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

Akimichi Tatsukawa[a,*], Emiko Shinohara[b], Takeshi Imai[a], Yoshimasa Kawazoe[a], Kazuhiko Ohe[a]

[a]*Department of Planning, Information and Management, The University of Tokyo Hospital, Tokyo, Japan.*
[b]*Center for Disease Biology and Integrative Medicine, Graduate School of Medicine, The University of Tokyo, Tokyo, Japan.*

Abstract

ISO 13606 defines a standard specification for exchange and integration of electronic health record (EHR) data with the goal of the interoperability of EHR data across heterogeneous systems. In order to achieve this goal, the standard provides a dual-model architecture in which archetypes are extensible schemas of EHR data via their specialization and composition mechanism. However, the standard lacks 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. Our results indicate that the archetypes should be covariant and immutable schemas in order to guarantee both the interoperability of EHR data and the extensibility of archetypes.

Keywords: ISO 13606, Archetypes, Typed Lambda Calculus, Formal Semantics, openEHR

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. Heterogeneity between various local EHR systems is a major reason why interoperabilities such as the construction of a central repository or an exchange of EHR data are not possible. In order to achieve interoperability of EHR data from local systems in a consistent and longitudinal manner, the schema of EHR data must satisfy the requirements of various organizations and evolve over time in accordance with the development of medical concepts.

*Corresponding author. Address: Department of Planning, Information and Management, The University of Tokyo Hospital, 7-3-1 Hongo, Bunkyo-ku, Tokyo, Japan, Fax: +81 3 3813 7238
Email address: akimichi.tatsukawa@gmail.com

ISO 13606 defines a common healthcare model as an international standard. 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"[1, vi] so as to "support the interoperability of systems and components that need to communicate EHR data"[1, 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 concepts. In order to overcome these difficulties, ISO 13606 has adopted a dual-model approach that is based on the separation of concepts in two levels: (1) information and (2) knowledge. Knowledge is represented by an archetype, which is formal model of a clinical concept[2, p.158].

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 the openEHR specification is more appropriate, we will refer to this specification.

1.2. Objective

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 definition makes implement an EHR system based on ISO 13606 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?

Since ISO 13606 does not provide a sufficient 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.

Assertion 1 Any data created via the use of a specialized archetype shall be conformant both to it and to its parent[6, viii].

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?

The goal of the present study is to clarify the ISO 13606 archetype semantics by means of a formal methodology in order to support the above assertions. We will investigate the semantics in a formal manner under the assumption that at least these two assertions must hold and will 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 a theory that 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

Before introducing the formal methodology of typed lambda calculus, we begin by explaining several concepts that are fundamental to the exchange and integration of EHR data. These concepts are important because obtaining a final decision from different results based on a formal reasoning requires conceptual criteria that are higher than formalization itself. In other words, formalization is the necessary requirement for building a solid semantics, but it is not sufficient. When we encounter different results from different premises, we have to choose either one of the results even though these results are both valid in terms of formalization. In order to achieve EHR data integration, these criteria 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. However, definitions of interoperability in the standardizations vary between the standardizations. ISO 13606 defines 'interoperability' as the "ability for data shared by systems to be understood at the level of fully defined domain concepts"[1, 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 perform computations. Moreover, we believe that the root of this ambiguity is due to the fact that two different types of concepts are being explained using one terminology. Thus, we divided the concept of interoperability into two levels, as follows.

- Structural Interoperability

  Structural interoperability is defined as the ability to process data based on a shared formal schema and the sound semantics of the data. 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.

- Semantic Interoperability

  Semantic interoperability is defined as the ability to process data based on the shared meanings of terms, as similarly stated in [9, 10, 11, 12, 13]. The meanings of terms can be shared by annotating data items with extra labels, which are not exactly structure. This type of interoperability is referred to as 'semantic' because we can distinguish two data by looking at their annotated labels, even if the data are structurally identical.

Comparing these two aspects of interoperability, structural interoperability is more fundamental than semantic interoperability for EHR data integration, because semantic annotations are useful only if the annotated attribute represents a valid value. In investigating the archetype semantic, we concentrate on the concept of structural interoperability.

2.1.2. EHR schema should be extensible

When we build every EHR system based on a unified and fixed EHR schema, we can easily achieve structural interoperability. In real settings, however, building a single giant schema is difficult especially in the medical domain, where concepts are not only large but also open-ended with breadth, depth, and complexity, because new information, finer-grained detail, and new relationships are always being discovered in medicine [14, 15].

The following example, which is a longitudinal cancer cohort study, illustrates the difficulties involved (Fig. 1). At the starting 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. 1). Or, if an unexpected exposure, such as a nuclear plant accident, is observed, the extra exposure (in this case, a dose of irradiation) should be added.

As this example indicates, when data are stored and traced longitudinally, the EHR schema must be extended as medicine progresses. 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 for modeling architecture. In particular, we focus on dual-model architecture and the specialization and composition of archetypes. At the end of this section, we point out the problem associate with their combination (variances).

