# Artemis Message Exchange Framework: Semantic Interoperability of Exchanged Messages in the Healthcare Domain *

Veli Bicer, Gokce B. Laleci, Asuman Dogac, Yildiray Kabak
Software Research and Development Center
Middle East Technical University (METU)
06531 Ankara Türkiye
email: asuman@srdc.metu.edu.tr

## ABSTRACT

One of the most challenging problems in the healthcare domain is providing interoperability among healthcare information systems. In order to address this problem, we propose the semantic mediation of exchanged messages. Given that most of the messages exchanged in the healthcare domain are in EDI (Electronic Data Interchange) or XML format, we describe how to transform these messages into OWL (Web Ontology Language) ontology instances. The OWL message instances are then mediated through an ontology mapping tool that we developed, namely, OWLmt. OWLmt uses OWL-QL engine which enables the mapping tool to reason over the source ontology instances while generating the target ontology instances according to the mapping patterns defined through a GUI.

Through a prototype implementation, we demonstrate how to mediate between HL7 Version 2 and HL7 Version 3 messages. However, the framework proposed is generic enough to mediate between any incompatible healthcare standards that are currently in use.

## 1. INTRODUCTION

Most of the health information systems today are proprietary and often only serve one specific department within a healthcare institute. A number of standardization efforts are progressing to address this interoperability problem such as EHRcom [3], openEHR [15] and HL7 Version 3 [6]. Yet, it is not realistic to expect all the healthcare institutes to conform to a single standard. Furthermore, different versions of the same standard (such as HL7 Version 2 and Version 3) and even the different implementations of the same standard, for example, some HL7 Version 2 implementations, do not interoperate. Therefore there is a need to address the interoperability problem at the semantic level. Semantic interoperability is the ability for information shared by systems to be understood at the level of formally defined domain concepts so that the information is computer processable by the receiving system [10].

In this paper, we describe an engineering approach developed within the scope of the Artemis project [1] to provide the exchange of meaningful clinical information among healthcare institutes through semantic mediation. The proposed framework, called AMEF (Artemis Message Exchange Framework) involves first providing the mapping of a source ontology into a target ontology with the help of a mapping tool which produces a mapping definition. This mapping definition is then used to automatically transform the source ontology message instances into target message instances.

Through a prototype implementation, we demonstrate how to mediate between HL7 Version 2 and HL7 Version 3 messages. However, the framework proposed is generic enough to mediate between any incompatible healthcare standards that are currently in use.

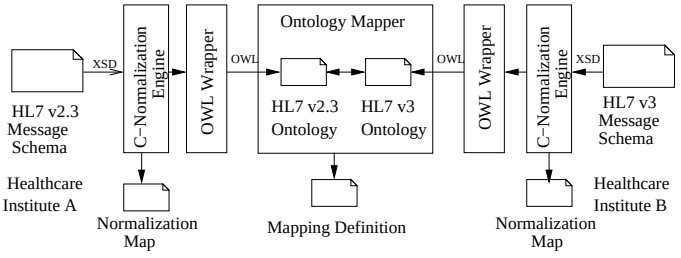


**Figure 1: Message Schema Mapping Process**

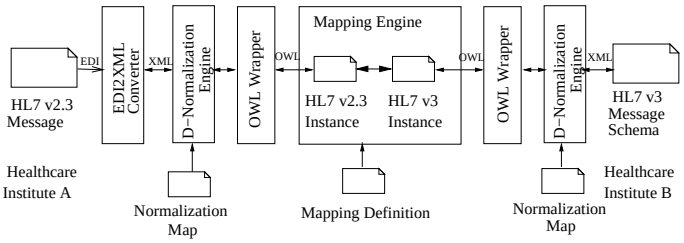


**Figure 2: Automatic Message Instance Transformation Process**

The semantic mediation between HL7 Version 2 and HL7 Version 3 messages is realized in two phases:

- *Message Ontology Mapping Process*: In the first phase, the message ontologies of two healthcare institutes are mapped one another (Figure 1). Assume that healthcare institute A uses HL7 v2 and healthcare institute B uses HL7 v3 to provide system interconnection. The message ontologies of these institutes are mapped one into other by using an ontology mapping tool. For this

*This work is supported by the European Commission through IST-1-002103-STP Artemis project and in part by the Scientific and Technical Research Council of Turkey (TÜBİTAK), Project No: EEEAG 104E013

purpose we have developed an OWL (Web Ontology Language) ontology mapping tool, namely, OWLmt [16]. With the help of a GUI, OWLmt allows to define semantic mappings between structurally different but semantically overlapping OWL ontologies, and produces a "Mapping Definition".

Since message ontologies for HL7 messages do not exist yet, we use the HL7 Version 2 and Version 3 XML Schemas (XSDs) [19] to generate OWL ontologies. This process, called "Conceptual Normalization" [5] produces a "Normalization map" describing how a specific message XSD is transformed into the corresponding OWL schema.

The "Mapping Definitions" and the "Normalization map" produced in the first phase are used during the second phase to automatically transform the message instances one into another.

- *Message Instance Mapping*: In the second phase (Figure 2), first the XML message instances of healthcare institute A are transformed into OWL instances by using the "Data Normalization" engine [5]. Note that if the message is in EDI (Electronic Data Interchange) format, it is first converted to XML. Then by using the *Mapping definition*s, OWL source (healthcare institute A) messages instances are transformed into the OWL target (healthcare institute B) message instances. Finally the OWL messages are converted to XML again through the "Data Normalization" engine.

| RQC Request Clinical Information | |
|---|---|
| MSH | Message Header |
| QRD | Query Definition |
| [ QRF ] | Query Filter |
| { | |
| PRD | Provider Data |
| [{ CTD }] | Contact Data |
| } | |
| PID | Patient Identification |
| [{ NK1 }] | Next of Kin/Associated Parties |
| [{ GT1 }] | Guarantor |
| [{ NTE }] | Notes and Comments |

| RCI Return Clinical Information | |
|---|---|
| MSH | Message Header |
| MSA | Message Acknowledgment |
| [ QRF ] | Query Filter |
| { | |
| PRD | Provider Data |
| [{ CTD }] | Contact Data |
| } | |
| PID | Patient Identification |
| [{ DG1 }] | Diagnosis |
| [{ DRG }] | Diagnosis Related Group |
| [{ AL1 }] | Allergy Information |
| [ | |
| { | |
| OBR | Observation Request |
| [{ NTE }] | Notes and Comments |
| [ | |
| { | |
| OBX | Observation Result |
| [{ NTE }] | Notes and Comments |
| } | |
| ] | |
| } | |
| ] | |
| [{ NTE }] | Notes and Comments |

**Figure 3: The Structures of the RQC/RCI EDI messages for the HL7 Version 2 event I05**

The paper is organized as follows: In Section 2, we briefly summarize the HL7 standard. Section 3 describes the semantic mediation of HL7 v2 and v3 messages. The details of OWL mapping tool used in the mediation is presented in Section 4. Transforming HL7 v2 EDI messages to XML is briefly introduced in Section 5. Finally Section 6 describes the "Normalization" tool used and the improvements realised on this tool.

