An analysis of structural interoperability in ISO 13606 based on typed lambda calculus

December 9, 2014

Abstract

ISO 13606 defines a standard specification for the exchange and integration of electronic health record (EHR) data with the goal of interoperability of EHR data across heterogeneous systems. In order to achieve this goal, the ISO 13606 standard provides a dual-model architecture in which archetypes are extensible schemas of EHR data via their specialization and composition mechanisms. However, the ISO 13606 standard lacks the formal semantics of archetypes, which makes it difficult to build EHR systems or archetype repositories in a consistent and longitudinal manner. The goal of the present study was to clarify the archetype semantics of ISO 13606 by means of a formal methodology called typed lambda calculus, which has been widely used to define and analyze the semantics of modern programming languages and database systems. We focus on the variance and immutability of the archetype, because these factors determine the expressive power of archetypes. We defined a type system that represents the fundamental components of ISO 13606 semantics and analyzed the semantics based on a deductive system of the type theory. The obtained results indicate that the archetypes should be covariant and immutable schemas in order to guarantee both the structural interoperability of EHR data and the extensibility of archetypes.

1 Introduction

1.1 Background

The efficient use of medical resources and the fulfillment of large-scale clinical research requires integration of electronic healthcare record (EHR) systems that store diverse clinical data for long periods of time. This challenge, however, remains unsolved even today. The heterogeneity of these data sources, i.e., the interoperability of EHR data from local systems in a consistent and longitudinal manner, is a major obstacle for the integration or exchange of data across EHR systems [1]. In order to achieve interoperability, the schema of EHR data must satisfy the requirements of various organizations and evolve over time in accordance with the development of medical domains.

ISO 13606 standard defines a common healthcare model for such a purpose. The overall goal of the standard is "to devise a generalized approach representing every conceivable kind of health record data structure in a consistent way" [2, vi] so as to "support the interoperability of systems and components that need to communicate EHR data" [2, v]. Due to the complexity and continuing evolution of the healthcare domain, the common model must meet the needs of organizations and cope with the progress of medical domains. In order to overcome these difficulties, ISO 13606 has adopted a dual-model approach that distinguishes two modeling layers, i.e., information layer and knowledge layer, which are explained later.

While this dual-model architecture is a major feature of ISO 13606, the original idea stems from the openEHR specification, which was also an attempt at the "sharing of EHRs via interoperability at data and knowledge levels" [3, p.12]. In fact, ISO 13606 is largely considered to be a subset of the full openEHR specification [4, 5]. Therefore, both specifications share common features. The present paper refers primarily to ISO 13606 as a standard for the interoperability of EHR systems. However, for the case in which ISO 13606 is short of sufficient descriptions and the openEHR specification is more appropriate, we will refer to the openEHR specification as a secondary resource.

As specifications become larger and more complicated, the consistency and correctness of the specifications become uncertain, and ISO 13606 is no exception. The ISO 13606 specification describes the behavior of archetypes but does not define any formal semantics. This is problematic because the lack of a formal definition makes the implementation of an EHR system difficult. For example, when we define an archetype using both specialization and composition, are we allowed to combine them in an arbitrary manner? Or, are there constraints embedded in the semantics of the archetypes? These questions have to be answered before the design and implementation of an EHR system.

Since ISO 13606 provides insufficient explanation regarding these questions, we must look back at the original openEHR specifications for clues. In the two specifications, we found two important statements. One is cited from the ISO 13606 specification, and the other from the openEHR specification.

(i). Assertion 1

> Any data created via the use of a specialized archetype shall be conformant to both the archetype and its parent [6, viii].

(ii). Assertion 2

> Data created with any specialized archetype will always be matched by queries based on the parent archetype [3, p.51].

These assertions are prerequisites for laying the groundwork on archetype semantics, because they suggest the requirements for EHR data conformance and queries based on archetype. However, the exact meanings remain unclear because natural language descriptions leave room for a wide range of interpretations. For example, what does the phrase 'conformant to' mean exactly? How

can we assure that certain queries match appropriate EHR data and that others do not?

1.2 Objective

The goal of the present study is to clarify the ISO 13606 archetype semantics in order to support the two assertions above. We investigate the semantics in a formal manner under the assumption that at least these two assertions must hold, and we pursue further consequences that are logically derived from the semantics. The term 'formal methodology' refers to the situation in which every term in a syntax or every property in semantics is defined by a symbol so that each has a precise meaning that serves as a foundation for reasoning or verification.

As a means of such formalization, we used **typed lambda calculus**, which is widely used as meta-language in the field of computer science to define and analyze the semantics of various programming languages and database systems [7, 8]. We believe that this formalization technique can contribute to the analysis of ISO 13606 in the same manner as previous domains in computer science. To the best of our knowledge, this is the first attempt to apply typed lambda calculus to semantics in the medical field.

2 Material and Methods

2.1 Concepts for EHR data integration

Even if a formal methodology is necessary for building a solid semantics, it is not sufficient. When we encounter different results from different premises, we must choose one of the results based on higher conceptual criteria, even though these results are both valid in terms of formalization. Before introducing the formal methodology of typed lambda calculus, we begin by explaining several fundamental concepts for the exchange and integration of EHR data. These concepts are 'structural interoperability' and 'schema extensibility'.

2.1.1 Structural interoperability should be guaranteed

Every standardization of clinical information, including ISO 13606, attempts to achieve a certain level of interoperability between EHR systems. ISO 13606 defines 'interoperability' as the "ability for data shared by systems to be understood at the level of fully defined domain concepts" [2, p.5]. However, the term 'to be understood' is not precise in terms of computer science. Computers can never 'understand' the meaning of data, they only compute. Moreover, we believe that this ambiguity has its root in the fact that two different types of concepts are being explained using one term 'interoperability'. Thus, we divided the concept of interoperability into two levels, as follows.

(i). Structural Interoperability

Structural interoperability is defined as the ability to process data based on a shared formal schema with sound semantics. This type of interoperability is referred to as 'structural' because modern database schemas or data models are solely concerned with the structure of data, such as field names, data types, and relations.

(ii). Semantic Interoperability

Semantic interoperability, in contrast, is defined as the ability to process data based on the shared meanings of terms [9, 10, 11, 12, 13]. In order to share the meanings of terms, we need mapping between objects in a virtual world (i.e., the computer) and concepts in the real world, because the 'meaning' refers to our interpretation of assignments between concrete objects and abstract concepts [14, p.97]. The concepts in the real world, however, can neither be directly represented nor processed by computer. Therefore, the assignments are technically performed both by defining ontologies, which are surrogates for the real world concept, and by annotating terms or attributes with the corresponding items in the ontologies.

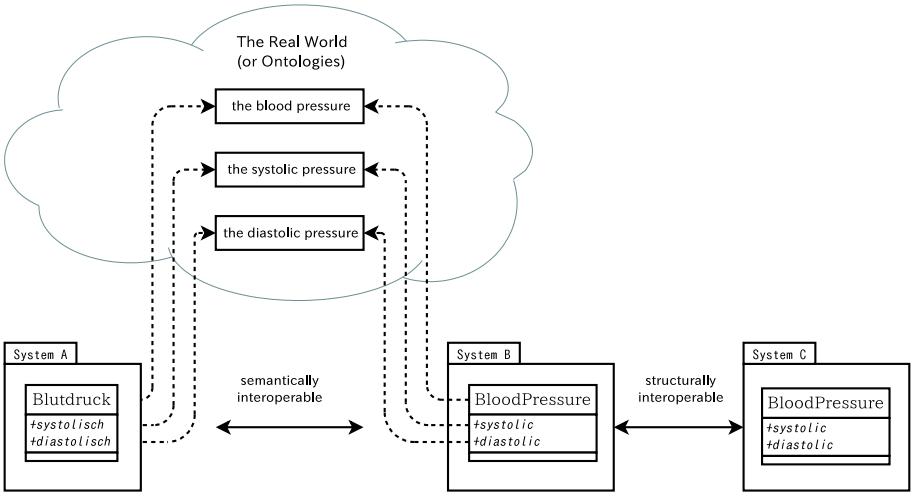

Figure 1: Structural interoperability and semantic interoperability

Figure 1 illustrates the difference between structural interoperability and semantic interoperability. All three EHR systems define blood pressure in their schemas. Systems B and C have structurally identical schemas and are therefore structurally interoperable. System A has a different schema from the other systems, because their attribute names are written in German. As a result, Systems A can be semantically interoperable with System B only if the mapping between the attributes and the real concepts is correctly defined.

Figure 1 appears to indicate that structural interoperability and semantic interoperability are two independent concepts. However, structural interoperability is necessary even when semantic interoperability is to be accomplished, because semantic annotations are useful only if the valid values can be obtained without errors. For example, let us think of a situation in which we collect systolic blood pressure data both from Systems A and B in Fig. 1. According to the ontology, we get table names (BloodPressure and Blutdruck) and column names (*systolic* and *systolisch*) for each system. Only when we have this structural information can we actually retrieve the data. Thus, structural interoperability is more fundamental than semantic interoperability. The present study focuses on the concept of structural interoperability in the investigation of archetype semantics.

2.1.2 EHR schema should be extensible

When every EHR system is built upon a unified and fixed EHR schema, we can easily achieve structural interoperability. In real settings, however, building such a single giant schema is difficult, especially in the medical domain, where concepts are not only large, but also open-ended with respect to breadth, depth, and complexity [15, 16].

The following example, which is a longitudinal cancer cohort study, illustrates the difficulties involved (Fig. 2). At the start of the study, we store smoking histories as the exposure level and several tumor markers as surrogates for cancer outcome. When a novel marker is discovered during the course of the study, we want to add the marker as an additional item, such as RevisedOutcome in RevisedCase (Fig. 2). Or, if an unexpected exposure, such as a nuclear plant accident, is observed, the extra exposure (in this case, an irradiation dose) should be added.

As this example indicates, when data are stored and traced longitudinally, the EHR schema must be extended as medicine progresses or a new event occurs. Otherwise, a number of disorganized schemas, many of which are similar but slightly different, will accumulate, and the cost of maintenance will increase significantly.

2.2 ISO 13606 architecture

In this section, we present an overview of ISO 13606 in terms of the dual model architecture. In particular, we focus on the specialization and composition of archetypes. At the end of this section, we point out the problem associated with their combination (archetype variances).