2.2.1. Dual-model approach

ISO 13606, like openEHR, adopts a dual-model architecture (Fig. 2) to guarantee interoperability and support flexibility of the EHR schema. The architecture has a Reference Model (RM) and an Archetype Model (AM) in

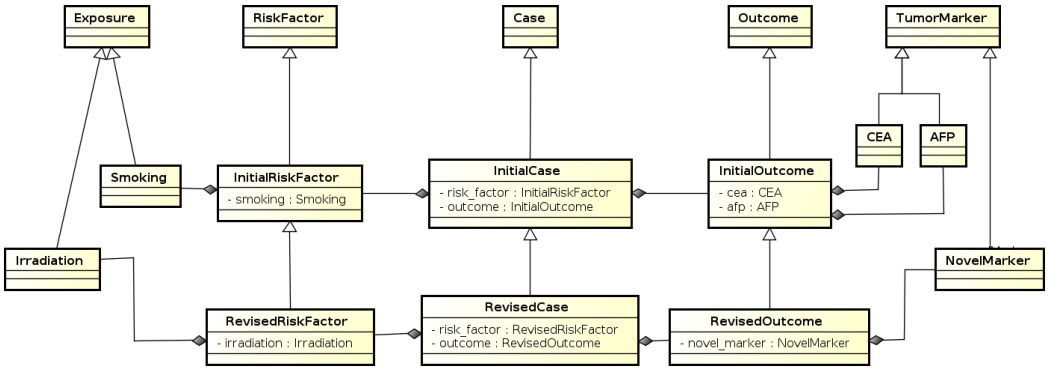

Figure 1: Unified modeling language diagram of an imaginary cohort study

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

the lower layer and archetypes in the upper layer. The primary feature of the modeling architecture is to construct archetypes from the components of the RM and the AM.

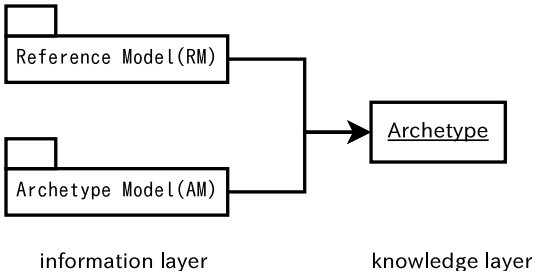

Figure 2: Dual-model approach

The diagram describes the dual-model architecture of ISO 13606, which consists of two layers, an information layer and a knowledge layer. The left-hand side of the figure indicates the information layer, in which the Reference Model and the Archetype Model are located, and the right-hand side indicates the knowledge layer, in which the Archetype resides. The arrows between these two layers indicate the dependency between the two layers, i.e., the combination of the RM and the AM yields an Archetype.

The lower layer of the dual-model architecture contains a Reference Model and an Archetype Model. The **Reference Model** defines a set of data types and data structure from which any health record information is composed [1, p.vi]. The model sits in the lower layer of the dual-model architecture, and represents the generic properties of health record information. [1, p.vi]. In short, the model forms the generic building blocks of the EHR and contributes to interoperability, data consistency, and data quality.

The **Archetype Model** is another component in the lower layer, and the model acts as a skeleton or template for

archetypes. The AM is an ”information model of the metadata to represent the domain-specific characteristics of electronic health record entries” [1, p.2]. 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 includes a specification of the hierarchical schema to which instances of data (i.e., EHR data) shall conform. The 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 the combinations of the RM entities in their definition section. On the other hand, the ontology section defines all linguistic entities in the archetype. The section provides ”the textual description for each concept from the definition section and bindings to other terminologies” [16, p.871], so that ”the archetypes can be natural-language- and terminology-neutral” [6, vii].

Looking back at the definitions of interoperability, we found that the definition and ontology sections in the AM correspond to each level of interoperability. The definition section defines the structure and constraints of the EHR data, and so is a means of assuring structural interoperability. In contrast, the ontology section binds an attribute with meaningful terminology, and so is a means of assuring semantic interoperability [17, p.586].

The **Archetype** resides in the upper layer of the dual-model architecture. Archetypes define clinical concepts in the form of structured and constrained combinations of the entities of an RM. While the difference between an Archetype and the AM is confusing, they reside on different layers, as shown in Fig. 2. The AM is static because it is defined by the specification in advance, whereas Archetypes are dynamic because they can be constructed at any time. 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 contrast, dual-model architectures statically include the basic components of schemas and the general mechanism for their construction, and the actual schemas are constructed dynamically from these components.

2.2.2. Specialization and composition of archetypes

The dual-model architecture enables the EHR schema to be defined dynamically and thus realizes Archetype flexibility. Moreover, ISO 13606 provides a method by which to further enhance flexibility, namely, specialization and composition of archetypes. With this mechanism, we can gradually extend the EHR schema in accordance with the progress of medicine or research.

First, archetypes may be specialized. 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 may be composed [6, ix]. New archetype can be defined by containing other archetypes when they are specified in ’slots’ of the new archetype (Fig. 3). Similar to class composition in object-oriented languages, this ’composition’ mechanism allows large data structures to be flexibly constrained via the hierarchical re-use of

smaller archetypes. [3, 18] Note that the 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) [19, p.61]. 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 3: Excerpt of architecture description language with archetype _slot

This fragment of architecture description language defines an archetype slot, indicating which SECTION archetypes are allowed and excluded under the items property. Note that regular expressions are used to designate the allowed and excluded archetypes.

2.2.3. Variances of archetype

When we define an archetype by combining specialization and composition, we encounter several possibilities, referred to as variances. 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. 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. Finally, the variance of a type is said to be invariant if the type is neither covariant nor contravariant (Fig. 4).

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 our tool 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.

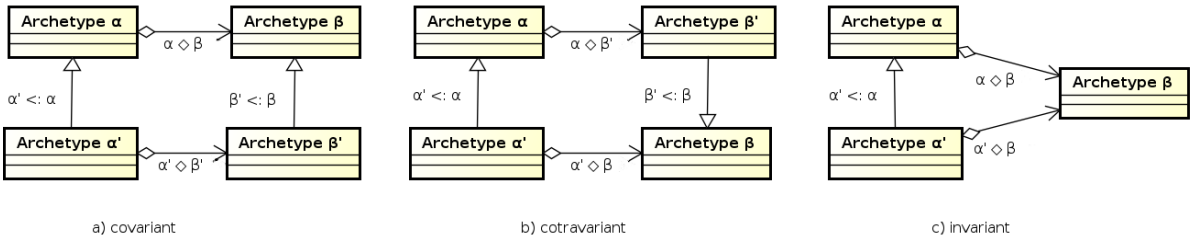

Figure 4: Three alternatives of archetype variances

The figure shows three alternative archetype variances. 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 'subtype of', and $\diamond$ means 'composed of'.

