A Set-Oriented Formalism as a Foundation for the Modeling and Verification of Connected Data and Process Specifications

Julia Martini, Hannes Restel, Raik Kuhlisch, Jörg Caumanns
E-HEALTH // ESPRI
Fraunhofer-FOKUS
Berlin, Germany
{julia.magdalena.martini, hannes.restel, raik.kuhlisch, joerg.caumanns}@fokus.fraunhofer.de

Abstract—**In this work-in-progress report, a methodology based on a fully formalized and machine-readable formalism is introduced. The goal is to help modelers/developers to design and verify specifications, standards, and profiles in the field of information exchange. The formalism allows the specification and verification of process models as well as data/information models. These formalized specifications describe the structure (syntax) and meaning (semantics) of data models and process models as well as relations (requirements, dependencies, rules, constraints, pre-/post-conditions) between them. Unlike the traditional approach of defining a specification, which is to first write an unstructured specification document and then to derive a platform-specific binding from it (e.g., XML Schema), the specification itself is directly defined in a structured and machine-understandable formalism on a logical level. Fully formalized specifications allow for automatic validation and verification and, therefore, allow checking if the specification is complete and consistent so that dependencies between process steps can be verified. This work in progress lines out the very foundations of the described methodology by introducing a Set-Oriented formalism (SOF) that is used to formalize data models and dependencies.**

Keywords-specification; formalism; profiling; validation; information modeling.

I. INTRODUCTION

To simplify the development of applications and to achieve interoperability, acknowledged specifications (de jure standards, de facto standards) for document and message exchange are widely used in our current era of net-based information exchange. An information exchange specification may define data and information models as well as process/protocol models. To support a multitude of different use cases in a variety of domains, standards are usually defined in a rather generic way. This often results in an (intentionally) ambiguous specification that allows multiple interpretations by different parties and, therefore, limits interoperability. To counter this effect, domain specific profiles are derived to restrict the specification and to make it unambiguous. Even a set of specifications may be compiled into a single profile to specify a more complex process.

Often, the existing information exchange specifications are barely represented as fully formalized documents and, therefore, cannot be understood by machines. They need to be manually interpreted, transformed and bound into a serializable, machine-computable representation on the platform specific level [1] that is finally used to generate and exchange instances (messages, documents) of those models on runtime. Technologies/methodologies are widely used to support those steps (see Table I).

For example, the Unified Modeling Language (UML) [2] and the Object Constraint Language (OCL) [3] may be used to define the data models as well as constraints, and the Extensible Markup Language (XML) Schema [4] and Schematron [5] may be used for the binding. Still, large parts of the specifications located on the computational independent level (CIM, see [1]) and platform independent level (PIM, see [1]) are represented as unstructured (free text) documents describing the purpose, syntax and semantics of a specification. Thus, an automated transformation from one step of the specification development chain to the other is rarely possible.

If a set of specifications is compiled to define a profile, then the complexity increases because relations (requirements, dependencies, rules, constraints, pre-/ post-conditions) between the data and process models of each incorporated specification do exist. The more complex those relations are, the more difficult it is to define a valid, complete and unambiguous profile and to verify the correctness of the profile.

To counteract the above-mentioned problems, a methodology based on a mathematical, formalized and machine-readable formalism is introduced in this report, called Set-Oriented formalism (SOF).

This formalism allows the specification and verification, both of the defined process models as well as the data/information models of a specification including the relations between the models on a level prior to the platform specific level, i.e., on CIM and PIM.

 ISBN: 978-1-61208-367-4

TABLE I. DEVELOPMENT CHAIN FROM SPECIFICATION OVER BINDING TO INSTANCES

Step	Denoted in
Specification (Standard, Profile)	Unstructured document, UML/OCL, Business Process Modeling Language, Fundamental Modeling Concepts (FMC) etc.
Binding	XML Schema, Schematron, Resource Description Framework-Schema (RDFS), Structured Query Language (SQL), JSR-94, etc.
Instance	XML, JavaScript Object Notation (JSON), etc.

In Section II, existing formalisms are evaluated, especially UML and OCL. In Section III, the Set-Oriented Formalism itself is introduced by formally defining its structure component and rule component. Subsequently, the transformation from a sample model depicted in SOF into a platform specific perspective (i.e., XML/Schematron representation) is described in Section IV. To support comprehensibility, all Sections make use of a shared example. This report concludes with a short summary of the findings and an outlook (Section V).