2.2.1 Dual-model architecture

Like openEHR, ISO 13606, adopts a dual-model architecture (Fig. 3) to guarantee interoperability and support flexibility of archetypes as EHR schema. The architecture defines a Reference Model (RM) and an Archetype Model (AM)

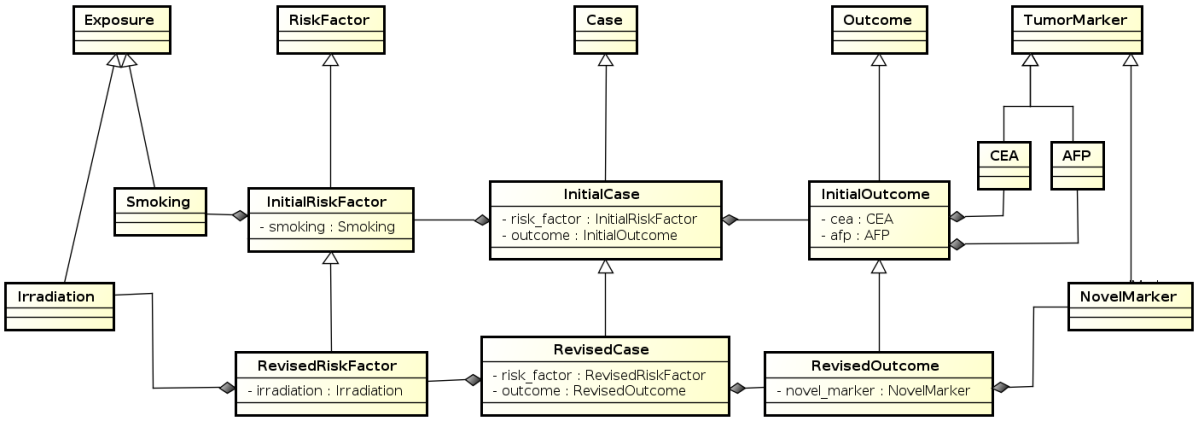

Figure 2: Unified Modeling Language (UML) diagram of an imaginary cohort study

The diagram describes the schema of an imaginary cohort study in chronological order. The lower diagram indicates the more recent content.

in the lower layer and archetypes in the upper layer, and it proposes building archetypes by constraining RM classes by means of an AM.

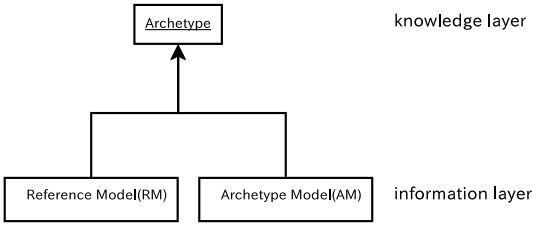

Figure 3: Relationships between RM, AM, and Archetype in the dual model architecture

The diagram describes the dual-model architecture of ISO 13606, which consists of two layers, an information layer and a knowledge layer. The lower side of the figure indicates the information layer, in which the Reference Model and the Archetype Model are located, and the upper side indicates the knowledge layer, in which the Archetype resides.

The lower layer of the dual-model architecture contains a Reference Model and an Archetype Model. The **Reference Model** defines a set of data structures and data types from which any health record information is composed [2, p.vi]. Sitting in the lower layer of the dual-model architecture, the RM represents the generic properties of health record information [2, p.vi] , and forms the generic building blocks of the EHR.

The **Archetype Model** also sits in the lower layer, and acts as a skeleton or template for archetypes. The model consists of "identifying information, a description (its metadata), a definition (expressed in terms of constraints on instances of an object model), and an ontology" [6, p.vii]. The definition section defines the hierarchical organization of a set of nodes and constraints on the permitted values of attributes and data values [6, p.6]. In other words, the AM defines the structure and constraints of EHR data from combinations of RM entities in their definition section.

In contrast to the previous models, the **Archetype** resides in the upper layer of the dual-model architecture, and defines medical domains in the form of structured and constrained combinations of RM and AM entities. While the difference between an Archetype and the AM is confusing, they reside on different layers, as shown in Fig. 3. The RM and AM are static because they are defined by the specification in advance, whereas Archetypes are dynamic because they can be constructed at any time as clinical practice evolves [6, vi]. At this stage, the significance of the dual-model architecture is revealed. In ordinary architectures that have only one layer of modeling, data models or a schema is already defined in a system or specification, which makes it difficult or impossible to modify or extend the schema. In dual-model architectures, which include the basic components of schemas and the general mechanism for their construction, the actual schemas (i.e., archetypes) can be constructed dynamically from these components.

Archetypes can be serialized through a language named Archetype Definition Language (ADL) [6, xiv]. While AM, RM, and Archetype are all inhabited in the layer of semantics that is invisible and abstract, ADL is inhabited in the layer of syntax that is readable and intuitive for human users. Therefore, we presented several examples in the form of ADL fragments to help in understanding the semantics.

2.2.2 Specialization and composition of archetypes

The dual-model architecture enables archetypes to be defined dynamically and thus realizes schema flexibility. Moreover, ISO 13606 provides a method by which to further enhance flexibility, namely, specialization and composition of archetypes.

First, archetypes can be specialized [6, viii]. New archetypes can be defined by further constraining the parent archetype. Similar to class inheritance in object-oriented languages, the child of an archetype inherits all its attributes from the parent archetype and adds new items or refines existing items in the child archetype.

Second, archetypes can be composed [6, ix]. A new archetype is defined by containing other archetypes when they are specified in 'slots' of the new archetype. For example, Fig. 4 is a fragment of ADL with slots, indicating that archetypes for which the id is matched by `/.*\.iso-ehr\.section\..*/` can be composed, and that archetypes of `/.*\.iso-ehr\.section\.patient_details\..*/` should not be composed in the *items* attribute. The 'composition' mechanism

allows large data structures to be flexibly constrained via the hierarchical re-use of smaller archetypes [3, 17]. Note that the ISO 13606 specification allows a composed archetype to be specified either by stating a unique archetype identifier, or vaguely by wildcards (i.e., matches many possible archetypes) [18, p.61] as shown in Fig. 4. This issue is discussed later herein.

```
SECTION [at2000] occurrences matches {0..1} matches {
   items matches {
      allow_archetype SECTION occurrences matches {0..*} matches {
         include
            id matches {/.*\.iso-ehr\.section\..*/}
         exclude
            id matches {/.*\.iso-ehr\.section\.patient_details\..*/}
      }
   }
}
```

Figure 4: Excerpt of ADL with an archetype slot

2.2.3 Variances of archetype

When we define an archetype by combining specialization and composition, we encounter several possibilities, referred to as variances [19, p.26][20]. The variance of a type is said to be covariant in its components when the type varies in the same direction as one of its parts with respect to subtyping (Fig. 5(a)). The variance of a type is said to be contravariant when the type varies in the opposite direction as one of its parts with respect to subtyping (Fig. 5(b)). Finally, the variance of a type is said to be invariant if the type is neither covariant nor contravariant (Fig. 5(c)). We presented three alternatives of archetype variances as ADL fragments in Fig. 6 for better understanding the differences. Note that in each example, the right-hand ADL is a specialization of the left-hand ADL.

The actual variance of an archetype depends on the semantics of the archetype, i.e., how the archetype is constructed as a type. We consider this problem to be a major issue in archetype semantics because it affects the archetype expressiveness and extensibility as EHR schemas.

2.3 Typed lambda calculus

In this section, we briefly introduce typed lambda calculus, which is the tool we use to investigate the archetype semantics. Lambda calculi are mathematical abstractions of computations, and in the typed lambda calculus, every variable must be explicitly typed. We herein use the terms typed lambda calculus and type theory interchangeably.

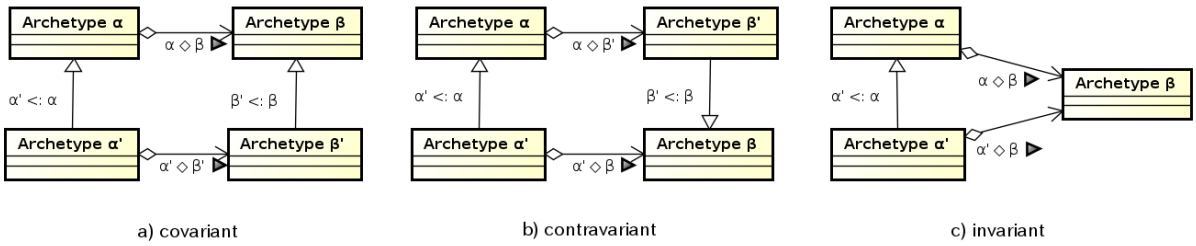

Figure 5: Three alternative archetype variances in a UML diagram

The figure shows three alternative archetype variances in a UML diagram. a) covariant: if $\alpha' <: \alpha$ and $\beta' <: \beta$, then $\alpha' \diamond \beta' <: \alpha \diamond \beta$, b) contravariant: if $\alpha' <: \alpha$ and $\beta' <: \beta$, then $\alpha' \diamond \beta <: \alpha \diamond \beta'$, and c) invariant: if $\alpha' <: \alpha$ and $\beta' = \beta$, then $\alpha' \diamond \beta <: \alpha \diamond \beta$. Note that, in this figure, the operator $<:$ denotes 'is a subtype of', and $\diamond$ denotes 'is composed of'.

2.3.1 How type theory is applied to ISO 13606

Before introducing the details of the typed lambda calculus, we present an overview of the correspondence between the components of typed lambda calculus and ISO 13606. In an abstract sense, a type is a well-defined set of values, so that every model in ISO 13606 can be encoded as a type. Because each type has its own characteristics, we must carefully decide which types can be assigned to each component of ISO 13606.

We encoded unstructured RM components as primitive types in type theory. Not surprisingly, a string in the RM is mapped to the string type. In contrast, structured RM components are mapped to structured types in type theory, such as record type, variant type, and list type. For example, the TEXT type in ISO 13606 has charset, languages, and originalText as its field [2, p.36]. Therefore, the TEXT type can be encoded as a record type using Eq. 1. Note that, in the present paper, concrete types, such as Record or CS, start with a capital letter and are shown in SansSerif font.

Archetypes, which are also structured, are encoded in record types. For example, the vital signs archetype (Fig. 7) can be encoded as shown in Eqs. 2 and 3. The specialization of archetypes corresponds to Record subtyping, and the composition of archetypes corresponds to Record nesting.

EHR data are instances of Archetypes, [1] i.e., instances of the corresponding Record type, because each EHR datum is created according to the corresponding archetype. For example, Eq. 4 represents an EHR datum for the vital sign archetype of Eq. 2. Note that instance names are written in lower case, e.g., *record*, and the details of BloodPressure and BloodTemperature are omitted for clarity in this equation.