2.3.1. How type theory is applied to ISO 13606

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

We encoded the RM components without any internal structure as primitive types in type theory. Not surprisingly, a string in the RM is mapped to the string type, and structured 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 [1, 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.

$$\mathsf{TEXT} \equiv \mathsf{Record}\{\mathit{charset} : \mathsf{CS}, \mathit{languages} : \mathsf{CS}, \mathit{originalText} : \mathsf{String}\} \tag{1}$$

Archetypes, which also have some internal structures, are encoded in record types. For example, the vital signs archetype (Fig. 5) can be encoded as shown in Eq. 2. The specialization of archetypes corresponds to Record subtyping, and the composition of archetypes corresponds to Record nesting.

$$\begin{aligned}\mathsf{VitalSigns} \equiv \mathsf{Record}\{\mathit{definition}[\mathit{at}0000] : \mathsf{Record}\{\mathit{items}[\mathit{at}0001] : \mathsf{BloodPressure},\\ \mathit{items}[\mathit{at}0002] : \mathsf{BloodTemperature};\}\}\end{aligned} \tag{2}$$

EHR data are instances of Archetypes, [1] i.e., instances of the corresponding Record type, because each EHR

[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

```
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 5: Excerpt of the vital signs archetype

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

datum is created according to the corresponding archetype. For example, Eq. 3 yields 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.

$$vital\_signs \equiv record\{definition[at0000] : record\{items[at0001] : record\{systolic : 120, diastolic : 80\}, items[at0002] : record\{temperature : 37.5, unit : C\}\}\} \quad (3)$$

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

2.3.2. Introducing types

We presented the minimum set of typed lambda calculus as long as it was sufficient to provide the basis for our purpose, the purpose to clarify archetype semantics by means of formalization in support of the aforementioned two assertions. We start from fundamental types, such as the function type or basic types, and progressively extend the

[1, p.2].

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

system with more complex types, such as record or variant type (Table 2). For a more comprehensive explanation of type theory, see for instance,[20, 21, 22].

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

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

First, a formal assertion relating entities such as terms or types under a certain environment, is referred to as a **judgment** (Eq. 4). The expression $\Gamma \vdash M : \tau$ should be read as "a term M has a type of τ under an environment Γ". We write $\Gamma \vdash \tau$ to express 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**.

$$\Gamma \vdash \Omega \qquad \text{(judgment)} \tag{4}$$

In typed lambda calculus, every term has a type that is a collection of values with shared characteristics. The types are introduced into the system by adding **typing rules**, such as those given by Eq. 5. These typing rules assert the validity of certain judgments based on other judgments that are known to be valid. 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. 5. The notation ($M\ N$) denotes a function application in which a function M is applied to the argument N. 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. 6). This function type appears in database queries (such as Eq. 24), because a query is a function that accepts a database and returns a result

set of the database.

Typing rules:

$$\frac{\Gamma \vdash \tau \qquad \Gamma \vdash \sigma}{\Gamma \vdash \tau \to \sigma} \qquad \text{(function type)} \tag{5}$$

$$\frac{\Gamma \vdash M : \tau \to \sigma \qquad \Gamma \vdash N : \tau}{\Gamma \vdash (M\ N) : \sigma} \qquad \text{(application)} \tag{6}$$

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. 7. $\Gamma \vdash \diamond$ means that Γ is well-typed, which means that the type environment Γ has nothing but valid types.

$$\frac{\Gamma \vdash \diamond \qquad \tau \in \mathsf{BasicType}}{\Gamma \vdash \tau} \tag{7}$$

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. Note that, in the present study, we use M and N as terms τ and σ for meta-variables over type.

- Record Types

  A record type is a cross-product of the values with a projection operation for extracting components by name [23, 20, p.18].

  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. 8 and 9). The types section defines the Record type (Eq. 10). The typing rules section defines rules for the type of record construction and its projection (Eqs. 11 and 12). 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 from 1 to n.

Terms:

$$M ::= \ldots$$

$$\mid \mathit{record}\{l_1 = M, \ldots, l_n = M\} \qquad \text{(record construction)} \qquad (8)$$

$$\mid M.l_i \qquad \text{(record projection)} \qquad (9)$$

Types:

$$\tau ::= \ldots$$

$$\mid \mathsf{Record}\{l_1 : \tau, \ldots l_n : \tau\} \qquad (10)$$

Typing rules:

$$\frac{\Gamma \vdash M_i : \tau_i}{\Gamma \vdash \mathit{record}\{l_i : M_i\} : \mathsf{Record}\{l_i : \tau_i\}} \quad \mathit{where}\ i \in 1..n \qquad (11)$$

$$\frac{\Gamma \vdash M : \mathsf{Record}\{l_i : \tau_i\}}{\Gamma \vdash M.l_i : \tau_i} \quad \mathit{where}\ i \in 1..n \qquad (12)$$

- Variant Type

  While the 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. Figure 6 is an example of such a case, and this fragment of the archetype should be encoded as a variant type. The variant type, which is a named disjoint union of types, is a type into which values fall into a limited number of categories [20, 23].

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