II. RELATED WORK

The worldwide established modeling language UML supports the standardized specification, construction and documentation of a system. UML's boundaries are that element-spanning semantic constraints/dependencies are not representable. For defining these rules for model elements, UML was augmented with the OCL, which is a declarative language. With the OCL one can, for example, define invariants, pre-, and post-conditions.

In SOF, the elementary dependencies regarding the cardinalities and data types that the UML depicts graphically are expressed with the structure component. The rules for model elements (e.g., invariants, pre- and post-conditions) are defined in the rule component, thereby interpreting each rule as a set of sets, which constitutes a valid instance of a model regarding that rule. OCL's boundaries are that inconsistent specifications, that is the combination of constraints contradicting each other, cannot be recognized. In SOF, the recognition of inconsistent specifications is possible. Since each constraint represents a set of valid instances of the model, checking via the intersecting set can determine whether an instance exists at all that fulfills all constraints (see Section III.C, list item c). In particular, this can be done pairwise for the constraints. The criteria for such a consistent specification is, hence, that the intersecting set of constraint sets is not empty. The familiar "frame problem" that can arise with OCL can also be solved with SOF since the post-conditions can be represented in SOF as the final state of the whole system.

III. THE SET-ORIENTED FORMALISM (SOF)

The SOF is designed to specify fully formalized and serializable data/ information models. Any data model, which can be transformed into a tree, is defined as a serializable information model. The basic principle of SOF is to represent all elements and their properties and constraints of a serializable information model as a set. For example, dependencies regarding the cardinality, data type or value of each element are represented as a set. This enables the interpretation of element-spanning constraints as a set of sets that constitutes a valid instance of the model regarding the constraint. Through this set-oriented representation of each element and the constraints, each verification problem can be reduced to a subset or intersecting-set problem (see Section III.C). The SOF has two main components: the structure and the rule component. The structure component is a degenerated table. Each cell represents an element of the information model referenced by an identifier. Using these identifiers, rules defined within the rule component may be attached to each cell to express dependencies regarding the values and cardinalities of the elements among themselves.

A. The Structure Component

The structure component itself includes a constructively defined table T_S that represents a serializable information model S in a way that each cell unit (called T_S-cell) is a representative of a unique element of S. Each T_S-cell is fully formalized, providing information about the data types and cardinalities of its S-element representative. The construction is hierarchy-preserving, so that each T_S-cell is assigned to a unique identifier i by its location. The identifier i is assigned to the same element of the information model using an algorithm to navigate within the tree of the information model (as shown in Figure 5 below).

1) Construction of T_S

Assume a serializable information model S and an infinite table T so that T contains an infinite amount of rows and columns. Additionally, the tuple (i, j) with $i, j \in \mathbb{N}^+$ will be the representative of the T-cell that is located in the i-th row and j-th column. The algorithm for the construction of T_S is displayed in Figure 1, whereby the infinite table T evolves into T_S, as the representative of the information model S.

Each time *createStructuralComponent()* is executed, e provides the current element and its location *(i,j)* within the table (see Figure 1). At the first call (highest level of recursion) e is the root element of the serializable information model S and *(i,j)=(1,1)*. The cells of the sub-elements of the current element e are recursively evolved until there is only one element with no further sub-elements left, i.e., a leaf.

The following section describes how an element (i.e., T_S-cell) of the information model in SOF (as suggested by the function *fill_T-Cell()*) has to be coded.

```
1. createStructuralComponent(Element e, T-cell (i,j))
2.  fill_T-cell(e, (i,j))
3.  for each sub-element c
4.   k=i
5.   i= createStructuralComponent(c, (i,j+1))
6.   connect all T-cells between (k,j) and (i,j) to one Ts-cell
7.   if no sub-elements exists
8.    return i+1
9.   else
10.    return i
```

Figure 1. Algorithm for the construction of T_S.

2) Syntax and Markup for the T_S-cells

The name and specifications regarding the cardinality and data types written in the *T*-cell are predetermined by the information model S. The syntax markup depicted in Figure 2 has been developed in order to represent each element's features.