Table 1 summarizes our assignment of ISO 13606 to type theory. We believe

[1] Note that this is not an "archetype instance", which is defined as a metadata class instance of an archetype model in the ISO 13606 specification [2, p.2].

a) covariant:

```
archetype
  ISO-EN13606-COMPOSITION.alpha.v1


definition
  COMPOSITION[at0000] matches {
    content matches {
      allow_archetype SECTION[at0001] matches {
        include
        archetype_id/value matches {
          "ISO-EN13606-SECTION.beta.v1"}}}}
```

```
archetype
  ISO-EN13606-COMPOSITION.alpha-dash.v1
specialize
  ISO-EN13606-COMPOSITION.alpha.v1
definition
  COMPOSITION[at0000] matches {
    content matches {
      allow_archetype SECTION[at0001] matches {
        include
        archetype_id/value matches {
          "ISO-EN13606-SECTION.beta-dash.v1"}}}}
```

b) contravariant:

```
archetype
  ISO-EN13606-COMPOSITION.alpha.v1


definition
  COMPOSITION[at0000] matches {
    content matches {
      allow_archetype SECTION[at0001] matches {
        include
        archetype_id/value matches {
          "ISO-EN13606-SECTION.beta-dash.v1"}}}}
```

```
archetype
  ISO-EN13606-COMPOSITION.alpha-dash.v1
specialize
  ISO-EN13606-COMPOSITION.alpha.v1
definition
  COMPOSITION[at0000] matches {
    content matches {
      allow_archetype SECTION[at0001] matches {
        include
        archetype_id/value matches {
          "ISO-EN13606-SECTION.beta.v1"}}}}
```

c) invariant:

```
archetype
  ISO-EN13606-COMPOSITION.alpha.v1


definition
  COMPOSITION[at0000] matches {
    content matches {
      allow_archetype SECTION[at0001] matches {
        include
        archetype_id/value matches {
          "ISO-EN13606-SECTION.beta.v1"}}}}
```

```
archetype
  ISO-EN13606-COMPOSITION.alpha-dash.v1
specialize
  ISO-EN13606-COMPOSITION.alpha.v1
definition
  COMPOSITION[at0000] matches {
    content matches {
      allow_archetype SECTION[at0001] matches {
        include
        archetype_id/value matches {
          "ISO-EN13606-SECTION.beta.v1"}}}}
```

Figure 6: Examples of archetype variance in ADL

$$\mathsf{TEXT} \equiv \mathsf{Record}\{charset{:}\mathsf{CS}, languages{:}\mathsf{CS}, originalText{:}\mathsf{String}\} \qquad (1)$$

```
definition
   SECTION[at0000] matches {   -- Vital signs
      items cardinality matches {2} matches {
         allow_archetype OBSERVATION[at0001] matches {
            include
               archetype_id/value matches {"openEHR-EHR-OBSERVATION.blood_pressure.v1"}
         }
         allow_archetype OBSERVATION[at0002] matches {
            include
               archetype_id/value matches {"openEHR-EHR-OBSERVATION.body_temperature.v1"}
         }
      }
   }
```

Figure 7: Excerpt of a vital signs archetype

This is an example of the definition section in a vital signs archetype. Note that we modified the certified archetype in the Clinical Knowledge Manager (CKM) repository to match the explanation herein.

$$\mathsf{VitalSigns} \equiv \mathsf{Record}\{definition[at0000]{:}\mathsf{SECTION}^{\mathsf{VitalSigns}}\} \tag{2}$$

$$\mathsf{SECTION}^{\mathsf{VitalSigns}} \equiv \mathsf{Record}\{items[at0001]{:}\mathsf{BloodPressure},\ items[at0002]{:}\mathsf{BloodTemperature}\}\} \tag{3}$$

$$vital\_signs \equiv record\{definition[at0000]{:}record\{items[at0001]{:}record\{systolic{:}120, diastolic{:}80\},\ items[at0002]{:}record\{temperature{:}37.5, unit{:}"C"\}\}\}\} \tag{4}$$

that these assignments are intuitively plausible, as explained above. However, this does not exclude other possibilities of assignments, which might lead to different results from those of the present study.

Table 1: Correspondence between type theory and ISO 13606

ISO 13606	type theory
primitive types in RM	basic types
data structures in RM	structured types
archetype	record type
EHR data	instances of record type

2.3.2 Introducing types

We next present the minimum set of typed lambda calculus that is sufficient to clarify the basis of archetype semantics (in particular variance and immutability of archetypes) by means of formalization in support of the aforementioned two assertions. We start from fundamental types, such as the function or basic type, and progressively extend the system with more complex types, such as the record or variant type (Table 2). For a more comprehensive explanation of type theory, see for instance, [21, 22, 23].

Table 2: List of major types and corresponding notation used in the present study

type		notation
function type		$\tau \to \sigma$
basic types		$\mathsf{String}, \mathsf{Boolean}$, etc.
structured types	record type	Record
	variant type	$\mathsf{Variant}$
	list type	$[\tau]$

First, a formal assertion relating entities such as terms or types is referred to as a **judgment** (Eq. 5). The judgment $\Gamma \vdash M : \tau$ should be read as "a term M has a type of τ under an environment Γ", and the judgment $\Gamma \vdash \tau$ means that τ is a well-formed type under the environment Γ. In either case, Γ is a set of assumptions about the types of free variables in M and is referred to as the **type environment**. Note that, in the present study, we use M and N as terms, and τ and σ for meta-variables over type.

$$\Gamma \vdash \Omega \qquad \text{(judgment)} \tag{5}$$

In typed lambda calculus, types are introduced into the system by adding **typing rules**, such as those given by Eq. 6. These typing rules assert the validity of certain judgments based on other valid judgments. The judgments above the horizontal line are referred to as the premises of the rule, and those below the horizontal line are referred to as the conclusion.

The most fundamental type is the Function Type ($\tau \to \sigma$), which satisfies the typing rule given by Eq. 6. The notation ($M\ N$) in Eq. 7 denotes a function application in which a function M is applied to the argument N. A function is represented by a lambda abstraction, such as $\lambda x{:}\tau.M$ (Eqs. 9 and 10). In this lambda abstraction, x represents a parameter of the function, and M represents the body of the function [22, p.14]. When we apply the term M of function type $\tau \to \sigma$ to the value of type τ, we obtain the term of type σ (Eq. 7). This function type appears in database queries (such as Eq. 39), because a query is a function that accepts a database and returns a result set of the database.

Typing rules:

$$\frac{\Gamma\ \vdash \tau \qquad \Gamma\ \vdash \sigma}{\Gamma\ \vdash \tau \to \sigma} \qquad \text{(function type)} \qquad (6)$$

$$\frac{\Gamma \vdash M{:}\tau \to \sigma \qquad \Gamma \vdash N{:}\tau}{\Gamma\ \vdash (M\ N){:}\sigma} \qquad \text{(application)} \qquad (7)$$

Figure 8: Function type and its application

Terms:

$$M ::= \ldots \qquad (8)$$

$$\mid\ \ \lambda x{:}\tau.M \qquad (\lambda\ \textit{abstraction}) \qquad (9)$$

Typing rule:

$$\frac{\Gamma, x{:}\tau\ \vdash M{:}\sigma}{\Gamma\ \vdash \lambda x{:}\tau.M{:}\tau \to \sigma} \qquad (10)$$

Figure 9: λ abstraction

Because EHR data are aggregates of various types of data, we need to introduce such data types. For example, blood pressure data are constructed as aggregates with the numeric type for the pressure value and a string type for the unit of measurement. We classify these data types into two categories: basic and structured. Basic types, such as Boolean or Integer, are primitive data types in that they lack any internal structure. When these primitive types are built in BasicType, they are available in our system as Eq. 11. $\Gamma\ \vdash \diamond$ means that Γ is well-typed, which means that the type environment Γ has nothing but

valid types.

$$\frac{\Gamma \vdash \diamond \qquad \tau \in \mathsf{BasicType}}{\Gamma \vdash \tau} \tag{11}$$

ISO 13606 provides several kinds of primitive types, such as String and Boolean [2, p.42]. For example, the Boolean type can be introduced into the proposed type system by Eq. 11, because Boolean $\in$ BasicType holds in ISO 13606. As one such primitive type, we introduce the **Unit** type (Eq. 12), which has no corresponding type in ISO 13606. The Unit type has only one value called *unit* (Eq. 13), which is often used as a filler for uninteresting results [22, 21]. In the present paper, this type is used in a field of Variant type (Eq. 24) and a return value of assignments (Eq. 46).

Types:

$$\tau ::= \ldots \mid \mathsf{Unit} \qquad \text{(type unit)} \tag{12}$$

Typing rules:

$$\frac{\Gamma \vdash \diamond}{\Gamma \vdash \mathit{unit}{:}\mathsf{Unit}} \qquad \text{(value unit)} \tag{13}$$

Figure 10: Unit

EHR data are inherently structured in that they consist of large collections of data entities of several types. Thus, we introduce such structured data types into our type system in order to capture the ISO 13660 data model. These structured types are Record, Variant, and List types.

(i). Record Type

A **Record** type is a cross-product of the values with a projection operation for extracting components by name [22, 21, p.18]. As stated previously, most of the components in archetypes, such as Fig. 7, are encoded as Record types (Eqs. 2 and 3).

The definition of structured type has three sections. In the record type, the terms section defines syntactic forms of a record and its projection (Eqs. 14 and 15). The types section defines the Record type itself (Eq. 16). And, the typing rules section defines rules for the type of record construction and its projection (Eqs. 17 and 18). Note that $i \in 1..n$ is shorthand for $\{i \mid i \in \mathbb{N} \wedge 1 \leq i \wedge i \leq n)\}$, which indicates that the variable i ranges over natural numbers $\mathbb{N}$ from 1 to n. Also note that a record is basically a tuple, the components of which are not indexed by numbers but rather labeled by names. Therefore, the components in Record type are unordered.

Terms:

$$M ::= \ldots$$

$$\mid \; record\{l_i = M\} \qquad where\ i \in 1..n \qquad \text{(record construction)} \quad (14)$$

$$\mid \; M.l_i \qquad \text{(record projection)} \quad (15)$$

Types:

$$\tau ::= \ldots$$

$$\mid \; \mathsf{Record}\{l_i{:}\tau_i\} \qquad where\ i \in 1..n \qquad (16)$$

Typing rules:

$$\frac{\Gamma \vdash M_i{:}\tau_i}{\Gamma \; \vdash record\{l_i{:}M_i\} : \mathsf{Record}\{l_i{:}\tau_i\}} \qquad where\ i \in 1..n \qquad (17)$$

$$\frac{\Gamma \vdash M{:}\mathsf{Record}\{l_i{:}\tau_i\}}{\Gamma \; \vdash M.l_i{:}\tau_i} \qquad where\ i \in 1..n \qquad (18)$$

Figure 11: Records

(ii). Variant Type

While Record type requires all of the fields to be set by values, there is a case in which only one of the fields must be filled in. This is called the **Variant** Type, which is a named disjoint union of types [21, p.136][22, p.19]. Figure 13 is an example of such a case, and this fragment of the archetype should be encoded as a variant type (Eqs. 23 and 24).