Figure 6: Example of a Variant Type in DV_CODED_TEXT

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

- List Types

EHR data often contain a series of events, such as a Holter electrocardiogram. The list type has finite-length sequences of the same type that describe a series of events and is widely used in numerous programming languages, as well as in ISO 13606 [23, 20, 24]. In the present paper, $[\tau]$ describes finite-length sequences in which the element are taken from τ.

2.3.3. Subtyping

EHR data may be organized hierarchically. In Fig. 7, 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 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.

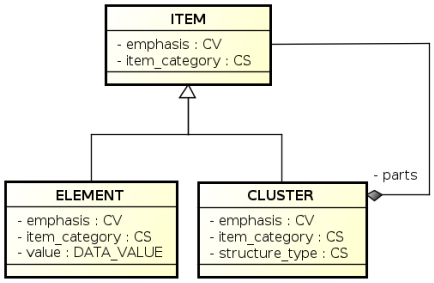

Figure 7: Excerpt of the ISO 13606 Reference Model

This unified modeling language diagram is an excerpt from the ISO 13606 Reference Model[1, p.10] and defines the hierarchy of ITEM and its subordinate ELEMENT and CLUSTER.

The central theorem of subtyping is the **subsumption principle** (Eq. 13), which says that if type τ' is a subtype of τ, then an instance of type τ' can be used in any context in which an instance of type τ can appear. In subtyping, the operator '<:' is a reflexive transitive closure (Eqs. 14 and 15) 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 $a = b$).

Typing rules:

$$\frac{\Gamma \vdash M : \tau' \qquad \Gamma \vdash \tau' <: \tau}{\Gamma \vdash M : \tau} \quad \text{(subsumption principle)} \tag{13}$$

$$\frac{\Gamma \vdash M : \tau}{\Gamma \vdash \tau <: \tau} \quad \text{(reflexive)} \tag{14}$$

$$\frac{\Gamma \vdash \tau'' <: \tau' \qquad \Gamma \vdash \tau' <: \tau}{\Gamma \vdash \tau'' <: \tau} \quad \text{(transitive)} \tag{15}$$

Subtyping rules should be defined for each type, and every subtyping rule should be compatible with the subsumption principle. We presented the 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. 16), 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. 17). The depth subtyping rule for the variant type is identical to that of the record type (cf., Eqs. 19 and 17), 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. 18).

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. 16 and 17.

Record subtyping rules:

$$\frac{\Gamma \vdash \tau_i \qquad where\ i \in 1..n+m}{\Gamma\ \vdash \mathsf{Record}\{l_i : \tau_i\}_{i\in 1..n+m} <: \mathsf{Record}\{l_i : \tau_i\}_{i\in 1..n}} \tag{16}$$

$$\frac{\Gamma \vdash \tau_i' <: \tau_i}{\Gamma\ \vdash \mathsf{Record}\{l_i : \tau_i'\} <: \mathsf{Record}\{l_i : \tau_i\}} \qquad where\ i \in 1..n \tag{17}$$

Variant subtyping rules:

$$\frac{\Gamma \vdash \tau_i \qquad where\ i \in 1..n+m}{\Gamma\ \vdash \mathsf{Variant}\{l_i : \tau_i\}_{i\in 1..n} <: \mathsf{Variant}\{l_i : \tau_i\}_{i\in 1..n+m}} \tag{18}$$

$$\frac{\Gamma \vdash \tau_i' <: \tau_i}{\Gamma\ \vdash \mathsf{Variant}\{l_i : \tau_i'\} <: \mathsf{Variant}\{l_i : \tau_i\}} \qquad where\ i \in 1..n \tag{19}$$

List subtyping rule:

$$\frac{\Gamma \vdash \tau' <: \tau}{\Gamma\ \vdash [\tau'] <: [\tau]} \tag{20}$$

In addition, structured types, even basic types, can be subtyped by introducing an ad hoc subtyping rule [23, p.28]. When we define the ad hoc rules, their actual semantics should be compatible with the subtyping. Otherwise, the system is consistent in theory, but useless in reality. For example, since we can regard an integer value as a real number (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. 7, for example, we can define subtyping between ELEMENT and ITEM as shown in Eq. 21 and CLUSTER and ITEM as shown in Eq. 22.

$$\frac{\Gamma \vdash \diamond}{\Gamma \vdash \mathsf{ELEMENT} <: \mathsf{ITEM}} \quad \text{(ad-hoc)} \tag{21}$$

$$\frac{\Gamma \vdash \diamond}{\Gamma \vdash \mathsf{CLUSTER} <: \mathsf{ITEM}} \quad \text{(ad-hoc)} \tag{22}$$

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**. In the given type system, we can construct a 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 8 shows a simple example of a derivation tree. This deductive system is the most powerful feature of typed lambda calculus, because we can investigate the semantics of ISO 13606 with formal deductions.

$$\frac{\Gamma \vdash \lambda x : \mathsf{Bool}.x : \mathsf{Bool} \to \mathsf{String} \qquad \Gamma \vdash \mathit{true} : \mathsf{Bool}}{\Gamma \vdash ((\lambda x : \mathsf{Bool}.x)\ \mathit{true}) : \mathsf{String}}\ [6]$$

Figure 8: Example of a derivation tree

This derivation tree indicates that the function of type $\mathsf{Bool} \to \mathsf{String}$ yields string type when applied to a boolean 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. 6).

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) and investigate the semantics of ISO 13606.

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. 23 and 24). 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. 24 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. 25 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".

$$database \equiv N : [\alpha] \tag{23}$$

$$query\ p \equiv foldr\ p\ [\,] : [\alpha] \rightarrow [\alpha] \tag{24}$$

$$\text{where} \quad p \equiv \lambda\, x : \alpha.(\lambda\, xs : [\alpha].M) : \alpha \rightarrow [\alpha] \rightarrow [\alpha] \quad \text{and,} \tag{25}$$

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

Based on these preconditions, **Assertion 2** is formally expressed in Fig. 9. The next task is to derive the judgment $(query\ p)\ N' : [\alpha]$ from these premises to support **Assertion 2**.

$$\text{provided that} \quad \Gamma \vdash \alpha' <: \alpha, \text{ and} \tag{26}$$

$$\Gamma \vdash query\ p : [\alpha] \rightarrow [\alpha], \text{ and} \tag{27}$$

$$\Gamma \vdash N' : [\alpha'] \tag{28}$$