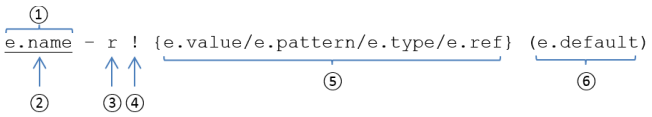

Figure 2. Syntax of the cell markup.

a) Name of element:

① The name of the element *e*.

b) Markup concerning the cardinalities:

② If *e* is a *required* element, ① will be underlined.

④ If *e* is an *at-most-once* element, it will be marked with "*!*".

→ If *e* is a *prohibited* element, see ⑤.

→ If *e* is an *exactly-once* element, ① will be underlined and marked with "*!*".

→ If neither the name of *e* is underlined nor the annotation "*!*" is used, *e* is an *optional-many* element.

③ Restrictions regarding the cardinality and data types of *e* that are determined by other elements are defined as rules in the rule component. Each rule of *r* is referenced using unique Roman numerals (e.g., "*III*") with an optional prefix or universal rules (see Section B.2).

c) Markup concerning data types:

⑤ The value range of *e* is illustrated using curly brackets. For instance, within an XML-based standard the value range is interpreted using the following XML Schematron definitions: *value, pattern, type, ref*. In the case of *ref* anon, the value range of the referencing XSD element is used.

→ If *e* is a *prohibited* element, "*Ø*" will be annotated instead of the value range.

⑥ If *e* holds a *default* value, it will be noted within round brackets.

Figure 3 shows an extract of the SAML specification [6], which will serve as a continuous example in this report.

Figure 4 shows how the structure component is applied to that SAML extract.

```
1. <?xml version="1.0" encoding="US-ASCII"?>
2. <schema targetNamespace="urn:oasis:names:tc:SAML:2.0:assertion"
   xmlns=http://www.w3.org/2001/XMLSchema
   xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion" version="2.0">
3.  <!–[...]-->
4.  <element name="Assertion" type="saml:AssertionType"/>
5.  <complexType name="AssertionType">
6.   <sequence>
7.   <element ref="saml:Subject" minOccurs="0"/>
8.   </sequence>
9.   <!–[...]-->
10.  </complexType>
11.  <element name="Subject" type="saml:SubjectType"/>
12.   <complexType name="SubjectType">
13.   <!--[...]-->
14.       <element ref="saml:SubjectConfirmation"
   maxOccurs="unbounded"/>
15.   </complexType>
16.  <element name="SubjectConfirmation"
   type="saml:SubjectConfirmationType"/>
17.   <complexType name="SubjectConfirmationType">
18.     <sequence>
19.       <!--[...]-->
20.       <element ref="saml:SubjectConfirmationData"
   minOccurs="0"/>
21.     </sequence>
22.     <attribute name="Method" type="anyURI" use="required"/>
23.   </complexType>
24.  <element name="SubjectConfirmationData"
   type="saml:SubjectConfirmationDataType"/>
25.   <complexType name="SubjectConfirmationDataType" mixed="true">
26.     <complexContent>
27.       <restriction base="anyType">
28.         <attribute name="NotBefore" type="dateTime"
   use="optional"/>
29.         <attribute name="NotOnOrAfter" type="dateTime"
   use="optional"/>
30.         <attribute name="Recipient" type="anyURI"
   use="optional"/>
31.         <!--[...]-->
32.       </restriction>
33.     </complexContent>
34.   </complexType>
35.  </schema>
```

Figure 3. Extract from the XML Schema Definition of a SAML assertion.

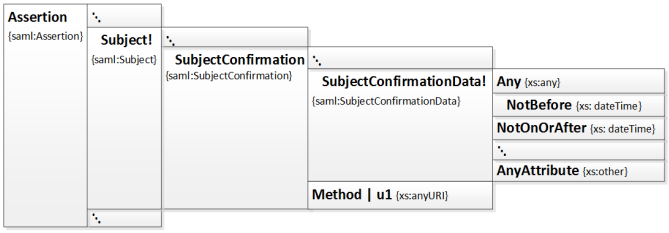

Figure 4. Extract of a SAML assertion as a structure component in SOF.