## 2. HEALTH LEVEL 7 (HL7) STANDARD

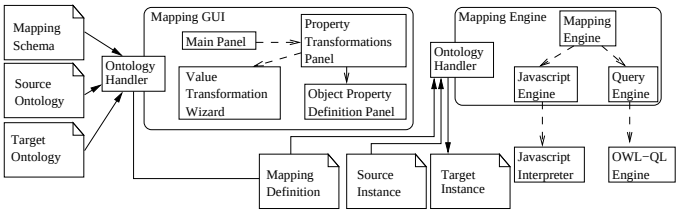


**Figure 4: Architecture of OWLmt**

The primary goal of HL7 is to provide standards for the exchange of data among healthcare computer applications. The standard is developed with the assumption that an event in the healthcare world, called the *trigger event*, causes exchange of messages between a pair of applications. When an event occurs in an HL7 compliant system, an HL7 message is prepared by collecting the necessary data from the underlying systems and it is passed to the requestor, usually as an EDI message. For example, as a result of a trigger event, say "I05", the clinical patient information for a given patient identifier is passed to the requestor as shown in Figure 3. Clinical information refers to the data contained in a patient record such as problem lists, lab results, current medications, family history, etc. [7].

HL7 version 2 is the most widely implemented healthcare informatics standard in the world today. Yet being HL7 Version 2 compliant does not imply direct interoperability between healthcare systems. Version 2 messages, contain many optional data fields. For example every attribute presented in square brackets in Figure 3, denotes optional information that may be omitted. This optionality provides great flexibility, but necessitates detailed bilateral agreements among the healthcare systems to achieve interoperability.

To remedy this problem, HL7 has developed Version 3 [6] which is based on an object-oriented data model, called Reference Information Model (RIM) [8]. The main objective of the HL7 Version 3 is to eliminate the optionality. RIM is used as the source of the content of messages and this results in a more efficient message development process. The result of the Version 3 process is the Hierarchical Message Definition (HMD), which defines the schema of the messages based on the RIM classes. Note that HL7 Version 3 messages do not interoperate with HL7 Version 2 messages.

## 3. SEMANTIC MEDIATION OF HL7 V2 AND V3 MESSAGES

In Artemis Message Exchange Framework (AMEF), the semantic mediation of HL7 v2 and v3 messages is realized in two phases:

- *Message Schema Mapping Process*: In the first phase, the message schemas of two healthcare institutes are mapped one another through semantic mediation as shown in Figure 1. At the heart of this process is the OWL Mapping tool, OWLmt, transforming OWL ontologies one into other.

  The OWL ontologies corresponding to the message schemas involved are generated through a set of available tools. First, for healthcare institute A (Figure 1), the HL7 Version 2 XML Schemas (XSDs) [19] are

converted to RDFS (Resource Description Framework Schema) by using the Conceptual Normalization (C-Normalization) engine of the Harmonise project [5]. This process uses a set of heuristics as described in Section 6 and produces a "Normalization map" describing how a specific HL7 Version 2 message XSD is transformed into the corresponding RDFS schema and vice-versa. Then, by using the OWL Wrapper, which we developed using Jena API [11], RDFS Schemas are transformed to OWL.

On the other hand, for healthcare institute B (Figure 1), in order to generate the XSDs of HL7 v3 messages, RoseTree tool of HL7 is used [18]. RoseTree allows the user to graphically build a HMD (Hierarchical Message Definition) from the Reference Information Model of HL7 v3. This generated HMD file describes the structure of the v3 XML messages, but it is not in XSD format. In order to translate the HMD file to XSD, "HL7 v3 Schema Generator" [9] is used.

The next step is to map the source ontology into the target ontology by using OWLmt. This process is described in detail in Section 4.

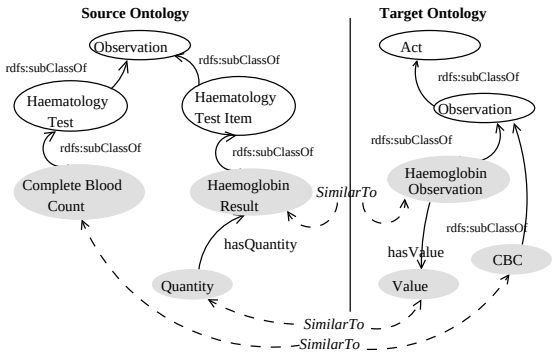


**Figure 5: Mapping between HL7 v2 and HL7 v3 message structures**

- *Message Instance Mapping*: In the second phase (Figure 2), first the HL7 version 2 EDI messages are converted to XML. The open-source programming library from HL7, namely, HL7 application programming interface (HAPI) [4] is used for transforming the EDI messages into their XML representations.