$$\text{then, derive} \quad \Gamma \vdash (query\ p)\ N' : [\alpha] \tag{29}$$

Figure 9: Assertion 2 formalized

Note that α is a parent archetype, α' is the specialized archetype of α, and $N' : [\alpha']$ indicates "a list of data created by the specialized archetype α'"

2.4.2. How to introduce assignments for EHR data

Thus far, it has been impossible to update EHR data, because the record type that was introduced has only the projection operation and not the assignment operation for updating (Eq. 9). Most of the programming languages and database systems, however, use destructive updating methods. In other words, old values are replaced by new values. Therefore, we attempted to introduce update functionality to our system in order to determine what will happen to the archetype semantics.

In order to represent updatable EHR data, we introduced the additional types **Reference type** and **Unit type**. The Reference type is the mutable locations in memory [2], described as $\mathsf{Ref}(\tau)$ (Eq. 31). 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. 32 and 33).

[2]Interested readers can refer to Pierce [20, p.159] or Bruce [32, p.77] for more concrete explanations.

Types:

$$\tau ::= \ldots \tag{30}$$

$$\mid \mathsf{Ref}(\tau) \qquad \text{(type reference)} \tag{31}$$

$$\mid \mathsf{Unit} \qquad \text{(type unit)} \tag{32}$$

Typing rules:

$$\frac{\Gamma \vdash M : \mathsf{Ref}(\tau) \qquad \Gamma \vdash N : \tau}{\Gamma \vdash M := N : \mathsf{Unit}} \qquad \text{(assignment)} \tag{33}$$

Thus, mutable EHR data were encoded as Archetype, the fields of which are Ref Type, i.e., $\mathsf{Archetype}\{l_i : \mathsf{Ref}(\tau_i)\}$. The typing rule of Archetype is given by Eq. 34, which indicates that the subtyping relation of mutable Archetype depends on the subtyping of Reference type. In the Results section of the present paper, we describe a critical change in the archetype semantics that is brought by this new encoding of the archetype.

$$\frac{\Gamma \vdash M_i : \mathsf{Ref}(\tau_i)}{\Gamma \vdash \mathit{archetype}\{l_i : M_i\} : \mathsf{Archetype}\{l_i : \mathsf{Ref}(\tau_i)\}} \qquad (i \in 1..n) \tag{34}$$

$$\frac{\Gamma \vdash \tau' <: \tau \qquad \Gamma \vdash \tau <: \tau'}{\Gamma \vdash \mathsf{Ref}(\tau') <: \mathsf{Ref}(\tau)} \tag{35}$$

3. Results

We defined the basic semantics of ISO 13606 archetypes based on typed lambda calculus in the previous section. We derived two assertions (Assertions 1 and 2) based on the defined type system. Moreover, we investigated the consequences of introducing mutable EHR data 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, from our type system as follows. We divided this assertion into two cases: 1) any data created 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, $\alpha' <: \alpha$. 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. 13).

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. 16 and 17), we obtain the subtyping rule for archetypes as Eq. 36, which reveals that archetypes are covariant in their components.

$$\frac{\Gamma \vdash \tau'_i <: \tau_i \qquad where\ i \in 1..n+m}{\Gamma\ \vdash \mathsf{Archetype}\{l_i : \tau'_i\}_{i\in 1..n+m} <: \mathsf{Archetype}\{l_i : \tau_i\}_{i\in 1..n}} \tag{36}$$

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. 10. 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 10: Uniquely identifying archetype slot

In this fragment of ADL, the archetype 'openEHR-EHR-CLUSTER.device.v1' is identified as a string in the archetype slot, thus it is uniquely specified.

3.3. Queries should be reusable

In Section 2.4.1, we formalized **Assertion 2** in terms of typed lambda calculus as shown in Fig. 9. 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. 11. Given these conditions, we constructed the following derivation tree to support **Assertion 2** (Fig. 12).

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

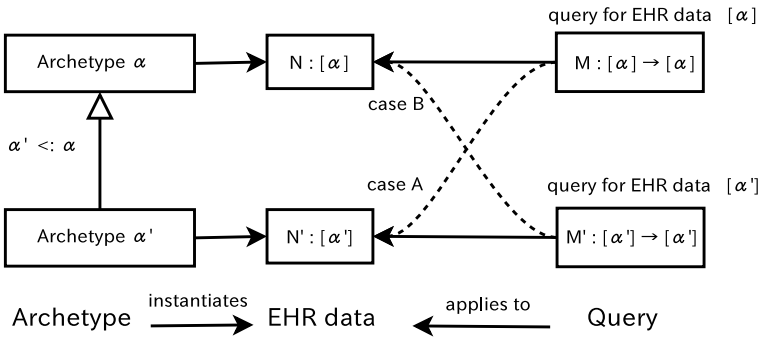

Figure 11: Two cases of query applications

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. In case A, we are applying a query for a parent archetype α to the data created from its specialized archetype α'. This is the case for **Assertion 2**. In case B, in contrast, we are applying a query for a specialized archetype α' to the data created from its parent archetype α.

$$\textsc{Derivation.}\quad \dfrac{\dfrac{\dfrac{\Gamma \vdash \alpha' <: \alpha}{\Gamma \vdash [\alpha'] <: [\alpha]}\ [20] \qquad \Gamma \vdash N' : [\alpha']}{\Gamma \vdash N' : [\alpha]}\ [13] \qquad \Gamma \vdash \mathit{query}\ p : [\alpha] \to [\alpha]}{\Gamma \vdash (\mathit{query}\ p)\ N' : [\alpha]}\ [6]$$

Figure 12: Derivation of Assertion 2

Note that α is a parent archetype, α' is its specialized archetype, and $N' : [\alpha']$ is a list of EHR data created by the specialized archetype α'.

Fig. 11. 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. 6). This indicates that queries based on a specialized archetype are not applicable to data created with the parent archetypes, whereas the queries based on the parent archetype can be reapplied to the data created by its specialized archetype.

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. 34. Therefore, we need to determine how to derive the judgment $\mathsf{Archetype}\{l_1 : \mathsf{Ref}(\tau')\} <: \mathsf{Archetype}\{l_1 : \mathsf{Ref}(\tau)\}$ based on the modified type system.

Subtyping of Reference types was defined in Eq. 35. We derived the tree shown in Fig. 13 by combining the mutable Archetype judgment and the Reference subtyping rules.

$$\textsc{Derivation.}\quad \frac{\dfrac{\tau' <: \tau \qquad \tau <: \tau'}{\mathsf{Ref}(\tau') <: \mathsf{Ref}(\tau)}\ [35]}{\mathsf{Archetype}\{l : \mathsf{Ref}(\tau')\} <: \mathsf{Archetype}\{l : \mathsf{Ref}(\tau)\}}\ [36]$$