(iii). List Type

EHR data sometimes contain a series of events, such as a sequence of laboratory data in the context of hemodialysis. These kinds of collective values are represented by the **List** type [22, 21, 24]. In the present paper, the List type is denoted by $[\tau]$, describing finite-length sequences the element of which are taken from τ. Figure 14 is an example of such a case, and the *parts* attribute in this fragment of the archetype should be encoded as a List type (Eq. 26).

2.3.3 Subtyping

EHR data are organized hierarchically. In Fig. 15, for example, ELEMENT and CLUSTER are subordinate to ITEM in their hierarchy, because each subordinate class (i.e., ELEMENT and CLUSTER) inherits all of the members of its parent class ITEM. This hierarchical relation, which is implemented as inheritance in object-oriented languages, is formalized as **subtyping** in type theory [25, p.85]. In this section, we introduce general rules for subtyping and specific rules for

Terms:

$$M ::= \ldots$$
$$\mid \; variant\langle l_i = M_i \rangle \qquad where\ i \in 1..n \quad \text{(variant construction)} \tag{19}$$

Types:

$$\tau ::= \ldots$$
$$\mid \; \mathsf{Variant}\langle l_i{:}\tau_i \rangle \qquad where\ i \in 1..n \tag{20}$$

Typing rules:

$$\frac{\Gamma \vdash M_j{:}\tau_j \qquad where\ i \in 1..n}{\Gamma \; \vdash variant\langle l_j{:}M_j \rangle : \mathsf{Variant}\langle l_i{:}\tau_i \rangle^{i \in 1..n}} \tag{21}$$

$$\tag{22}$$

Figure 12: Variants

```
value matches {
    DV_CODED_TEXT matches {
      defining_code matches {
        [local::
         at1000,  -- Hepatitis A
         at1001   -- Hepatitis B
        ]
      }
    }
}
```

Figure 13: Example of a Variant Type in DV_CODED_TEXT

This fragment defines a constraint of type DV_CODED_TEXT, the value of which should be either hepatitis A or B.

$$value : \mathsf{DV\_CODED\_TEXT}^{\mathsf{HepatitisVirus}} \tag{23}$$

$$\mathsf{DV\_CODED\_TEXT}^{\mathsf{HepatitisVirus}} \equiv \mathsf{Record}\{defining\_code{:}\mathsf{Variant}\langle \mathrm{at1000{:}Unit},\ \mathrm{at1001{:}Unit} \rangle\} \tag{24}$$

```
definition
   CLUSTER [at0000] matches {
      parts cardinality matches {0..*; ordered} matches {
         allow_archetype ENTRY matches {
            include
               archetype_id/value matches {
                  "CEN-EN13606-ENTRY.laboratory.v1"
               }
         }
      }
   }
```

Figure 14: Example of a List Type in CLUSTER

$$definition{:}\mathsf{CLUSTER}^{\mathsf{Haemodialysis}} \tag{25}$$

$$\mathsf{CLUSTER}^{\mathsf{Haemodialysis}} \equiv \mathsf{Record}\{parts{:}[\mathsf{LaboratoryEntry}]\} \tag{26}$$

each type. When we define the specific subtyping rules so as to be compatible with the general rules, we are able to check the validity of a given subtyping based on a deductive system of the type theory.

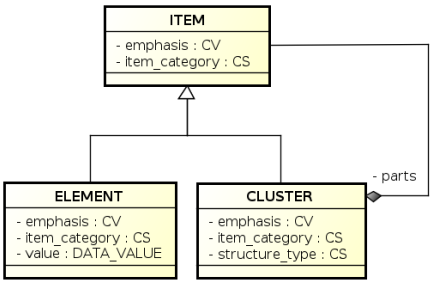

Figure 15: Excerpt of the ISO 13606 Reference Model

This UML diagram is an excerpt from the ISO 13606 Reference Model [2, p.10] and defines the hierarchy of ITEM and its subordinates ELEMENT and CLUSTER.

The central theorem of subtyping is the **subsumption principle** (Eq. 27), which says that if type τ' is a subtype of τ (i.e., $\tau' <: \tau$), then an instance of type τ' can be used in any context in which an instance of type τ can appear. Note that the subtyping operator '$<:$' is a reflexive transitive closure (Eqs. 28 and 29) similar to the numeric comparison operator '$\leq$', so that $\tau' <: \tau$ and $\tau <: \tau'$ means that τ is equal to τ' (cf., $a \leq b$ and $b \leq a$ means that $a = b$).

Subtyping rules should be defined for each type, and every subtyping rule should be compatible with the subsumption principle. We next present the

Typing rules:

$$\frac{\Gamma \vdash M{:}\tau' \qquad \Gamma \vdash \tau' <: \tau}{\Gamma \vdash M{:}\tau} \quad \text{(subsumption principle)} \quad (27)$$

$$\frac{\Gamma \vdash \tau}{\Gamma \vdash \tau <: \tau} \quad \text{(reflexive)} \quad (28)$$

$$\frac{\Gamma \vdash \tau'' <: \tau' \qquad \Gamma \vdash \tau' <: \tau}{\Gamma \vdash \tau'' <: \tau} \quad \text{(transitive)} \quad (29)$$

subtyping rules for structured types. In the record type, one record type is a breadth subtype of another if the first has at least all of the fields of the second (Eq. 30), and the other record type is a depth subtype of another if they have exactly the same fields, but the types of the corresponding fields are subtypes (Eq. 31). In the variant type, the depth subtyping rule for the variant type is identical to that of the record type (cf., Eqs. 33 and 31), but the breadth subtyping rules for the variant type is different from that for the record type: a subtype of the variant type cannot have more fields than its supertype (Eq. 32). The variance of the list type is defined as covariant, as in the case of the Record type. Therefore, in the case of a list having elements of type τ', and $\tau' <: \tau$, we can regard the list as having elements of type τ (Eq. 34) [21, p.197].

Since we abstracted archetypes as record types, archetype specialization can be modeled as record subtyping. We can add new items or modify the existing items in the specialized archetype as long as the extensions are compatible with Eqs. 30 and 31.

Structured types, even basic types, can be subtyped by introducing an ad hoc subtyping rule [22, p.28]. When we define the ad hoc rules, their actual characteristics of the values should be compatible with the subsumption principle. Otherwise, the system is consistent in theory, but useless in reality. For example, since an integer value can be used in the context in which a real value can appear (e.g., 1 as 1.0), a subtyping rule Integer <: Real can be introduced. This can also be applied to the structured types in the RM, as long as they are compatible with the ISO 13606 specification. According to Fig. 15, for example, we can define subtyping between ELEMENT and ITEM as shown in Eq. 35 and CLUSTER and ITEM as shown in Eq. 36.

$$\frac{\Gamma \vdash \diamond}{\Gamma \vdash \mathsf{ELEMENT} <: \mathsf{ITEM}} \quad \text{(ad-hoc)} \quad (35)$$

$$\frac{\Gamma \vdash \diamond}{\Gamma \vdash \mathsf{CLUSTER} <: \mathsf{ITEM}} \quad \text{(ad-hoc)} \quad (36)$$

Record subtyping rules:

$$\frac{\Gamma \vdash \tau_i \qquad where\ i \in 1..n+m}{\Gamma\ \vdash \mathsf{Record}\{l_i{:}\tau_i\}_{i\in 1..n+m} <: \mathsf{Record}\{l_i{:}\tau_i\}_{i\in 1..n}} \tag{30}$$

$$\frac{\Gamma \vdash \tau_i' <: \tau_i}{\Gamma\ \vdash \mathsf{Record}\{l_i{:}\tau_i'\} <: \mathsf{Record}\{l_i{:}\tau_i\}} \qquad where\ i \in 1..n \tag{31}$$

Variant subtyping rules:

$$\frac{\Gamma \vdash \tau_i \qquad where\ i \in 1..n+m}{\Gamma\ \vdash \mathsf{Variant}\langle l_i{:}\tau_i\rangle_{i\in 1..n} <: \mathsf{Variant}\langle l_i{:}\tau_i\rangle_{i\in 1..n+m}} \tag{32}$$

$$\frac{\Gamma \vdash \tau_i' <: \tau_i}{\Gamma\ \vdash \mathsf{Variant}\langle l_i{:}\tau_i'\rangle <: \mathsf{Variant}\langle l_i{:}\tau_i\rangle} \qquad where\ i \in 1..n \tag{33}$$

List subtyping rule:

$$\frac{\Gamma \vdash \tau' <: \tau}{\Gamma\ \vdash [\tau'] <: [\tau]} \tag{34}$$

Figure 16: Subtyping rules for Record, Variant, and List

2.3.4 Type system and its deductive system

Deductive proofs are necessary in order to confirm the validity of the assertions under a given system. When we define a number of type rules based on previous disciplines, the collection of axioms and typing rules is referred as a **type system**. Once we have a type system, we can construct a deductive proof by combining the typing rules, and the proof is referred to as a **derivation tree**. Any derivation tree has leaves at the top and a root at the bottom, where each judgment is obtained from those immediately above it by applying some of the rules of the system. Figure 17 shows a simple example of a derivation tree, indicating that the function of type $\mathsf{String} \rightarrow \mathsf{Integer}$ yields an integer type when applied to a string value. Note that the number in brackets to the right of the horizontal line indicates the equation number of the rule that was used (in this case, Eq. 7). This deductive proof system is the most powerful feature of typed lambda calculus, because we can investigate the semantics of ISO 13606 with formal deductions.

$$\frac{\Gamma \vdash \lambda x{:}\mathsf{String}.M{:}\mathsf{String} \rightarrow \mathsf{Integer} \qquad \Gamma \vdash \text{"}text\text{"}{:}\mathsf{String}}{\Gamma \vdash ((\lambda x{:}\mathsf{String}.M)\ \text{"}text\text{"}){:}\mathsf{Integer}}\ [7]$$

Figure 17: An example of a derivation tree

2.4 Application of type theory to ISO 13606

In the previous sections, we introduced basic components of type theory and the correspondence of type theory to ISO 13606. Next, we need to encode the problems in terms of this formal language in order to support the assertions (Assertions 1 and 2).

2.4.1 How to formalize queries

Assertion 2, which states that "data created with any specialized archetype will always be matched by queries based on the parent archetype", is related to a requirement as to which queries and specialization should be obeyed. Before deriving the assertion from the proposed type system, we encoded this assertion in terms of typed lambda calculus as follows.

