: metamodel definition format itself, formal projection definition for adapting existing data sources to metamodel-compliant format, DDL generator for adapting metamodel definition to persistent data store and generating basic data access API, and user interface generation algorithm that can be used to autogenerate form-based and list-based views for data access based on metamodel. The framework itself provides a set of extension points at various stages: bootstrap stage to hook into initialization lifecycle events and register metamodel definitions nad projections, data link stage to modify the process of accessing a specific persistent storage, interface generation stage to implement custom user interface logic,

and decision-support algorithm initialization stage to add custom predictive or knowledge-based models and integrate them with user interface.

II. Metamodel definition

Data model usually refers to definitions basic entities and attributes present in a set of data. For example, in medical information system, an entity may refer to general patient data, and consist of attributes such as name, sex, birthdate, address of residence, etc. In turn, a metamodel refers to metadata definitions that can be used to define data models, i.e., data model allows to describe the dataset with specific entity and attribute definitions, and metamodel allows to describe data models using generalized descriptors [2], [3]. Semantic metamodel is a type of metamodel that is used to describe knowledge base schemas in terms of semantic concepts and relationships [4].

Proposed metamodel format is based on attribute descriptors that, in turn, are grouped together under an entity descriptor.

Attribute descriptor contains the following information:

- attribute identifier – string-based identifier that must conform to variable identifier rules (should consist of alphanumeric characters and underscores, and should not start with numeric character)
- attribute name – human-friendly string name for this attribute, suitable for use as user interface field label
- attribute description – optional string value that contains extended description of a specific attribute, suitable for use as user interface field hint or description
- attribute type and attribute metadata – one of the predefined types for this specific attribute and additional type information
- visibility flag – a flag indicating whether this particular attribute is present on form representations and is able to be edited by user

The following types and metadata points are supported:

- string – string value, e.g., name, diagnosis, etc.
  - single-line or multi-line
  - validation regular expression
- number – floating-point or integer numeric value, e.g., blood glucose level, CT item voxel density, etc.
  - floating point precision
  - minimum value
  - maximum value
- boolean – a simple true/false value, e.g., presence or absence of a specific marker
  - descriptions for true and false values
  - preferred display style – as checkbox or as two radio buttons
- single-value categorical – a value defined as a single choice of multiple options, where each option is represented by string-based value and optional identifier, e.g., type of stroke, sex, etc.
  - options defined as string values or id & value tuples
  - preferred display style – as single-value drop-down or as radio button set
- multiple values categorical – a value defined as zero or more choices of multiple options, where each option is represented by string-based value and optional identifier, e.g., ASPECTS scale visible changes, or multiple related markers
  - options defined as string values or id & value tuples
  - preferred display style – as multiple-value drop-down or as checkbox set
- date, time or date with time – a value represented as a UNIX timestamp
  - minimum date
  - maximum date
  - minimum time
  - maximum time
- attachment – an attribute representing a file with unspecified format
  - required extension
  - maximum file size
- single reference – an attribute representing reference to another entity, which is mapped to a multiple reference from the other entity for one-to-many relationship
- multiple reference – an attribute representing a collection of references to another entity, which can be mapped either to a single reference for many-to-one relationship, or to multiple references from the other entity for many-to-many relationship

Attribute metadata definition is extensible, i.e., the set of properties described above can also include any non-standard definitions that can be later handled using various framework extension points. For example, it is possible to provide additional "type" metadata for string fields for implementing complex validation scenarios like e-mail validation, and then add appropriate validation handler that is able to parse required metadata and provide necessary enhancements at runtime.