  In the next step, as shown in Figure 2, the XML message instances of healthcare institute A are transformed to OWL instances by the "Data Normalization (D-Normalization) engine [5] using the "Normalization map" produced during the first phase.

  Then by using the *Mapping definition*s, OWLmt transforms OWL source (healthcare institute A) messages instances into the OWL target (healthcare institute B) message instances. Finally the OWL messages are converted to the XML format that the healthcare institute B understands, again through the "Data Normalization" engine as shown in Figure 2.

In the following sections, we describe how these tools realize the described functionality.

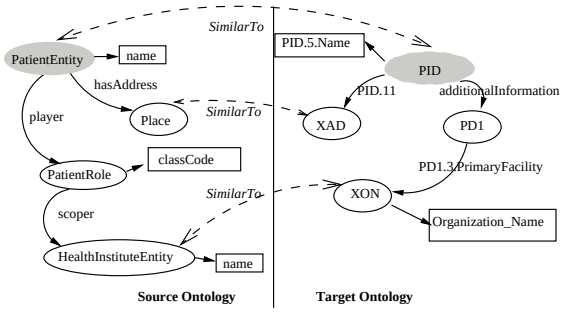


**Figure 6: Mapping Object properties**

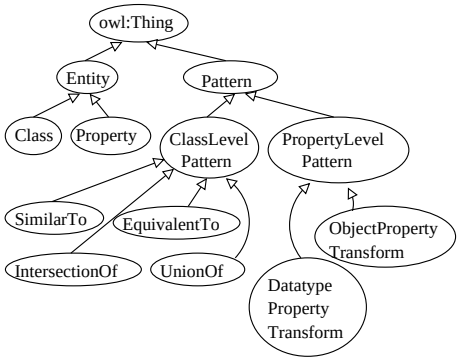


**Figure 7: OWL Mapping Schema**

# 4. OWL MAPPING TOOL: OWLMT

We have developed an OWL mapping tool, called OWLmt, to handle ontology mediation by mapping the OWL ontologies in different structures and with an overlapping content one into other. The architecture of the system, as shown in Figure 4, allows mapping patterns to be specified through a GUI tool based on a Mapping Schema. The Mapping Schema, as shown in Figure 7, is also defined in OWL.

Mapping patterns basically involve the following:

- *Matching the source ontology classes to target ontology classes*: In order to represent the matching between the classes of source and target ontologies, we have defined four mapping patterns: *EquivalentTo*, *SimilarTo*, *IntersectionOf* and *UnionOf*. Two identical classes are mapped through *EquivalentTo* pattern. *SimilarTo* implies that the involved classes have overlapping content. How the similar classes are further related is detailed through their data type properties and object properties by using "property mapping patterns". As an example, in Figure 5, the "HaemeglobinResult" class in HL7 v2 ontology is defined to be similar to "HaemoglobinObservation" class in HL7 v3 ontology. The mappings of the "hasQuantity" and "hasValue" object properties of these classes are handled by defining an "ObjectPropertyTransform" pattern between these properties.

  The *IntersectionOf* pattern creates the corresponding instances of the target class as the intersection of the declared class instances. Similarly, the UnionOf pat-

tern implies the union of the source classes' instances to create the corresponding instances of the target class. Furthermore, a class in a source ontology can be a more general (super class) of a class in the target ontology. In this case, which instances of the source ontology makes up the instances of the target ontology is defined through KIF (Knowledge Interchange Format) [13] conditions to be executed by the OWLmt mapping engine. When a source ontology class is a more specific (sub class) of a target ontology class, all the instances of the source ontology qualify as the instances of the target ontology.