Database theories [26, 27, 28, 29] tell us that the database can be presented as list of data, and its queries can be encoded as lambda terms and `foldr` (Eqs. 37 and 38). Note that archetype is abbreviated as α, N is a database containing a list of EHR data created by archetype α, and [ ] is an empty list. The database query has function type, because Eq. 38 indicates that *query p* takes a database that consists of a list of data (i.e., $[\alpha]$) and returns a result that also consists of a list of data (i.e., $[\alpha]$). The lambda term p in Eq. 39 is a user-level query that is applied to each element of a collection, as `foldr` calls itself recursively. The essential logic of query is embedded in term M. For instance,

if we define $p \equiv \lambda x{:}\alpha.(\lambda xs{:}[\alpha].(\text{if } x.id == \text{"blood pressure" then } x{:}xs \text{ else } xs))$, then *query* p N filters out the list of data for which the id field is "blood pressure".

$$database \equiv N{:}[\alpha] \tag{37}$$

$$query\ p \equiv \texttt{foldr}\ p\ [\ ]{:}[\alpha] \to [\alpha] \tag{38}$$

$$\text{where} \quad p \equiv \lambda\ x{:}\alpha.(\lambda\ xs{:}[\alpha].M){:}\alpha \to [\alpha] \to [\alpha] \quad \text{and,} \tag{39}$$

`foldr` is a higher-order combinator for recursion [30, 31]

Based on these preconditions, **Assertion 2** is formally expressed in Fig. 18. The next task is to derive the judgment $(query\ p)\ N'{:}[\alpha]$ from these premises to support **Assertion 2**.

$$\text{provided that} \quad \Gamma \vdash \alpha' <: \alpha, \text{ and} \tag{40}$$

$$\Gamma \vdash query\ p{:}[\alpha] \to [\alpha], \text{ and} \tag{41}$$

$$\Gamma \vdash N'{:}[\alpha'] \tag{42}$$

$$\text{then, derive} \quad \Gamma \vdash (query\ p)\ N'{:}[\alpha] \tag{43}$$

Figure 18: Assertion 2 formalized

2.4.2 How to introduce assignments for EHR data

Thus far, it has been impossible to update EHR data, because the record type, for example, has only the projection operation and not the assignment operation for updating (Eq. 15). Most programming languages and database systems, however, use destructive updating methods. In other words, old values are replaced by new values. Regarding this issue, an archetype is said to be mutable if a reference of the value can be obtained and updated by the assignment. Otherwise, it is said to be immutable, because the value cannot be mutated [32, p.295]. We attempted to change archetypes from immutable to mutable by introducing update functionality, and investigated what would happen to the archetype semantics.

In order to represent mutable archetypes, an additional type called the **Ref** type must be introduced. The value of the Ref type is a reference that designates the mutable locations in memory [2], described as $\mathsf{Ref}(\tau)$ (Eq. 45). If term M has type $\mathsf{Ref}(\tau)$, then M denotes a location that can hold a value of type τ. A datum can be updated by assignment operator ':=', the return type of which is Unit type (Eqs. 12 and 46).

[2]Interested readers can refer to Pierce [21, p.159] or Bruce [19, p.77] for more concrete explanations.

Types:

$$\tau ::= \ldots \tag{44}$$

$$\mid \mathsf{Ref}(\tau) \qquad \text{(type reference)} \tag{45}$$

Typing rules:

$$\frac{\Gamma \vdash M{:}\mathsf{Ref}(\tau) \qquad \Gamma \vdash N{:}\tau}{\Gamma \vdash M := N{:}\mathsf{Unit}} \qquad \text{(assignment)} \tag{46}$$

Subtyping rules:

$$\frac{\Gamma \vdash \tau' <: \tau \qquad \Gamma \vdash \tau <: \tau'}{\Gamma \vdash \mathsf{Ref}(\tau') <: \mathsf{Ref}(\tau)} \tag{47}$$

Figure 19: Reference Type

Thus, the mutable archetype itself is encoded as $\mathsf{Archetype}\{l_i{:}\mathsf{Ref}(\tau_i)\}$ [22, p.19], and the typing rule of the archetype is given by Eq. 48, which indicates that the subtyping relation of mutable Archetype depends on the subtyping of the Ref type. Note that the Ref type must be taken to be invariant in order to preserve type safety (Eq. 47). This restrictive subtyping rule is necessary because a value of type Ref can be used in a context of both reading and writing [21, p.198][22, p.29].

In the Results section , we describe a critical change in the archetype semantics that is brought about by this new encoding of the archetype.

$$\frac{\Gamma \vdash M_i{:}\mathsf{Ref}(\tau_i)}{\Gamma \vdash archetype\{l_i{:}M_i\}{:}\mathsf{Archetype}\{l_i{:}\mathsf{Ref}(\tau_i)\}} \quad (i \in 1..n) \tag{48}$$

3 Results

Based on the basic semantics of ISO 13606 archetypes , we derived two assertions (Assertions 1 and 2) in this section. Moreover, we investigated the consequences of introducing mutable archetypes and clarified the effects of this introduction on the archetype semantics.

3.1 Archetypes should follow the subsumption principle

We derived **Assertion 1**, which states that any data created via the use of a specialized archetype shall be conformant both to the specialized archetype and its parent archetype as follows. We divided this assertion into two cases: 1) any data created via an archetype shall be conformant to itself, and 2) any data created via the specialized archetype shall be conformant to its parent archetype. Assume an archetype is abbreviated as α and a specialized archetype is abbreviated as α', then we get $\alpha' <: \alpha$. Hence, the first case is straightforward

because we have a trivial equality of $M{:}\alpha = M{:}\alpha$. The second case is also straightforward. If archetypes are encoded as Record types and follow the subsumption principle, then the second case states the subsumption principle itself (cf., Eq. 27).

3.2 Archetypes should be covariant

In Section 2.2.3, we encountered three alternatives of variance when we combined archetypes by both specialization and composition. By defining the archetypes as $\mathsf{Archetype}\{l_i{:}\tau_i\}$ and combining subtyping rules for record type (Eqs. 30 and 31), we obtain the subtyping rule for archetypes as Eq. 49, which reveals that archetypes are covariant in their components.

$$\frac{\Gamma \vdash \tau_i' <: \tau_i \qquad where\ i \in 1..n+m}{\Gamma\ \vdash \mathsf{Archetype}\{l_i{:}\tau_i'\}_{i\in 1..n+m} <: \mathsf{Archetype}\{l_i{:}\tau_i\}_{i\in 1..n}} \tag{49}$$

Once the variance of archetypes is fixed as covariant, this affects not only the underlying semantics but also the syntax of the archetype slot. Suppose that an archetype composes another archetype with a regular expression /openEHR-EHR-CLUSTER.device.*/ in the archetype slot and that its specialized archetype composes the other archetype with `/*/`. In this case, any data, even if the data are a supertype of the CLUSTER-device, can be set to the field, which violates the covariant requirement. Therefore, we claim that every composed archetype should be uniquely specified in the archetype slot by an exact archetype identifier, such as that shown in Fig. 20. In addition, we do not have to explicitly exclude improper archetypes, because they are implicitly excluded if they are not subtypes of the specified archetype.

```
ITEM_TREE[at0011] matches {
    items matches {
        allow_archetype matches {
            include
                archetype_id/value matches {"openEHR-EHR-CLUSTER.device.v1"}
        }
    }
}
```

Figure 20: An example of a uniquely identified archetype slot
In this fragment of ADL, the archetype 'openEHR-EHR-CLUSTER.device.v1' is identified as a string in the archetype slot, and so is uniquely specified.

3.3 Queries should be reusable

Assertion 2 insists that queries should be reusable. We define a number of archetypes at a certain point in time, and the data are stored in the EHR

system based on the defined archetypes. As time passes, some of the archetypes are specialized to create more detailed archetypes, as in Fig. 2. New EHR data, which are conformant to the specialized archetypes, will accumulate in the system. Then, it is natural to want to run the queries defined on the first generation archetypes on the EHR data that are conformant to their specialized archetypes. This is what Assertion 2 demands.

In Section 2.4.1, we formalized Assertion 2 in terms of typed lambda calculus, as shown in Fig. 18. The formalized statement of the assertion was used to derive $(query\ p)\ N'{:}[\alpha]$ from the assumptions $\alpha' <: \alpha, query\ p{:}[\alpha] \to [\alpha]$, and $N'{:}[\alpha']$. Note that α is a parent archetype, α' is the specialized archetype of α, and $N'{:}[\alpha']$ refers to "a list of data created by the specialized archetype α'". This is case A in Fig. 21, because we are applying a query for a parent archetype α to the data created from its specialized archetype α'. Given these conditions, we constructed the following derivation tree to support Assertion 2 (Fig. 22).

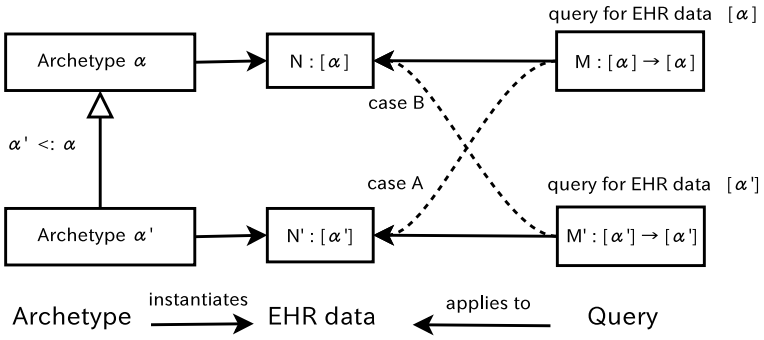

Figure 21: Two cases of query applications

This figure describes the relationships between archetypes, EHR data, and their queries. Four arrows are drawn from queries to EHR data. The two solid arrows indicate ordinary queries, the argument type of which is matched to the type of the EHR data (e.g., $M{:}[\alpha] \to [\alpha]$ and $N{:}[\alpha]$). The two dotted arrows indicate the cases for which we investigated the validity in the text.

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash \alpha' <: \alpha}{\Gamma \vdash [\alpha'] <: [\alpha]}\ [34] \qquad \Gamma \vdash N'{:}[\alpha']}{\Gamma \vdash N'{:}[\alpha]}\ [27] \qquad \Gamma \vdash query\ p{:}[\alpha] \to [\alpha]}{\Gamma \vdash (query\ p)\ N'{:}[\alpha]}\ [7]$$

Figure 22: Derivation of Assertion 2

The significance of this derivation is not so clear. Therefore, we presented the opposite case of applying a query for the specialized archetype α' to the data from the parent archetype α. This assumption corresponds to case B in

Fig. 21. In this case, the query is encoded as $M'{:}[\alpha'] \to [\alpha']$, and the target database is encoded as $N{:}[\alpha]$. Then, it is obvious that $(M'\ N)$ causes a type error, because $[\alpha']$ does not match $[\alpha]$ (cf., Eq. 7). This indicates that the queries based on the parent archetype are applicable to the data created by its specialized archetype whereas queries based on a specialized archetype are not applicable to data created with the parent archetypes.