Figure 13: Derivation of the mutable archetype

Figure 13 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. 14 and 15). Therefore, the derivation of Fig. 13 reveals that archetypes have to be invariant if they are mutable schemas.

Since this consequence is less intuitive, we demonstrate it through a counterexample. Based on Eq. 37, let us assume that mutable archetypes are covariant. Based on this false hypothesis, we identify an inconsistency.

Typing rules:

$$\frac{\Gamma \vdash \tau'_i <: \tau_i \qquad \textit{where } i \in 1..n+m}{\Gamma\ \vdash \mathsf{Archetype}\{l_i : \mathsf{Ref}(\tau'_{\mathrm{i}})\}_{i\in 1..n+m} <: \mathsf{Archetype}\{l_i : \mathsf{Ref}(\tau_{\mathrm{i}})\}_{i\in 1..n}} \tag{37}$$

Provided that type Int is integer and type PosInt is a positive integer, we assert that $\mathsf{PosInt} <: \mathsf{Int}$ by ad hoc rule, provided 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, $\mathsf{Archetype}^{\mathsf{A}} <: \mathsf{Archetype}^{\mathsf{B}}$. In addition, let us define $a \equiv \mathit{archetype}\{l = 1\} : \mathsf{Archetype}^{\mathsf{A}}$ and $b \equiv \mathit{archetype}\{l = -1\} : \mathsf{Archetype}^{\mathsf{B}}$. Given these settings, we can derive $a.l := -1$ as in Fig. 14. This, however, contradicts $a.l : \mathsf{Ref}(\mathsf{PosInt})$, because -1 is *not* a positive integer.

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

$$\textsc{Derivation.}\quad \dfrac{\dfrac{\dfrac{\dfrac{\mathsf{PosInt} <: \mathsf{Int}}{\mathsf{Archetype}^{\mathsf{A}} <: \mathsf{Archetype}^{\mathsf{B}}}\ [37] \qquad a : \mathsf{Archetype}^{\mathsf{A}}}{a : \mathsf{Archetype}^{\mathsf{B}}}\ [13]}{a.l : \mathsf{Ref(Int)}}\ [9] \qquad -1 : \mathsf{Int}}{a.l := -1 : \mathit{Unit}}\ [33]$$

Figure 14: Derivation from the false hypothesis

Type Int is integer, and type PosInt is a positive integer.

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 are 'structural interoperability' and 'schema extensibility'.

For the case in which the archetype is covariant, we can construct a specialized archetype along with its composition. As shown in Fig. 1, 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 this still guarantees structural interoperability, schema extensibility is still 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 in ISO 13606. We defined the semantics in terms of the type system and 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. Therefore, the simple question as to whether there are any discrepancies between the results obtained from the deductive proofs and the values of the EHR data from the actual evaluation results remains. The answer lies in 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 is assured to be valid, and the system is free from type errors [21, 23]. This is the fundamental requirement of structural interoperability. The soundness is stated in a more formal manner [23, p.12] by Eq. 38, indicating 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.

$$\text{if } \Gamma \vdash M : \tau \text{ is valid, then } [\![M]\!] \in [\![\tau]\!] \text{ holds.} \tag{38}$$

Note that the aforementioned type system is sound [33, 34], i.e., Eq. 38 holds for every component of the proposed type system. We omit the soundness proof, because it is beyond the scope of this 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. 39) and N is an instance of that archetype (Eq. 40). When we evaluate the term $(M\ N)$, we obtain the value "text" from Eq. 41. On the other hand, when we infer the type of (M N), we obtain TEXT (Fig. 16). Thus, "$text$" $\in \mathsf{TEXT}$ holds.

$$\text{provided that,} \quad M \equiv \lambda x : \mathsf{Archetype}\{l : \mathsf{TEXT}\}.(x.l) : \mathsf{Archetype}\{l : \mathsf{TEXT}\} \to \mathsf{TEXT} \tag{39}$$

$$N \equiv archetype\{l = \text{"}text\text{"}\} : \mathsf{Archetype}\{l : \mathsf{TEXT}\} \tag{40}$$

$$\text{then,} \quad [\![(M\ N)]\!] \Rightarrow_\beta \text{"}text\text{"} \tag{41}$$

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

Figure 15: Evaluation of a lambda term $(M\ N)$

$$\textsc{Derivation.} \quad \frac{M : \mathsf{Archetype}\{l : \mathsf{TEXT}\} \to \mathsf{TEXT} \qquad N : \mathsf{Archetype}\{l : \mathsf{TEXT}\}}{(M\ N) : \mathsf{TEXT}} \quad [6]$$

Figure 16: Type inference of a lambda term $(M\ N)$

Although this is a small example, Eq. 38 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. The novelty of our conclusion becomes clear when we present related research.

[3] Interested readers can refer to Fisher [35] or Wright [34] for a complete reference on the proof.

4.1. Why formal semantics are needed

The primary reason we could reach the above conclusion is that we built the 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*.[1, v]

However, semantics is an indispensable part of the standard and should be thoroughly investigated, even though the primary goal of ISO 13606 is in the specification of messages for communication of EHR data.

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 on not only 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. The semantics 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 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 we could never foresee. It is like the laws of physics, in terms of the degree to which they are conceptualized, which capture the true nature of the world as mathematical symbols and greatly expand the boundaries of our knowledge.

4.2. 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, 36]. Second, type theory as a formalized deductive system is useful for proof-theoretic investigations [37, p.9]. We can construct proofs of already defined typed rules by properly combining the rules. Furthermore, the validity of the proofs depend 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 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.