- *Matching the source ontology Object Properties to target ontology Object Properties*: In addition to matching a single object property in the source ontology with a single object property in the target ontology, in some cases, more than one object properties in the source ontology can be matched with one or more object properties in the target ontology. Consider the example given in Figure 6. According to the HL7 v3 specifications, two entities, "Patient" and "HealthInstitueEntity" are connected by a "Role" class which is "PatientRole" in this case. On the other hand, in the target ontology, the "XON" class in HL7 v2.x represents the healthcare facility that a patient is registered. "PD1" (Patient Demographics 1) gives the patient information. "XON" is connected to the "PD1" by the "PD1.3.PrimaryFacility" object property. As it is clear from this example, relating a single object property in source ontology with a single object property in the target ontology does not suffice: There may be paths consisting of object property relations in the source and target ontologies that need to be mapped.

  OWLmt allows defining "ObjectPropertyTransform" pattern which represents the path of classes connected through object properties such that whenever a path defined in the source ontology (inputPath) is encountered in the source ontology instance, the path defined for target ontology (outputPath) is created in the target ontology instance. Paths are defined as triples in KIF [13] format and executed through the OWL-QL [17] engine. For example, assuming the path defined in the source ontology (Figure 6):

  (rdf:type ?x PatientEntity) (player ?x ?y)

  (rdf:type ?y PatientRole) (scoper ?y ?z)

  (rdf:type ?z HealthInstituteEntity).

  and assuming that it corresponds to the following path in the target ontology:

  (rdf:type ?x PID) (additionalInformation ?x ?y) (rdf:type ?y PD1) (PD1.3.PrimaryFacility ?y ?z) (rdf:type ?z XON)

  OWLmt constructs the specified paths among the instances of the target ontology in the execution step based on the paths defined among the instances of the source ontology.

- *Matching source ontology Data Properties to target ontology Data Properties*: Specifying the "DatatypePropertyTransform" helps to transform data type properties of an instance in the source ontology to the corresponding data type properties of instance in the target ontology. Since the data type properties may be structurally different in source and target ontologies, more complex transformation operations may be necessary than copying the data in source instance to the target instance. XPath specification [20] defines a set of basic operators and functions which are used by the OWLmt such as "concat", "split", "substring", "abs", and "floor". In some cases, there is a further need for a programmatic approach to specify complex functions. For example, the use of conditional branches (e.g. if-then-else, switch-case) or iterations (e.g while, for-next) may be necessary in specifying the transformation functions. Therefore, we have added JavaScript support to OWLmt. By specifying the JavaScript to be used in the "DatatypePropertyTransform" pattern, the complex functions can also be applied to the data as well as the basic functions and the operators provided by XPath.

## 4.1 OWLmt Mapping Schema

The mapping patterns used in the OWLmt are defined through an OWL ontology called "Mapping Schema". Each mapping pattern is an owl:class in the "Mapping Schema" as shown in Figure 7. The additional information needed in the execution of the patterns are provided as KIF [13] expressions such as *inputPath*s and *outputPath*s. The *inputPath* and *outputPath* are data type properties of "ObjectProperty Transform Pattern" class and hold the query strings in the KIF format which are used in the execution to query the source ontology instances in order to build the target instances.

Each mapping relation specified through OWLmt GUI represents as an instance of these pattern classes, and the final the mapping definition is stored as an instance of the "Mapping Scheme" as a collection of pattern class instances.

In Figure 8, a part of the mapping definition of the example in Figure 6 is presented. First the *SimilarTo* relationship between the "Patient" and "PatientEntity" classes are represented with an instance of *SimilarTo* pattern. Then through an "ObjectPropertyTransform" pattern instance, the relationships between object properties linking the "PatientEntity" to "HealtInstituteEntity" classes and the object property linking the "PID" to "XON" classes are represented. Further details of the mapping tool are presented in [2].