3.4 Archetypes should be invariant if they are mutable schema

Section 2.4.2 revealed that if we want to update EHR data, we must encode Archetype as $\mathsf{Archetype}\{l_i{:}\mathsf{Ref}(\tau^i)\}$ as in Eq. 48. Therefore, we need to determine how to derive the judgment $\mathsf{Archetype}\{l{:}\mathsf{Ref}(\tau')\} <: \mathsf{Archetype}\{l{:}\mathsf{Ref}(\tau)\}$ based on the modified type system.

Since subtyping of Ref types is defined in Eq. 47, we derived the tree shown in Fig. 23 by combining the mutable Archetype judgment and the Ref subtyping rules.

$$\dfrac{\dfrac{\tau' <: \tau \qquad \tau <: \tau'}{\mathsf{Ref}(\tau') <: \mathsf{Ref}(\tau)}\ [47]}{\mathsf{Archetype}\{l{:}\mathsf{Ref}(\tau')\} <: \mathsf{Archetype}\{l{:}\mathsf{Ref}(\tau)\}}\ [49]$$

Figure 23: Derivation of the mutable archetype

Figure 23 shows that subtyping of mutable archetypes depends on the type of τ and τ'. However, $\tau' <: \tau$ and $\tau <: \tau'$ means that τ is equal to τ', because subtyping operator $<:$ is both transitive and reflexive (Eqs. 28 and 29). Therefore, the derivation of Fig. 23 reveals that archetypes must be invariant if they are mutable archetypes.

Since this consequence is less intuitive, it is demonstrated through a counterexample. Based on Eq. 50, let us assume that mutable archetypes are covariant. Then, we identify an inconsistency from this false hypothesis.

Typing rules:

$$\dfrac{\Gamma \vdash \tau_i' <: \tau_i \qquad where\ i \in 1..n+m}{\Gamma\ \vdash \mathsf{Archetype}\{l_i{:}\mathsf{Ref}(\tau_\mathsf{i}')\}_{i\in 1..n+m} <: \mathsf{Archetype}\{l_i{:}\mathsf{Ref}(\tau_\mathsf{i})\}_{i\in 1..n}} \quad (50)$$

Provided that type Int is integer and type PosInt is a positive integer, we assert that $\mathsf{PosInt} <: \mathsf{Int}$ by ad hoc rule, and suppose that $\mathsf{Archetype}^\mathsf{A}\{l{:}\mathsf{Ref}(\mathsf{PosInt})\}$ and $\mathsf{Archetype}^\mathsf{B}\{l_1{:}\mathsf{Ref}(\mathsf{Int})\}$.

Then, since we assume the mutable Archetype to be covariant, $\mathsf{Archetype}^\mathsf{A} <: \mathsf{Archetype}^\mathsf{B}$ should hold (Eq. 50). In addition, let us define $a \equiv archetype\{l =$

$1\}$:$\mathsf{Archetype}^{\mathsf{A}}$ and $b \equiv archetype\{l = -1\}$:$\mathsf{Archetype}^{\mathsf{B}}$. Given these settings, we can derive $a.l := -1$ as in Fig. 24. This, however, contradicts $a.l$:$\mathsf{Ref}(\mathsf{PosInt})$, because -1 is *not* a positive integer.

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{PosInt} <: \mathsf{Int}}{\mathsf{Archetype}^{\mathsf{A}} <: \mathsf{Archetype}^{\mathsf{B}}}\,[50] \qquad a{:}\mathsf{Archetype}^{\mathsf{A}}}{a{:}\mathsf{Archetype}^{\mathsf{B}}}\,[27]}{a.l{:}\mathsf{Ref}(\mathsf{Int})}\,[15] \qquad -1{:}\mathsf{Int}}{a.l := -1{:}\mathsf{Unit}}\,[46]$$

Figure 24: Derivation from the false hypothesis

As described above, if we allow EHR data to be updatable, then the semantics of the archetype is modified, and the variance of archetypes is changed from covariant to invariant.

3.5 Archetypes should be both covariant and immutable

Thus far, we have obtained different consequences of archetype variance from different premises. If the archetype is immutable, then the archetype is covariant, otherwise the archetype is invariant. Both consequences are valid in terms of type theory. As previously explained in Section 2.1, we need higher concepts than the type theory to decide which consequence is more appropriate. These higher concepts were 'structural interoperability' and 'schema extensibility'.

For the case in which the archetype is covariant, we can construct a specialized archetype along with its composition. Looking back at the aforementioned schema in Fig. 2, it should be noted that the schema is composed in a completely covariant manner. When we specialize the RevisedCase from the InitialCase, we can also specialize its composition outcome from InitialOutcome to RevisedOutcome. For the case in which the archetype is invariant, we cannot specialize the components of the parent archetypes. Although structural interoperability is still guaranteed in this case, schema extensibility is severely restricted. Comparing these two alternatives, we conclude that, for the sake of schema extensibility, the archetype should be immutable.

3.6 EHR data are safe

EHR data are created, validated, and queried based on their corresponding archetypes. We defined the basic semantics of archetypes - in particular variance and immutability of archetypes - in terms of the type system, and we clarified the behaviors of the semantics in terms of the derivations. However, we only inferred these derivations by symbolic manipulations referred to collectively as deduction and did not perform any evaluation such as queries or projections. Finally, a simple question remains: Aren't there any discrepancies between the

results from the deductions and the values of the EHR data from the actual evaluation results? The answer lies in a property of the type theory.

A carefully constructed type system can take advantage of a property called **soundness**. If a type system is sound, then every derived judgment in the system is assured to be valid, so that the system is free from type errors [33, 22]. This is the fundamental requirement of structural interoperability, because if you cannot perform a computation on some data but obtain an error, the EHR system can never be operated safely.

The soundness is stated in a more formal manner [22, p.12] by Eq. 51, which indicates that if M is an instance type τ (i.e., $M{:}\tau$), then M will be evaluated to obtain a value that has an inferred type of τ. Note that $[\![\ ]\!]$ is a function that evaluates a lambda term to obtain its value when applied to the term or infers the type of the term to obtain its simpler type when applied to the type itself.

$$\text{if } \ \Gamma \vdash M{:}\tau \ \text{ is valid, then } \ [\![M]\!] \in [\![\tau]\!] \text{ holds.} \tag{51}$$

Note that the aforementioned type system is sound [34, 35], i.e., Eq. 51 holds for every component of the proposed type system. We omit the soundness proof, because it is beyond the scope of the present paper [3]. Instead, we present an example in which the soundness holds.

Suppose term M is a projection function from $\mathsf{Archetype}\{l{:}\mathsf{TEXT}\}$ to TEXT (Eq. 52) and N is an instance of that archetype (Eq. 53). Then, the evaluation of the term $(M\ N)$ produces the value "text" from Eq. 54. On the other hand, when we infer the type of (M N), we obtain TEXT (Fig. 26). Looking at the results of evaluation in both the term and the type, it is obvious that "$text$" $\in \mathsf{TEXT}$ holds.

$$\text{provided that,} \quad M \equiv \lambda x{:}\mathsf{Archetype}\{l{:}\mathsf{TEXT}\}.(x.l){:}\mathsf{Archetype}\{l{:}\mathsf{TEXT}\} \to \mathsf{TEXT} \tag{52}$$

$$N \equiv archetype\{l = \text{"}text\text{"}\}{:}\mathsf{Archetype}\{l{:}\mathsf{TEXT}\} \tag{53}$$

$$\text{then,} \quad [\![(M\ N)]\!] \Rightarrow_\beta \text{"}text\text{"} \tag{54}$$

where $\Rightarrow_\beta$ is called β-conversion that does $(\lambda x.M)N \Rightarrow_\beta M[x := N]$

Figure 25: Evaluation of a lambda term $(M\ N)$

$$\frac{M{:}\mathsf{Archetype}\{l{:}\mathsf{TEXT}\} \to \mathsf{TEXT} \qquad N{:}\mathsf{Archetype}\{l{:}\mathsf{TEXT}\}}{(M\ N){:}\mathsf{TEXT}} \quad [7]$$

Figure 26: Type inference of a lambda term $(M\ N)$

[3] Interested readers can refer to Fisher [36] or Wright [35] for a complete reference on the proof.

Although this is a small example, Eq. 51 holds for every component, even for larger components such as EHR data, as long as the system is sound. Then, we are justified in saying that the proposed sound type system guarantees the conformance of EHR data to their archetypes and that EHR data are safe in terms of type theory.

4 Discussion

In the present paper, we assigned major components of ISO 13606 to their proper types and derived multiple results for variance from different premises of mutability, using a deductive system of the type theory. As a result, we reached the conclusion that the archetype should be a covariant and immutable schema. Furthermore, we redefined the method of composing an archetype as uniquely designating the target archetype in an archetype slot.

In this section, we explain the importance of formal semantics, and discuss the assets and limitations of typed lambda calculus as compared with other formal methodologies that have been used in related research.

4.1 Why formal semantics is needed

The primary reason we could reach the above conclusion is that we built the basic formal semantics of ISO 13606. However, some readers might suspect that the primary subject of the standard is message modeling of EHR Extract, and such a conceptualized semantics is considered to be beyond the scope of the standard. Indeed, the purpose of ISO 13606 is stated as follows:

> ISO 13606 may offer a practical and useful contribution to the design of EHR systems but will primarily be realized as a common set of external *interfaces* or *messages* [2, v].

Despite this fact, we insist that semantics is an indispensable part of the standard and should be thoroughly investigated.

In informatics, the term 'message' implies a grammatical arrangement of words rather than a substantial internal structure or behaviors. The validity of messages, however, depends not only on the grammar but also the semantics: an underlying principle for the internal structure and behavior. For example, "Plato loved Socrates" is a sensible sentence, whereas "Friday hates Saturday" is not, even though this sentence is grammatically correct. In other words, messages must conform to semantics, as well as to grammar. In these examples, the semantics is a constraint that restricts the domain of the function 'love' or 'hate' to human beings. It is semantics that underpins the validity of messages.

Semantics, despite its significance, is invisible, and thus appears unfamiliar. We can literally see the phrase "Plato loved Socrates" as an alphabetic sequence, but the function of 'love' itself, let alone its domain, is unclear. However, once we formalize the semantics, they can be visualized. When every term and rule is defined as a symbol, then inference can be seen as an arrangement of

symbols, just as a sentence can be seen as an arrangement of words. Then, we can easily explore the semantics: the inference mechanism rigorously excludes invalid results and derives novel results that could never be foreseen. In terms of the degree to which they are conceptualized, this mechanism is similar to the laws of physics, which capture the true nature of the world as mathematical symbols and greatly expand the boundaries of our knowledge.