3) Referencing and Navigation using Identifiers

Each serializable information model *S* can be transformed into a tree. In order to refer to an element within such a tree, it is sufficient to provide, for instance, the path in a tuple form $(x_1, x_2, \dots, x_n)$. The path has to be followed from the root element in order to get to the last element. After the construction of T_S, a T_S-cell is accessible by a special navigation through a tuple referring to the same element (see algorithm in Figure 5). If $(x_1, x_2, \dots, x_n)$ is a tuple, describing an element *e* within the tree of the information model *S*, then the navigation in T_S will be, as shown in Table III.

Figure 6 applies the algorithm on the sample SAML extract (compare to listing in Figure 3).

```
1. while i < n
2.  goto x_i − 1 cell downward
3.  goto the top right-hand column relative to the current cell
4. goto x_i − 1 cell downward and print output
```

Figure 5. Algorithm to navigate within the tree of the information model.

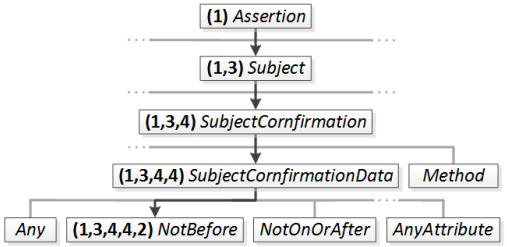

Figure 6. Navigation within the tree representation for an SAML assertion.

The table-oriented navigation within the structure component, according to the path in Figure 6, is further illustrated in Figure 7.

To keep the rules short, elements of *S* are referenced with tuples instead of their full names.

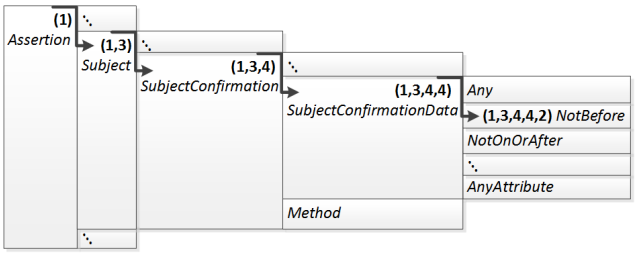

Figure 7. Navigation path within the structure component.

B. The Rule Component

The rule component for an information model *S* holds rules that *S* has to fulfill at all times, i.e., evaluates to *true*. Each rule is identified by unique Roman numerals to reference it within the T_S-cells of the structure component. Vice versa the T_S-cells addressed within a rule are identified by their tuple identifier.

1) Syntax of the Rule Component:

The syntax of the rule component is defined using a Domain-Specific Language (DSL) created in Xtext [7]. The syntax of the DSL will be explained in the subsequent paragraphs. The basic elements are sets that are categorized as follows:

a) Simple Sets

- *Enumerative sets* specify the elements contained (via their identifiers), separated by commas and encompassed by curly brackets.
- *Defined sets* are an accumulation of elements that fulfill specific characteristics regarding their cardinality. The four defined sets are A_S, R_S, P_S and I_S:
  - A_S contains all *at-most-once* elements,
  - R_S contains all *required* elements,
  - P_S contains all *prohibited* elements and
  - I_S instanced in the instantiation of *S*.

Therefore, the *defined sets* are S-specific and defined within the structure component's context since its elements are already underlined and marked with the respective symbols ("*!*" and "*Ø*").

b) Feature Sets

Feature sets are denoted as "*[A]*" and contain the elements that fulfill the requirements of statement *A*. For example, the set $[\{(1),(1,2)\} \subset \{(1),(1,3)\}]$ evaluates to the set $\{(1)\}$ since $[\{(1)\} \subset \{(1),(1,3)\}]$ is *true,* but $[\{(1,2)\} \subset \{(1),(1,3)\}]$ is *false.*

The sets C and D can be combined with the following operations and relations:

- $C - D$ forms the set of all elements of C except those contained in D.
- $C + D$ forms the set of all elements contained in C or D.
- $C \cap D$ forms the set of elements that are contained in C and D; machine-understandable: C *intersect* D.
- $< x >$ provides the value of the element with the identifier x.

Statements are created through the subsets' correlations or the set operators, respectively:

- $C \subset D$ is true if and only if all elements contained in C are also contained in D; machine-understandable: C *subsetOf* D.
- $C \not\subset D$ is true if and only if all elements in C are not contained in D; machine-understandable: C *notSubsetOf* D.
- $\#C$ provides the number of elements in C. A statement is formed with the #-operator, the relational operators >=, <=, =, >, < together with an accompanying integer.