This mapping definition is given as an input to the OWLmt Mapping Engine, which translates source ontology instances to target ontology instances.

## 4.2 OWLmt GUI

OWLmt GUI [16] consists of five components: Ontology Handler, Main Panel, Property Transformations Panel, Value Transformation Wizard and Object Property Definition Panel. The Ontology Handler is used in parsing and serializing the ontology documents. The class mapping patterns are defined in the main panel. The property mapping patterns are defined in the property transformation panel. This panel lets the user to create new property mapping patterns such as the "ObjectPropertyTransform" and "DatatypePropertyTransform". The value transformation wizard is used to configure a "DatatypePropertyTransform" pattern. By using this wizard, the functions used in the value transformation of the data type properties can be spec-

```
<SimilarTo rdf:ID="SimilarTo_1">
 <similarToInput>
    <relatedTo rdf:resource=#PatientEntity/>
 </similarToInput>
 <similarToOutput>
    <relatedTo rdf:resource=#PID/>
 </similarToOutput>
 <operationName>PatientEntity_SimilarTo_PID</operationName>
</SimilarTo> .......

<ObjectPropertyTransform rdf:ID="ObjectPropertyTransform_1">
  <operationName>ObjectPropertyTransform_1</operationName>
  <includedIn rdf:resource=#SimilarTo_1/>
  <inputPath>(rdf:type ?x PatientEntity) (player ?x ?y)
       (rdf:type ?y PatientRole)   (scoper ?y ?z)
        (rdf:type ?z HealthInstituteEntity)
  </ inputPath>
  <outputPath>(rdf:type ?x PID) (additionalInformation ?x ?y)
          (rdf:type ?y PD1) (PD1.3.PrimaryFacility ?y ?z)
          (rdf:type ?z XON)
  </ outputPath>
</ObjectPropertyTransform>
```

**Figure 8: An Example Mapping Definition**

ified.

## 4.3 OWLmt Engine

The mapping engine is responsible for creating the target ontology instances using the mapping patterns given in the Mapping Definition and the instances of the source ontology. It uses OWL Query Language (OWL-QL) to retrieve required data from the source ontology instances. OWL-QL is a query language for OWL developed at the Stanford University [17]. While executing the class and property mapping patterns, the query strings defined through the mapping GUI are send to the OWL-QL engine with the URL of the source ontology instances. The query engine executes the query strings and returns the query results.

The OWL-QL engine uses the JTP (Java Theorem Prover) reasoning engine [12], an object-oriented modular reasoning system. The modularity of the system enables it to be extended by adding new reasoners or customizing existing ones.

The use of the OWL-QL enables OWLmt to have reasoning capabilities. When querying the source ontology instances or while executing the KIF [13] patterns, OWL-QL reasons over the explicitly stated facts to infer new information. As an example, consider two instances, I1 and I2, which are the members of the classes C1 and C2 respectively. If these two instances are related with the "owl:sameAs" construct, one of them should be in the extension of the intersection class, say C3, of the classes C1 and C2. Hence, the *IntersectionOf* pattern transforms the instance I1 and I2 to the instance I3 which is a member of C3 in the target ontology. However, assume that there is no direct "owl:sameAs" construct but there is a functional property which implies that these two instances are the same. The reasoning engine can infer from the definition of the "owl:FunctionalProperty" by using the rule:

```
(rdf:type ?prop owl:FunctionalProperty)
(?prop ?instance ?I1)
(?prop ?instance ?I2)
->
(owl:sameAs ?I1 ?I2)
```

that the instances I1 and I2 are the same instance resulting in the instance I3 to be in the target ontology.

After executing the class mapping patterns, the mapping engine executes the property mapping patterns. Similar to the class mapping patterns, OWL-QL queries are used to locate the data. In order to perform value transformations, the mapping engine uses the JavaScripts in the "DatatypePropertyTransform" pattern. To execute the JavaScripts, an interpreter is used. The engine prepares the JavaScript by providing the values for the input parameters and sends it to the interpreter. The interpreter returns the result, which is then inserted as the value of the data type property in the target ontology instance.