Typed lambda calculus guarantees a certain type of interoperability, referred to herein as 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.

4.3. 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) [38, 39, 40, 12, 41].

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 [42, 43]. 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 n 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 these studies [38, 39, 40, 12, 41] simply translated archetypes to OWL without analyzing the semantics, Marcos et al. [44] attempted to analyze archetype semantics based on DL. They actually performed DL reasoning to check the consistency of archetypes and revealed 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. However, their approach towards archetype consistency only considered archetype specialization, and they overlooked the problems of archetype variance and immutability, as in other previous studies.

Although type theory and logic are deeply related, as indicated by the Curry–Howard isomorphism [45, 46], type theory has several practical advantages over DL. Since type theory is equipped with a deductive system, constructing proofs is easier than in DL, which has a set-theoretic background. Furthermore, in typed lambda calculus, being a theoretical basis for modern programming languages, results are is easier to confirm or even implement compared to a DL-based approach.

4.4. Limitations

Typed lambda calculus has several advantages, and we 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.18). Suppose that we wanted to store a history of hepatitis and 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, so we encoded the two alternatives as CODED_TEXT.defining_code (Fig. 6). Later, a problem was encountered when hepatitis C was discovered. Since the subtype of a variant type has fewer choices than the supertype (Eq. 18), 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 is not extending the schema but only creating an additional schema, and this new schema is not able to reuse existing queries, as we showed in Fig. 12.

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 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 try to introduce higher-order types [23, p.24], such attempts fail because the higher-order types parametrize only types and not the associated field names. We suspect that this type of parametrized archetype is very difficult to encode in any existing formalisms, including the regular grammar theory or DL.

4.5. Prospectives

As mentioned above, the proposed type theory did not cover the entire range of ISO 13606 archetype semantics. This is partly because the theory may have some limitations, partly because ISO 13606 may go so far as to depart from the range of any formalisms, and partly 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 [1, p.42]. These types can be correctly captured, if we introduce a recursive type to the proposed system [23, 24].

Second, we did not discuss constraints on container attributes, such as cardinality, existence, occurrences, order, and unique [6, p.83-84]. The cardinality and occurrences constraints are difficult to handle because they are expressed as intervals, such as $\{0..1\}$ or $\{10..*\}$. 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, xi], the proper collection type can be assigned according to the combination of order and unique constraints.

Third, ISO 13606 attempts to represent every conceivable type of health record data structure in a consistent manner [1, vi], but the unit of measurement is one of the major exceptions. 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 [47, 48]. Once we develop sound semantics, we can manipulate EHR data in a more consistent manner, such as 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 of 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:

1. Archetypes should be both covariant and immutable schema, so that the structural interoperability is guaranteed and EHR schema are extended over time.
2. An archetype should uniquely specify other archetypes in the archetype slot when they are composed.

We hope that the results of the present study would provide all who are involved in ISO 13606 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 notations 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] 13606-1 Health informatics Part 1: Reference model, http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=40784 [accessed 1.14], 2008.

[2] J. T. Fernández-Breis, M. M. Tortosa, D. Moner, R. Valencia-García, J. A. Maldonado, P. J. V. Vicente, T. G. Miranda-Mena, R. Martínez-Béjar, An Ontological Infrastructure for the Semantic Integration of Clinical Archetypes, in: PKAW, 156–167, doi:10.1007/11961239_14, 2006.

[3] openEHR Architecture : Architecture Overview, http://www.openehr.org/releases/1.0.2/architecture/overview.pdf [accessed 1.14], 2008.

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

[5] R. Chen, G. O. Klein, E. Sundvall, D. Karlsson, H. Ahlfeldt, Archetype-based conversion of EHR content models: pilot experience with a regional EHR system, BMC Med Inform Decis Mak. 9 (1) (2009) 33, doi:10.1186/1472-6947-9-33.

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

[7] A. Ohori, A Simple Semantics for ML Polymorphism, Tech. Rep., University of Pennsylvania, doi:10.1145/99370.99393, 1989.

[8] A. Igarashi, B. C. Pierce, P. Wadler, Featherweight Java: a minimal core calculus for Java and GJ, ACM Trans Program Lang Syst. (2001) 396–450doi:10.1145/503502.503505.

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

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