The statements can be linked with the known sentential connectives AND, OR, XOR, and =>.

Operator and sentential connective ranking order: As there is no existing operator and sentential connective ranking order, the latter has to be defined using appropriate brackets so that the nesting represents the desired priority.

 ISBN: 978-1-61208-367-4

When using SOF, it became apparent that there often was a repetition of rules with the same content. To avoid the latter, so-called universal rules have been established.

2) Universal Rules

A universal rule *u* applies to each T_S-cell in which *u* has been referenced. Subsequently, all universal rules are semantically limited in their cardinality since the referenced elements of the serialized information model *S*, which are underlying the universal rules, have to be derived from the position of the annotated field. An example of such a universal rule is *uI*:

$$uI\colon \left(\left\{\left(x_1, \dots, x_{\#(x)-1}\right)\right\} \subset I_S \; AND \; \{x\} \subset R_S\right) XOR$$

$$\left(\left\{\left(x_1, \dots, x_{\#(x)-1}\right)\right\} \not\subset I_S \; AND \; \{x\} \not\subset I_S\right).$$

The rule states that the element *e* annotated with *uI* has to be instantiated if and only if its parent element has been instantiated.

C. Example of a Validation

This section briefly demonstrates how a validation is conducted by validating, if a profile conforms to a standard. In order to make a correct statement to this effect, the problem will be reduced to three subset problems. Assume a profile *P*, represented in SOF, which is derived from an existing standard *S* that itself is represented in SOF.

P is a valid profile of S if and only if the following subset relations are fulfilled, so that $P \subset S$ evaluates to *true*.

a) For the A, R, P sets of the respective rule component (of P and S) that represent the cardinalities of all elements: $A_P \subset A_S$, $R_P \subset R_S$, $P_P \subset P_S$.

b) Let D_{X_i} be the set-representation of the data type or the value of the model element i of the model X, as defined in Section A.2). The following must apply for each element (cell of the structure component) with the identifier i:

$$\forall i \in P\colon D_{P_i} \subset D_{S_i}.$$

c) Assume *S* is consistent, i.e., there are no rules that contradict each other:

$$\bigcap_{r=I,II,\dots} r \neq \{\}.$$

Then, each rule $r = I, II, \dots$ of the rule component needs to be checked whether *P* is included in the set interpretation of this rule:

$$\forall r = I, II, \dots : P \in r \Leftrightarrow P \in \bigcap_{r=I,II,\dots} r\,.$$

IV. TRANSFORMATION FROM SOF TO XML

To realize a platform-specific binding, a data model being developed using SOF can be transformed into an XML representation. It appears inconvenient to define structural properties within Schematron, so it will be assumed that the structure is defined using an XML Schema that is acting as the structure component. The semantic restrictions are defined by embedded Schematron rules that are acting as the rule component. The XPath expressions of the *rule* elements *context*, *assert* and *report* are evaluated to a Boolean expression, while its constructs can be translated, as shown in Table II.

To exemplify how the transformation works, a rule defined in SAML Profiles [8] is displayed both in Schematron and in SOF. The rule states that if an attribute *Method* is given the URI-value "*urn:oasis:names:tc:SAML:2.0:cm:holder-of-key*", then "One or more *<ds:KeyInfo>* elements MUST be present within the *<SubjectConfirmationData>* element. An *xsi:type* attribute MAY be present in the *<SubjectConfirmationData>* element and, if present, MUST be set to *saml:KeyInfoConfirmationDataType* (the namespace prefix is arbitrary but must reference the SAML assertion namespace)" [8].

It is impossible to express this rule exclusively using XML Schema. A schema validation check would be insufficient, so a Schematron rule (see Figure 8) is embedded within an XML Schema for the SAML example.