# 5. EDI TO XML CONVERSION IN HL7

There are several commercial and open-source programming libraries that implement the HL7 standards. In our implementation, HAPI [4] (HL7 Application Programming Interface) Assembler/Disassembler Tool is used to transform the HL7 v2 EDI messages into their XML representations. HAPI provides open source libraries for parsing and manipulating both EDI and XML messages that are HL7 conformant. Furthermore the library enables message validation, that is, enforcement of HL7 data type rules for the values in the messages.

# 6. NORMALIZATION TOOL

As previously mentioned, currently the healthcare application messages are usually in XML or EDI format (which can be converted to XML). Hence there is a need for automatic bidirectional transformation of XML message instances to OWL message instances as well as automatic generation of OWL Schemas from XML Schema Definitions (XSDs). Such a transformation, called Normalization, has been realized within the scope of the Harmonise project [5].

The first step in the "Normalization" process is generating RDFS schemas from local XSD schemas. This step is called Conceptual Normalization (C-Normalization) phase where the C-Normalization engine parses the XML Schema, and using a set of predefined "Normalization Heuristics", creates the corresponding RDFS schema components for each XML Schema component automatically. Normalization Heuristics define how specific XML Schema construct can be projected onto a RDFS construct (entity or set of related entities) [5]. With this process, the complex type, element and attribute definitions of the XSD are represented as classes, and properties in the RDFS ontology. One of the "Normalization Heuristics" called "ComplexType2Class" projects each complex type definition in XSD onto a class definition in RDFS. Furthermore, the attribute definitions and the element definitions in XSD are converted to the "rdf:Property" by the "Attribute2Property" and "Element2Property" heuristics, respectively. After representing the complex types as classes and elements as properties, the domain and range of the properties are set. The "ElementParent2PropertyDomain" heuristic sets the domain of the property to the class which corresponds to the parent of the element in the XSD. Furthermore, the "ElementType2PropertyRange" heuristic sets the range of the property to the class which corresponds to the type of the element in the XSD as illustrated in Figure 9. The C-Normalization process produces a "Normalization Map" which defines the associations between the XML Schema and the re-engineered RDFS model. Further details

of this work are available in [5].

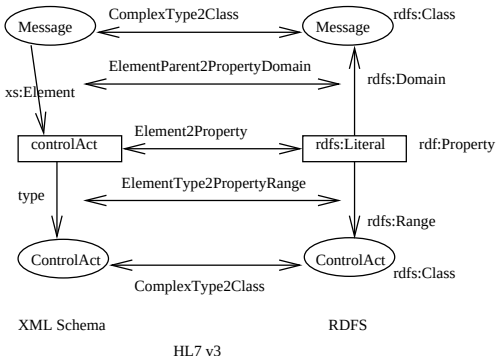


**Figure 9: C-Normalization Phase**

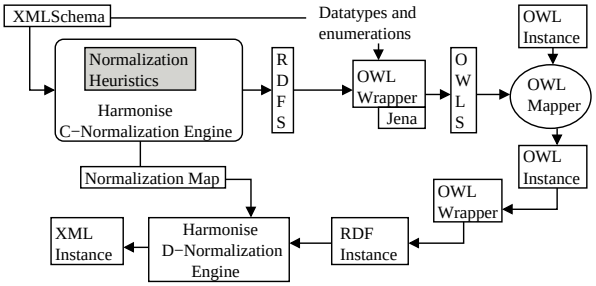


**Figure 10: Normalization process for the bidirectional transformation of XML instances to OWL instances**

The second step in "Normalization" is the Data Normalization Process (D-Normalization) which is used for transforming the data instances from XML to OWL or OWL to XML.