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

[12] C. Martínez-Costa, M. Menárguez-Tortosa, J. T. Fernández-Breis, An approach for the semantic interoperability of ISO EN 13606 and OpenEHR archetypes, J Biomed Inform. 43 (5) (2010) 736–746, doi:10.1016/j.jbi.2010.05.013.

[13] J. A. Maldonado, D. Moner, D. Boscá, J. T. Fernández-Breis, C. Angulo, M. Robles, LinkEHR-Ed: a multi-reference model archetype editor based on formal semantics, Int J Med Inform. 78 (8) (2009) 559–570, doi:10.1016/j.ijmedinf.2009.03.006.

[14] A. L. Rector, Clinical Terminology: Why is it so hard?, Methods of Inf Med. 38 (4) (1999) 239–252.

[15] S. A. Renner, A. S. Rosenthal, J. G. Scarano, Data Interoperability: Standardization or Mediation, in: IEEE Metadata Workshop, Silver Spring MD, 1996.

[16] C. M. Costa, M. Menarguez-Tortosa, J. T. Fernandez-Breis, Clinical data interoperability based on archetype transformation, J Biomed Inform. 44 (5) (2011) 869–880, doi:10.1016/j.jbi.2011.05.006.

[17] G. Duftschmid, T. Wrba, C. Rinner, Extraction of standardized archetyped data from Electronic Health Record systems based on the Entity-Attribute-Value Model, Int J Med Inform. 79 (8) (2010) 585–597, doi:10.1016/j.ijmedinf.2010.04.007.

[18] The openEHR Archetype Model : Archetype Object Model, www.openehr.org/releases/1.0.1/architecture/am/aom.pdf [accessed 1.14], 2007.

[19] The openEHR Archetype Model : Archetype Definition Language, ADL 1.4, http://www.openehr.org/releases/1.0.1/architecture/am/adl.pdf [accessed 1.14], 2007.

[20] B. C. Pierce, Types and Programming Languages, MIT press, 1 edn., 2002.

[21] J. C. Mitchell, Foundations for Programming Languages, MIT Press, 1 edn., 2000.

[22] R. Harper, Practical Foundations for Programming Languages, Cambridge, 1 edn., 2013.

[23] L. Cardelli, Type Systems, in: Handbook of Computer Science and Engineering, chap. 97, CRC Press, 2004.

[24] H. Geuvers, Introduction to Type Theory, in: Language Engineering and Rigorous Software Development, vol. 5520 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 1–56, doi:10.1007/978-3-642-03153-3_1, 2009.

[25] L. D. Craig, Object-Oriented Programming Languages, Springer-Verlag London, 2007.

[26] G. G. Hillebrand, P. C. Kanellakis, H. G. Mairson, Database Query Languages Embedded in the Typed Lambda Calculus, Inform Comput. 127 (2) (1996) 117–144, doi:10.1006/inco.1996.0055.

[27] T. Grust, Comprehending Queries, Tech. Rep., University of Konstanz, 1999.

[28] T. Grust, M. H. Scholl, How to Comprehend Queries Functionally, J Intell Inf Syst. 12 (2-3) (1999) 191–218, doi:10.1023/A:1008705026446.

[29] A. Poulovassilis, C. Small, Algebraic query optimisation for database programming languages, The VLDB Journal 5 (2) (1996) 119–132, doi:10.1007/s007780050019.

[30] G. Hutton, A tutorial on the universality and expressiveness of fold, J Funct Programming. 9 (4) (1999) 355–372, doi:10.1017/S0956796899003500.

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

[32] K. B. Bruce, Foundations of Object-Oriented Languages, MIT press, 2002.

[33] C. A. Gunter, Semantics of Programming Languages, Foundations of Computer Series, MIT Press, 1 edn., 1992.

[34] A. Wright, M. Felleisen, A Syntactic Approach to Type Soundness, Inform Comput. 115 (1) (1994) 38–94, doi:10.1006/inco.1994.1093.

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

[36] H. Barendregt, W. Dekkers, R. Statman (Eds.), Lambda Calculus with Types, Perspectives in Logic, Cambridge University Press, 1 edn., 2013.

[37] T. U. F. Program (Ed.), Homotopy Type Theory: Univalent Foundations of Mathematics, The Univalent Foundations Program Institute for Advanced Study, first-edition-323-g28e4374 edn., 2013.

[38] C. Martínez-Costa, M. Menárguez-Tortosa, J. T. Fernández-Breis, J. A. Maldonado, A model-driven approach for representing clinical archetypes for Semantic Web environments, J Biomed Inform. 42 (1) (2009) 150–164, doi:10.1016/j.jbi.2008.05.005.

[39] J. A. Maldonado, C. M. Costa, D. Moner, M. Menárguez-Tortosa, D. Boscá, J. A. M. Giménez, J. T. Fernández-Breis, M. Robles, Using the ResearchEHR platform to facilitate the practical application of the EHR standards, J Biomed Inform. 45 (4) (2012) 746 – 762, doi:10.1016/j.jbi.2011.11.004.

[40] L. Lezcano, M. A. Sicilia, C. R. Solano, Integrating reasoning and clinical archetypes using OWL ontologies and SWRL rules, J Biomed Inform. 44 (2) (2011) 343 – 353, doi:10.1016/j.jbi.2010.11.005.

[41] C. M. Costa, A. Q. de Andrade, D. Karlsson, D. Kalra, S. Schulz, Towards the harmonization of clinical information and terminologies by formal representations, European Journal for Biomedical Informatics. 8 (3) (2012) 3–10.

[42] B. Chidlovskii, Using Regular Tree Automata as XML schemas, in: Proc. IEEE Advances on Digital Libraries Conference, 89–98, doi:10.1109/ADL.2000.848373, 2000.

[43] M. Murata, D. Lee, M. Mani, K. Kawaguchi, Taxonomy of XML Schema languages using formal language theory, ACM Trans Internet Technol. 5 (4) (2005) 660–704, doi:10.1145/1111627.1111631.

[44] M. Menárguez-Tortosa, J. T. Fernández-Breis", OWL-based reasoning methods for validating archetypes, J Biomed Inform. 46 (2) (2013) 304–317, doi:10.1016/j.jbi.2012.11.009.

[45] F. Lawler, Classical Logic and the Curry-Howard Correspondence, Tech. Rep., Trinity College Dublin, 2008.

[46] M. H. B. Sørensen, P. Urzyczyn, Lectures on the Curry-Howard Isomorphism, 1998.

[47] A. Kennedy, Dimension Types, in: Programming Languages and Systems — ESOP '94, vol. 788, Springer Berlin Heidelberg, 348–362, doi:10.1007/3-540-57880-3_23, 1994.

[48] A. Kennedy, Types for Units-of-Measure: Theory and Practice, in: Central European Functional Programming School, vol. 6299, Springer Berlin Heidelberg, 268–305, doi:10.1007/978-3-642-17685-2_8, 2010.