Entity descriptors include:

- entity identifier – string-based identifier that must conform to variable identifier rules
- entity name – human-friendly string name for this entity
- entity description – extended description of the purpose and use cases for specific entity type
- mutability – an integer value indicating preferred handling for attribute modifications: 0-mutability

entities are accessed and modified in-place, while entities with mutability of $n > 0$ are modified by adding an entry to modification log, up to n entries, and accessed by reading latest version
- visibility flag – a flag indicating whether this particular entity type is visible as a separate entity on user interface – this can be useful to hide utility entities like many-to-many link tables when mapping to relational databases
- attribute set – a collection of attribute descriptors for this particular entity

An example of entity definition is as follows:

```
{
  "identifier": "OctScan",
  "name": "Optical Coherent Tomography
      scan protocol",
  "description": "OCT scan protocol,
      containing results from parsed
      tomography scan report and values
      from analyzing regions of image
      obtained by OCT scanner.",
  "mutability": 0,
  "visible": true,
  "attributes": [
    {
      "id": "eye",
      "name": "Eye",
      "description": "Which eye is
          examined during the scan",
      "type": "boolean",
      "typeMetadata": {
        "trueDescription": "OS",
        "falseDescription": "OD",
        "displayStyle": "radio"
      }
    },
    {
      "id": "octTemp",
      "name": "Temporal OCT",
      "description": "Retinal thickness
          in temporal side, as measured
          by OCT",
      "type": "number",
      "typeMetadata": {
        "precision": 3
        "min": 0
      }
    },
    {
      "id": "patient",
      "name": "Patient",
      "description": "Patient ref",
      "type": "ref",
      "typeMetadata": {
        "referenceType": "Patient"
      }
    }
  ]
}
```

Entity metadata definition is also extensible and can include any number of additional properties; various extension points throughout the framework can be used to access these properties and implement custom logic. Metamodel is represented as a set of entity descriptors available in the system.

For the purposes of universal serialization and usage in JavaScript language, the proposed metadata format is implemented as YAML or JSON document. Because of the extensible nature of YAML representation, it is also possible to define additional attributes that can be handled in extension points.

Metamodel can be passed to the framework as definition file in JSON or YAML format, or passed directly to the framework runtime during the bootstrap stage. Existing data sources may be adapted to common format by projecting individual data points to common metamodel format [5].

III. Adapting metamodel definition to database schema definition

Metamodel entity and attribute descriptors are designed to serve a storage-agnostic way of defining application schema. At the same time, it is possible to create database-specific adapters that convert metamodel-based definitions to compatible database schema definitions, and define an appropriate data access layer abstractions for specific platform.

Primary usage scenario of proposed metamodel is organizing access to centralized data storage, as explained earlier, that uses graph schema as a source. As such, an adapter for Neo4j graph database is implemented [5].

Since Neo4j database is schema-optional, it is not necessary to create schemas prior to manipulating actual data. However, metamodel can be used to translate API requests to database queries. For example, for OctScan entity, request for retireval of entity list can be triggered by API call, and metamodel definition can be used to translate the request to appropriate Neo4j Cypher query like this:

```
MATCH (x:OctScan)
RETURN (x.eye, x.octTemp)
```

The infromation about appropriate fields is taken directly from metamodel. Moreover, this approach allows to enable query support for any field, where own field constraints are translated to field MATCH clauses, and reference field constraints are translated to vertex-edge MATCH clauses. For example, retrieval of OctScan for specific patient with id 42 can be translated to the following request:

```
MATCH (x:OctScan)<--(p:Patient {id: 42})
RETURN (x.eye, x.octTemp)
```

In order to facilitate access to data storage itself, API endpoints can be used. Typical CRUD endpoints for data access can also be generated automatically based on metamodel definition. Server-side API route registration

and handling is implemented using express framework for Node.JS runtime.

As mentioned earlier, metamodel can be adapted to other types of persistant storage solutions, including knowledge bases, using information in metamodel as metaknowledge.

The contracts of data access layer are abstract and should be replacable with a suitable driver implementation. However, since some of the more exotic database functions cannot (and should not) be abstracted away, API also provides an extension point for accessing native underlaying database connection, while expecting "external" calling code to be transformed to the format compliant with metamodel definitions.

Besides working as API data access proxy and database mapping layer, server-side APIs also call any validation rule that are defined for the attributes in metadata. Validation API is also extendable, which means it's possible to register custom validators that would be triggered based on specific information present in attribute metadata.