4.2 Implication of formal semantics

The purpose of building the system of physics is to model the natural phenomena in a general and reproducible manner so that future phenomena, such as solar eclipses, can be predicted. Then, for what purpose is our formal semantics built? If there is no practical purpose, then the present study is merely an empty manipulation of abstract symbols, and there will be no contribution to medical informatics.

As previously stated in Section 3.6, the purpose of these formal semantics is the analysis of the ISO 13606 specification, so that only safe operations on EHR data are permitted and all erroneous operations are denied in terms of type system. In contrast, if data or type safety is not assured in the specification, then the EHR system will cause errors in the long run. The disaster of NASA's Mars Climate Orbiter in 1999 is a good example of such a failure of data management, because the root cause was a "failure to use metric units in the coding of a ground software file" [37]. A numerous system failures ,which rooted in Java's null pointer exceptions, has been occurring in the course of ordinary operations on earth. This kind of runtime exception occurs when the null value reference [4] is accessed. We can get rid of these errors in advance if we introduce proper types for each value based on a sound type system. Therefore, the implication of the present study, which is building engines for next-generation EHR systems devoid of data management failures, is rather practical.

4.3 Assets of typed lambda calculus

We used typed lambda calculus as the formalization methodology, and there are several reasons why we chose this formalism.

First, typed lambda calculus has a long history of research in computer science as the computational model of statically typed programming languages. The theory laid the ground for not only typed functional languages, such as ML and Haskell, but also for object-oriented languages, such as Java [8, 23]. Second, type theory as a formalized deductive system is useful for proof-theoretic investigations [38, p.9]. We can construct proofs of already defined typed rules by properly combining the rules. Furthermore, the validity of the proofs depends only on the syntactic aspects thereof. This property of the theory helps us to check the proof, as we did in the present paper. Third, the soundness of the type system ensures the validity of the consequences derived from the proof. If

[4] The null value is a reference that does not point to anything.

a value is proved to belong to a type, then the value must be exactly that type. This indicates that we can avoid erroneous operations of an EHR system that is built upon a sound archetype repository.

Thus, typed lambda calculus guarantees a certain type of interoperability - we call it structural interoperability - based on these properties. The knowledge and disciplines are so abstract and general that the results can be applied not only to ISO 13606, but also to every other standard in pursuit of the same goal.

4.4 Related research

We found two basic types of related research on formal semantics of archetypes. One type uses regular grammar theory [13, 11], and the other type uses Description Logic (DL) [39, 40, 41, 12, 42].

Maldonado et al. [13, 11] attempted to construct formal semantics of archetypes by means of regular grammar theory. They abstracted archetypes as a tree with labeled nodes and developed a type system based on the concept of a constrained multiplicity list. They chose this approach because the functionality of archetypes resembles that of XML Schema, and the use of the regular grammar theory has been successful in building a formal theory of XML Schema [43, 44]. These papers explained the semantics of archetype specialization and conformance of EHR data but did not investigate the issues of archetype variance or mutability as we did in the present study.

Several studies took a different approach, such as transferring archetypes to OWL, and these studies used DL, which is widely used as a theoretical foundation of ontology. Whereas most of the studies [39, 40, 41, 12, 42] simply translated archetypes to OWL, Marcos et al. [45] performed DL reasoning to check the consistency of archetypes, and they found out that 22.2% of the archetypes in the Clinical Knowledge Manager (CKM) repository and 21.2% of the archetypes in the National Health Service (NHS) repository were inconsistent. The study, however, has a weaknesses. Although they took into account compositions through archetype slots in checking the consistency of specialization, it was unclear which variance of archetypes was assumed to be true in the reasoning. It was presumed to be covariant only after the following citation was found: "The specialized archetype uses the datatype DV_TEXT instead of CODED_TEXT[5]" [45, p.312]. In addition, they provided no theoretical ground for this assumption, so that they had no choice but to overlook the issue of archetype variance or immutability that was intensively analyzed by typed lambda calculus in our study.

As a matter of fact, intuitionistic logic and typed lambda calculus are deeply related as indicated by the Curry-Howard isomorphism [47, 48]. The main idea here is that a logical formula can be interpreted as types in a type theory, and the proof of a formula is associated with a lambda term of the proper type. As a result, if a formula is derivable in a logical system, then the corresponding

[5]Note that CODED_TEXT is a subtype of DV_TEXT (i.e., CODED_TEXT <: DV_TEXT) in openEHR specification [46, p.21].

type is inhabited in the type theory [49]. This correspondence also seems to hold between DL and a type theory, because some DLs can be thought of as notational variants of certain propositional modal logics [50, p.38][51], and a certain type theory is known to have the same expressiveness as modal logic [52].

Even though there is some correspondence between DL and type theory, we insist that type theory is a more suitable methodology than DL for the purpose of the present study. Being a logical framework for knowledge representation, DL is capable of checking the consistency of defined semantics, but is incapable of deriving theorems from axioms and rules. On the other hand, type theory is a theory, the formal definition of which is a set of formulas that is closed under logical consequence [53, p.32]. These theories are constructed by selecting a set of formulas called axioms (e.g., typing rules) and deducing their logical consequences, which are called theorems (i.e., derivation trees). In investigating a formal semantics of a specification, it is crucial to define axioms and to derive the logical consequences of the axioms, i.e., theorems, in order to determine whether these axioms and theorems fit with the requirements of the specification. We chose type theory instead of DL, because the chosen theory is more suitable for the investigation of semantics.

4.5 Limitations

We chose typed lambda calculus as a formal methodology and have clarified some important aspects of archetype semantics that have been overlooked in previous studies. Despite these achievements, there are some limitations in this formalism. Specifically, whether type theory, which is inherently conservative due to its soundness, can cover the entire range of ISO 13606 archetype semantics remains unknown. Moreover, there are some difficult issues in encoding Archetype.

First, variant types prevent a schema from being extended, because its subtype has fewer choices than its supertype (Eq. 32). Suppose that we wanted to store a history of hepatitis and, therefore, developed an archetype in which the choice of hepatitis is expressed as a variant type. Only hepatitis A and B were known to us at the time the archetype was developed. Therefore, we encoded the two alternatives as CODED_TEXT.defining_code (Fig. 13). Later, a problem was encountered when hepatitis C was discovered. Since the subtype of a variant type has fewer choices than the supertype (Eq. 32), we cannot specialize the existing archetype. In order to avoid this inconvenience, we must define a new archetype that contains three types of hepatitis, namely, hepatitis A, B, and C. However, this approach does not involve extending the schema, but rather creating an additional schema, which is not reusable for existing queries, as shown in Fig. 22.

Second, it is difficult to encode an archetype model in which fields are parametrized by the RM type. Even though we have shown how to express Archetype in typed lambda calculus, we intentionally ignored the fact that the field name of Archetype is partly dependent on the type of RM [6, p.63] and

instead hard-coded the structure as a Record type. For example, archetypes having an RM type of SECTION, such as CEN-EN13606-SECTION.substance_use, should have a members field, whereas an archetype having an RM type of ENTRY, such as CEN-EN13606-ENTRY.substance_use, should have items and meaning fields. These fields cannot be parametrized according to the defined type system. Even if we attempt to introduce higher-order types [22, p.24], such attempts fail because the higher-order types parametrize only types and not the associated field names. This type of parametrized archetype is thought to be very difficult to encode in any existing formalisms, including the regular grammar theory or DL.

Third, we did not discuss attribute constraints, such as cardinality, existence, occurrences, order, unique, and multiple-valued attributes [6, p.83-84]. The cardinality and occurrences constraints are difficult to handle in this simple type theory, because they are expressed as intervals, such as $\{0..1\}$ or $\{10..*\}$. In the case of multiple-valued attributes, the constraints are also difficult to handle. The multiple-valued attributes are constrained by an instance of C_MULTIPLE_ATTRIBUTE, which allows multiple co-existing member objects of the container value of the attribute to be constrained [6, x]. Figure 27 shows an example of multiple-valued attributes. This *parts* attribute will be encoded as a List type, but its element type cannot be determined, because it contains two items.

```
definition
  INSTRUMENT[at0000] matches {
    size matches {|60..120|}
    date_of_manufacture matches {yyyy-mm-??}
    parts cardinality matches {0..*} matches {
      PART[at0001] matches { -- neck
         material matches {
            [local::at0003]  -- timber
         }
      }
      PART[at0002] matches { -- body
         material matches {
            [local::at0003]  -- timber
         }
      }
   }
}
```

Figure 27: Excerpt of the guitar archetype [6, xvi]

4.6 On Exactitude in Science

As stated in the previous section, the present study has several limitations, and therefore we cannot claim to have achieved full formal semantics of ISO

13606 based on the proposed type system. Nevertheless, we insist that building a formal semantics of archetypes, even if it is incomplete, is of fundamental importance.

The activity of modeling or abstraction is to focus on the essential part of the subject while discarding the subtle details for some purposes. For example, a map that describes only highways and ignores small lanes can be useful for specific purposes, such as driving between big cities. On the other hand, a map that describes every detail of geographical information will become gigantic, and may be useless as a map (c.f., Jorge Luis Borges's "On Exactitude in Science" [54]). A more suitable example in computer science is Featherweight Java (FJ) [8], which is a formal semantics of Java's type system based primarily on typed lambda calculus. Even though the semantics did not cover the full semantics of the Java programming language, it provided a significant boost to the enrichment of the language with advanced features, such as generic class or inner class [21, p.246]. It is often meaningful to build a formal semantics that is less complete but more compact, offering sufficient insight for minimum investment.

In the analysis of the present study, we have dealt with the type constraint, omitting attribute constraints, as described above. While our formal semantics was less complete, we obtained important insights on the variance and mutability of archetypes. Moreover, we cannot add new types or typing rules that would contradict the already defined type system, otherwise the soundness of the system will be lost. Therefore, as long as we take our type theory as a solid basis for the foundation of archetype semantics, these insights are expected to be valid even if we enrich the proposed type system to handle other constraints. Returning to the map metaphor, although we can draw a more detailed map by adding small lanes, these lanes would never override existing highways.

4.7 Prospectives

As mentioned above, the proposed type theory did not cover the entire range of ISO 13606 archetype semantics, because the type theory we introduced was too simple. We believe that the coverage of the proposed theory can be widened if we add more type rules or extend the type system by adding more advanced features.

First, bulk data structures, such as a list or a set, in ISO 13606 are parametrized according to the type of its elements, as in modern programming languages [2, p.42]. These types can be correctly captured, if we introduce a recursive type to the proposed system [22, 24].

Second, we did not discuss attribute constraints as stated previously. The order and unique constraints, however, can be handled together so that we can determine a suitable type for each container. As the specification indicates [6, p.84], the proper collection type can be assigned according to the combination of order and unique constraints. An existence constraint can be also handled if we introduce the Option type, which represents optional values [21, p.137].