TABLE II. EQUIVALENTS OF XPATH AND SOF

Language construct	Language	
	XPath	***SOF***
Path	a/b/c	(1,2,3)
Structure	<Rule context = „A“> <assert a=“B“> </assert>	A => B
	[a::b = "predicate"]	[<(1,2,3)> = "predicate"]
Operators/ Relations		
Integer	+, -, *, /, <, <=, =, >=, >, !=	
Boolean	and, or	
String	= , !=	
	concat()	++
Node set	count()	#
Node	string()	<>

```
1. <pattern id="subject-confirmation">
2.  <title>Holder of Key</title>
3.  <rule context="saml:SubjectConfirmation[@Method =
    'urn:oasis:names:tc:SAML:2.0:cm:holder-of-key']">
4.   <assert test= "saml:SubjectConfirmationData/ds:KeyInfo">
5.     Message1
6.   </assert>
7.   <assert test= "not(saml:SubjectConfirmationData[@xsi:type]) or
     saml:SubjectConfirmationData[@xsi:type =
     'saml:KeyInfoConfirmationDataType']">
8.     Message2
9.   </assert>
10.  </rule>
11. </pattern>
```

Figure 8. Exemplary Schematron rule for SAML profiles.

Using SOF, the same rule within the rule component in combination with the respective structure component, as depicted in Figure 9, is defined.

```
   //Holder of key
1. <(1,3,4,5)> = "urn:oasis:names:tc:SAML:2.0:cm:holder-of-key" =>
2. ({(1,3,4,4,1)}subsetOf R AND <(1,3,4,4,1)> subsetOf "ds:KeyInfo"
3.  AND ({(1,3,4,4,01)} subsetOf I => <(1,3,4,4,01)> =
  "saml:KeyInfoConfirmationDataType"))
```

Figure 9. Rule for SAML profiles represented in SOF.

V. CONCLUSION AND FUTURE WORK

A Set-Oriented Formalism (SOF) consisting of a structure component and rule component has been introduced in this work-in-progress report. The SOF defines a machine-understandable formalism for the specification of data models on a logical level (as part of information exchange specifications). The defined data models are fully formalized and, therefore, machine-understandable, allowing them to be verified automatically.

The SOF acts as the foundation for an underlying methodology that aims to support modelers/developers in creating complex information exchange profiles based upon a set of data models and process models. The goal is to represent the models and relations between the models in a fully formalized notation so that integrity and verification checks can be automatically performed and platform-specific bindings can be generated automatically. No further usage of different notations/standards for the specification, profiling, and binding is needed, as all of those steps are covered by a single formalism.

The current development state of the SOF allows to define data models and rules and to verify a single data model. Modeling of process models is not yet available. Further research is needed to identify whether an algebraic calculus is suited to cover the needed requirements for defining and verifying process models and the relations between models. In addition, it is already possible to derive a profile from a data model. A graphical user interface is planned to make those steps easier to use. Implementing a verification component is planned to verify a given information model to the syntax compliance as well as the semantic correctness with respect to the underlying rule component. Furthermore, components for the automatic generation of platform-specific binding are intended (XML Schema and Schematron as well as HL7 FHIR [9]). Finally, the formalism needs to be extended to work with a set of models so that relations (requirements, dependencies, rules, constraints, pre-/post-conditions) between process steps can be defined and verified.

Subsequent papers and publications are planned that will describe further components around the SOF (such as process modeling, combination of formalized specifications, multi-model verification, etc.).

REFERENCES

[1] M. Belaunde et al., “MDA Guide Version 1.0.1,” June, 2003.

[2] “Unified Modeling Language (UML).” [Online]. Available: http://uml.org/. [retrieved: August, 2014].

[3] “OCL.” [Online]. Available: http://www.omg.org/spec/OCL/. [retrieved: August, 2014].

[4] W3C, “W3C XML Schema.” [Online]. Available: http://www.w3.org/XML/Schema#dev. [retrieved: August, 2014].

[5] ISO/IEC, “Schematron.” [Online]. Available: http://www.schematron.com/. [retrieved: August, 2014].

[6] S. Cantor, J. Kemp, R. Philpott, and E. Maler, “Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0, http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf.” März-2005.

[7] “Xtext.” [Online]. Available: http://www.eclipse.org/Xtext/. [retrieved: August, 2014].

[8] J. Hughes et al., “Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0, http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf.” March, 2005.

[9] HL7, “HL7 FHIR.” [Online]. Available: http://www.hl7.org/implement/standards/fhir/. [retrieved: August, 2014].

 ISBN: 978-1-61208-367-4