In Artemis architecture, we have used the Harmonise Normalization Engine. However since we need OWL Schemas instead of RDFS schemas, we developed an OWL wrapper using Jena API to create OWL schemas from the RDFS files after the C-Normalization step. Additionally in the D-Normalization step, through the same wrapper, the generated RDF instances are further translated in to OWL instances or vice versa as depicted in Figure 10.

Note that in Harmonise C-Normalization step, the enumeration of property values or basic data types defined in XML Schemas cannot be preserved. To handle this, the OWL Wrapper developed carries the enumeration of property values and basic data types to the OWL Schema. The enumerated classes are represented using <owl:oneOf rdf:parseType="Collection"> construct in case of enumerated classes, and using <owl:oneOf> and <rdf:List> constructs in case of enumerated data types. The data types are represented by referring to XML Schema data types using RDF data typing scheme.

## 7. CONCLUSIONS

One of the most challenging problems in the healthcare domain today is providing interoperability among healthcare information systems. In order to tackle this problem, we propose an engineering approach to semantic interoperability within the scope of the Artemis project. For this purpose, the existing applications are wrapped as Web services and the messages they exchange are annotated with OWL ontologies which are then mediated through an ontology mapping tool developed, namely, OWLmt. One of the major contributions of the OWLmt is the use of OWL-QL engine which enables the mapping tool to reason over the source ontology instances while generating the target ontology instances according to the graphically defined mapping patterns.

## 8. REFERENCES


[1] Artemis A Semantic Web Servicebased P2P Infrastructure for the Interoperability of Medical Information Systems, http://www.srdc.metu.edu.tr/-webpage/projects/artemis/.

[2] Bicer, V., "OWLmt: OWL Mapping Tool", M.Sc. Thesis, Dept. of Computer Eng., METU, in preparation.

[3] ENV 13606:2000 "Electronic Healthcare Record Communication", http://www.centc251.org/TCMeet/-doclist/TCdoc00/N00048.pdf.

[4] HL7 Application Programming Interface (HAPI), http://hl7api.sourceforge.net

[5] Harmonise, IST200029329, Tourism Harmonisation Network, Deliverable 3.2 Semantic mapping and Reconciliation Engine subsystems.

[6] Health Level 7, http://www.hl7.org.

[7] HL7, Chapter 11 Patient Referral, http://www.hl7.org/library/General/v231.zip

[8] HL7 Reference Information Model (RIM), http://www.hl7.org/library/data-model/RIM/-modelpage_mem.htm.

[9] HL7 v3 Schema Generator, http://www.hl7.org/-library/data-model/V3Tooling/toolsIndex.htm

[10] ISO/TS Health Informatics - Requirements for an electronic health record architecture, Technical Specification, International Organization for Standardization (ISO), Geneva, Switzerland, 2004.

[11] Jena Framework, http://jena.sourceforge.net/ .

[12] Java Theorem Prover (JTP), http://www.ksl.stanford.edu/software/JTP/ .

[13] Knowledge Interchange Format (KIF), http://logic.stanford.edu/kif/kif.html .

[14] A. Maedche, D. Motik, N. Silva, R. Volz, "MAFRA-A MApping FRAmework for Distributed Ontologies", In Proc. of the 13th European Conf. on Knowledge Engineering and Knowledge Management EKAW-2002, Madrid, Spain, 2002.

[15] OpenEHR Foundation, http://www.openehr.org/ .

[16] OWL Mapping Tool (OWLmt), http://www.srdc.metu.edu.tr/artemis/owlmt/

[17] OWL Query Language, http://ksl.stanford.edu/projects/owlql/

[18] Rose Tree, http://www.hl7.org/library/data-model/-V3Tooling/toolsIndex.htm

[19] XML encoding rules of HL7 v2 messages - v2.xml, http://www.hl7.org/Special/Committees/xml/drafts/-v2xml.html

[20] XML Path Language, http://www.w3.org/TR/xpath .