Third, although ISO 13606 attempts to represent every conceivable type of health record data structure in a consistent manner [2, vi], the unit of mea-

surement is a major exception, because the units are expressed only as string representations, e.g., kg or mmHg [6, p.46,p.82], not as a structured data type. This makes it difficult to properly convert or calculate EHR data according to their units. Type theory can address this problem when the units are constructed as a type [55, 56]. Once we develop that kind of type system with sound semantics, we can manipulate EHR data in a more consistent manner, for example, by automatic unit conversion or dimension-based validation.

These examples suggest that if we add more advanced features, we can enrich the semantics and extend the interoperability of ISO 13606 even beyond the reach of the original scope of the standard. In the future, we intend to encode and analyze the semantics of the ISO 13606 archetype with the latest results for the theory.

5 Conclusions

For ISO 13606, a solid foundation is necessary for the archetype semantics in order to achieve structural interoperability between EHR systems. We analyzed the semantics by a formal methodology based on typed lambda calculus and clarified the following:

(i). Archetypes should be both covariant and immutable schema, so that the structural interoperability is guaranteed and EHR schema are extended over time.

(ii). An archetype should uniquely specify other archetypes in the archetype slot when they are composed.

We hope that the results of the present study will provide all who are involved in ISO 13606 with a measure of insight and suggestions for the renewal tasks of the standard.

Acknowledgments

The authors would like to thank Ms. Mirai Ikebuchi for her advice on mathematical notation and for checking our derivations and Mr. Yasuhide Miura for his assistance in preparing the manuscript.

The present study was supported in part by the Japan Society for the Promotion of Science (JSPS) through the Funding Program for World Leading Innovative R&D on Science and Technology (FIRST program).

References

[1] Sujansky W. Heterogeneous database integration in biomedicine. J Biomed Inform. 2001;34(4):285–298.

[2] 13606-1 Health informatics Part 1: Reference model; 2008. http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail .htm?csnumber=40784 [accessed 1.14].

[3] Beale T, Heard S, editors. openEHR Architecture : Architecture Overview; 2008. http://www.openehr.org/releases/1.0.2/architecture/overview.pdf [accessed 1.14].

[4] Schloeffel P, Beale T, Hayworth G, Heard S, Leslie H. The Relationship between CEN 13606, HL7, and openEHR. In: HIC 2006 and HINZ 2006; 2006. p. 24–28.

[5] Chen R, Klein GO, Sundvall E, Karlsson D, Ahlfeldt H. Archetype-based conversion of EHR content models: pilot experience with a regional EHR system. BMC Med Inform Decis Mak. 2009;9(1):33.

[6] 13606-2 Health informatics Part 2: Archetype interchange specification; 2008. http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail .htm?csnumber=50119 [accessed 1.14].

[7] Ohori A. A Simple Semantics for ML Polymorphism. University of Pennsylvania; 1989.

[8] Igarashi A, Pierce BC, Wadler P. Featherweight Java: a minimal core calculus for Java and GJ. ACM Trans Program Lang Syst. 2001;p. 396–450.

[9] Veltman K. Syntactic and Semantic Interoperability: New Approaches to Knowledge and the Semantic Web. The New Review of Information Networking. 2001;7:159–184.

[10] Garde S, Knaup P, Hovenga E, Heard S. Towards Semantic Interoperability for Electronic Health Records. Methods of Inf Med. 2007;46(3):332–343.

[11] Maldonado JA, Moner D, Tomás D, Ángulo C, Robles M, Fernández JT. Framework for Clinical Data Standardization Based on Archetypes. Stud Health Technol Inform. 2007;129(Pt 1):454–458.

[12] Martínez-Costa C, Menárguez-Tortosa M, Fernández-Breis JT. An approach for the semantic interoperability of ISO EN 13606 and OpenEHR archetypes. J Biomed Inform. 2010;43(5):736–746.

[13] Maldonado JA, Moner D, Boscá D, Fernández-Breis JT, Angulo C, Robles M. LinkEHR-Ed: a multi-reference model archetype editor based on formal semantics. Int J Med Inform. 2009;78(8):559–570.

[14] Hakimpour F, Geppert A. Resolution of Semantic Heterogeneity in Database Schema Integration Using Formal Ontologies. Inf Technol and Management. 2005;6(1):97–122.

[15] Rector AL. Clinical Terminology: Why is it so hard? Methods of Inf Med. 1999;38(4):239–252.

[16] Renner SA, Rosenthal AS, Scarano JG. Data Interoperability: Standardization or Mediation. In: IEEE Metadata Workshop, Silver Spring MD; 1996. .

[17] Beale T, editor. The openEHR Archetype Model : Archetype Object Model; 2007. www.openehr.org/releases/1.0.1/architecture/am/aom.pdf [accessed 1.14].

[18] Beale T, Heard S, editors. The openEHR Archetype Model : Archetype Definition Language, ADL 1.4; 2007. http://www.openehr.org/releases/1.0.1/architecture/am/adl.pdf [accessed 1.14].

[19] Bruce KB. Foundations of Object-Oriented Languages. MIT press; 2002.

[20] Castagna G. Covariance and contravariance: conflict without a cause. ACM Transactions on Programming Languages and Systems. 1995;17:431–447.

[21] Pierce BC. Types and Programming Languages. 1st ed. MIT press; 2002.

[22] Cardelli L. Type Systems. In: Handbook of Computer Science and Engineering. CRC Press; 2004. .

[23] Barendregt H, Dekkers W, Statman R, editors. Lambda Calculus with Types. 1st ed. Perspectives in Logic. Cambridge University Press; 2013.

[24] Geuvers H. Introduction to Type Theory. In: Language Engineering and Rigorous Software Development. vol. 5520 of Lecture Notes in Computer Science. Springer Berlin Heidelberg; 2009. p. 1–56.

[25] Craig LD. Object-Oriented Programming Languages. Springer-Verlag London; 2007.

[26] Hillebrand GG, Kanellakis PC, Mairson HG. Database Query Languages Embedded in the Typed Lambda Calculus. Inform Comput. 1996;127(2):117–144.

[27] Grust T. Comprehending Queries. University of Konstanz; 1999.

[28] Grust T, Scholl MH. How to Comprehend Queries Functionally. J Intell Inf Syst. 1999;12(2-3):191–218.

[29] Poulovassilis A, Small C. Algebraic query optimisation for database programming languages. The VLDB Journal. 1996;5(2):119–132.

[30] Hutton G. A tutorial on the universality and expressiveness of fold. J Funct Programming. 1999;9(4):355–372.

[31] Bird R. Introduction to Functional Programming. Prentice Hall; 1992.

[32] Harper R. Practical Foundations for Programming Languages. 1st ed. Cambridge; 2013.

[33] Mitchell JC. Foundations for Programming Languages. 1st ed. MIT Press; 2000.

[34] Gunter CA. Semantics of Programming Languages. 1st ed. Foundations of Computer Series. MIT Press; 1992.

[35] Wright A, Felleisen M. A Syntactic Approach to Type Soundness. Inform Comput. 1994;115(1):38–94.

[36] Fisher K, Honsell F, Mitchell JC. A Lambda Calculus of Objects and Method Specialization. Nordic J of Computing. 1994;1(1):3–37.

[37] Stephenson AG, LaPiana LS, Mulville DR, Rutledge PJ, Bauer FH, Folta D, et al. Mars Climate Orbiter Mishap Investigation Board: Phase I Report. NASA; 1999.

[38] Program TUF, editor. Homotopy Type Theory: Univalent Foundations of Mathematics. first-edition-323-g28e4374 ed. The Univalent Foundations Program Institute for Advanced Study; 2013.

[39] Martínez-Costa C, Menárguez-Tortosa M, Fernández-Breis JT, Maldonado JA. A model-driven approach for representing clinical archetypes for Semantic Web environments. J Biomed Inform. 2009;42(1):150–164.

[40] Maldonado JA, Costa CM, Moner D, Menárguez-Tortosa M, Boscá D, Giménez JAM, et al. Using the ResearchEHR platform to facilitate the practical application of the EHR standards. J Biomed Inform. 2012;45(4):746 – 762.

[41] Lezcano L, Sicilia MA, Solano CR. Integrating reasoning and clinical archetypes using OWL ontologies and SWRL rules. J Biomed Inform. 2011;44(2):343 – 353.

[42] Costa CM, de Andrade AQ, Karlsson D, Kalra D, Schulz S. Towards the harmonization of clinical information and terminologies by formal representations. European Journal for Biomedical Informatics. 2012;8(3):3–10.

[43] Chidlovskii B. Using Regular Tree Automata as XML schemas. In: Proc. IEEE Advances on Digital Libraries Conference; 2000. p. 89–98.

[44] Murata M, Lee D, Mani M, Kawaguchi K. Taxonomy of XML Schema languages using formal language theory. ACM Trans Internet Technol. 2005;5(4):660–704.

[45] Menárguez-Tortosa M, Fernández-Breis JT. OWL-based reasoning methods for validating archetypes. J Biomed Inform. 2013;46(2):304–317.

[46] Beale T, Heard S, Kalra D, Lloyd D. The openEHR Reference Model: Data Types Information Model; 2008. http://www.openehr.org/releases/1.0.2/architecture/rm/data_types_im.pdf.

[47] Lawler F. Classical Logic and the Curry-Howard Correspondence. Trinity College Dublin; 2008.

[48] Sørensen MHB, Urzyczyn P. Lectures on the Curry-Howard Isomorphism; 1998.

[49] Asperti A, Longo G. CATEGORIES TYPES AND STRUCTURES: An Introduction to Category Theory for the working computer scientist. FOUNDATIONS OF COMPUTING SERIES. MIT Press; 1991.

[50] Baader F, Calvanese D, McGuiness DL, Nardi D, Patel-Scheneider PF, editors. The Description Logic Handbook. 2nd ed. Cambridge; 2007.

[51] Rijke MD. Modal Logics and Description Logics. In: Proc. DL'98; 1998. p. 1–3.

[52] Nanevski A, Pfenning F, Pientka B. Contextual Model Type Theory; 2005.

[53] Ben-Ari M. Mathematical Logic for Computer Science. 3rd ed. Springer-Verlag; 2012.

[54] Borges JL. Collected fictions of Jorge Luis Borges. Penguin Books; 1999.

[55] Kennedy A. Dimension Types. In: Programming Languages and Systems — ESOP '94. vol. 788. Springer Berlin Heidelberg; 1994. p. 348–362.

[56] A Kennedy. Types for Units-of-Measure: Theory and Practice. In: Central European Functional Programming School. vol. 6299. Springer Berlin Heidelberg; 2010. p. 268–305.