IV. User interface generation

Metamodel contains sufficient information to automatically generate a suitable form-based user interface for creating and modifying a single entity, and create paged list views for working with multiple entities [3].

In order to generate form interface, each attribute of a specific entity is mapped to a specific form control. For example, string fields are represented by single-line or multi-line input fields, depending on field metadata, while categorical fields are represented using radio buttons, checkboxes, etc.

References are represented as a special control type that can be used to add refernce to existing entity, create new entity in-place, or remove reference. Multiple references are represented as inline lists.

List interface uses a simple table representation, with columns corresponding to individual metamodel attributes. Column order and visibility can also be adjusted. Columns of most types also usually support custom sorting.

Generated user interface is automatically connected to API data access points for data retrieval and modifications. Using auto-generated interface allows to skip UI development entirely and integrate generated interface directly, and is also guaranteed to correspond to metamodel definition, thus making it much less error-prone. Custom extension points exist that allow to modify and create new rules for mapping attributes to specific control types.

Validation rules defined for each attribute can also be duplicated on client side – this way, validation error and hint appears as soon as the user finishes editing the field or attempts to submit the form. It should be noted that client-side validation does not replace server-side validation completely, since it would still be possible to submit incorrect data by directly accessing the API, so it's used only to enhance user experience.

V. Conclusion

Creating medical information systems and integrating them into existing infrastructure can be greatly simplified by using proposed metamodel-first approach. Using a single metamodel definition across data access APIs, persistent storage schema definitions and user interface generation can greatly simplify rapid prototyping of various medical information systems, as well as supplement the integration of new and existing decision-making models and knowledge-based solutions. The extensibility of the framework allows to adapt it to various types of medical diagnosis, while unified projected data representation greatly enhances the capabilities for interoperability with knowledge sources and allows to create training and validation datasets for various purposes.

References

[1] B. S. Abu-Nasser. "Medical Expert System Survey," International Journal of Engineering and Information Systems (IJEAIS), vol. 1, iss. 7, pp. 218–224, 2017

[2] C. Gonzalez-Perez, B. Henderson-Sellers, "Metamodelling for Software Engineering," Wiley, 219 p., 2008.

[3] A. Kurochkin. "Integrating medical data management and decision-making systems with common metamodel," International Journal of Open Information Technologies, vol. 8, no. 12, pp. 49–53, 2020.

[4] T. Tokuda, Y. Kiyoki, H. Jaakkola, N. Yoshida, "24. Information Modelling and Knowledge Bases XXV (Frontiers in Artificial Intelligence and Applications)", IOS Press, 336 p., 2014.

[5] A. Kurachkin, V. Sadau. Agregatsiya i indeksirovanie neskol'kikh istochnikov dannykh na osnove grafovoi modeli v bazakh dannykh meditsinskikh ekspertnykh sistem [Aggregation and indexing of multiple data sources based on a graph model in databases of medical expert systems]. *Informatika [Informatics]*, 2020. pp. 25-35

Разработка интегрированных медицинских информационных систем поддержки принятия решений на основе общей метамодели

А. В. Курочкин, В. С. Садов, А. И. Головатый

Разработка интегрированных основанных на знаниях систем поддержки принятия решений с долговременным хранилищем для использования в медицине требует создания нескольких программных модулей с собственной реализацией, которая включает множество дублирующихся и необобщенных компонентов для описания тех данных, которыми оперирует система. В качестве решения этой проблемы в работе предлагается описательное задание в виде семантической метамодели для медицинских информационных системах, в основе которого лежит расширяемый формат дескрипторов сущностей и атрибутов, а также интеграционную среду, которая позволяет приводить внешние данные к общему формату, упрощает доступ к данным благодаря генерации описаний схемы и программных интерфейсов для доступа к долговременному хранилищу, а также устраняет необходимость в ручной разработке пользовательского интерфейса благодаря процедурной генерации представлений на основе форм и списков.

Received 31.05.2021