Mapping UML Sequence Diagram into the Web Ontology Language OWL

Mo'men Elsayed[1], Nermeen Elkashef[2]
Department of Mathematics and Computer Science
Faculty of Science, Alexandria University
Alexandria, Egypt

Yasser F.Hassan[3]
Professor of Computer Science
Faculty of Computer Science and Artificial
Intelligence, Pharos University, Alexandria, Egypt

***Abstract*—In this paper, we propose a new mapping technique from the OMG's UML modeling language into the Web Ontology Language (OWL) to serve the Semantic Web. UML (Unified Modeling Language) is widely accepted and used as a standardized modeling language in Object-Oriented Analysis (OOA) and Design (OOD) approach by domain experts to model real-world objects in software development. On the other hand, the conceptualization, which is represented in OWL, is designed to process the content of information rather than just present the information. Therefore, the matter of migrating UML to OWL is becoming an energetic research domain. OWL (Web Ontology Language) is a Semantic Web language designed for defining ontologies on the Web. An ontology is a formal specification naming and definition of shared data. This technique describes how to map UML Models into OWL and allows us to keep semantic of UML sequence diagrams such as messages, the sequence of messages, guard invariant, etc. to make data of UML sequence diagrams machine-readable.**

***Keywords*—Mapping; Unified Modeling Language; UML; sequence diagram; ontology; Web Ontology Language; OWL**

I. Introduction

Nowadays we are observing a growing effort for supporting semantics of data that is stored on the web that makes the web more intelligent that procreate a promising technology is called "Semantic Web". The Semantic Web is developed by W3C for providing the knowledge data interchange over the web that available standard formats, reachable, manageable, and understand which will be used by machines for overcoming the original web limitation of only interchanging data through documents [1]. The semantic web is also called "Web of data". The web will be able to process and explicate information to be better to meet the human requirement and able to provide full and immediate answers for natural language queries. Ontology and also the various languages designed for sharing data: Extensible Markup Language (XML), Resource Description Framework (RDF) and Web Ontology Language (OWL) are the stilts of this technology [2].

The ontology lies at the core of semantic data integration. The ontology is the knowledge data domain that will be shared and explored. The Semantic Web does not break away from the web but rather an extension of the current one, where data is given distinct meaning, better-enabling computers and people to work in cooperation [3]. Therefore, the issue of converting UML to OWL is becoming an active research point.

UML, the Unified Modeling Language, is the most utilized language to the requirements specification [4]. UML is a standardized modeling graphical language that includes an integrated set of diagrams. Every diagram depicts the modeling system in different portions, but together they can provide a full map of the modeling system. We focus on one diagram that is a UML sequence diagram. A UML sequence diagram is a type of interaction diagram where it illustrates how and in what order a set of objects works together. A sequence diagram is employed for dealing with the dynamic view of a system, whereas OWL is developed to form a semantic web to represent the explicit specification of a conceptualization, not just a document web. A picture is worth a thousand words, this idiom definitely fits describing UML. UML is a standard notation language that can be used for specifying, visualizing, constructing, and documenting the phases of software systems.

Ontology means an ontology may be a characterization (like a suitable specification of a program) of concepts and relationships which is able to exist among them through a community. In other words – an ontology illustrates a part of the globe.

This paper is organized as follows: Section 2 offers the background of our work. In Section 3 discusses our technique for mapping UML sequence diagrams into OWL 2 DL in detail. Section 4 presents an overview of our technique with a running case study. Section 5 concludes and points out the fields of future work.

II. Background

A. Unified Modeling Language

UML is a standardized blueprint representation to design and analyze a model of a system. These blueprints provide more than 10 diagrams in which every diagram supports an aspect to characterize every part of a system.

UML is developed to make available communication among the software developers by specifying, visualizing, constructing, and documenting the aspects of software systems. UML includes things, relationships, and diagrams, as shown in Table I. One exceptional diagram is a sequence diagram which is categorized as a behavioral diagram. Behavior diagrams depict a dynamic aspect of the objects in a

system, a chain of changes to the system through time. A sequence diagram is one of the interaction diagrams that illustrate the interactions between object instances and the ordering of messages according to time. A sequence diagram illustrates the interactions among objects, and emphasizes a sequential order of messages.

B. Ontology Foundations

An ontology is a specification of a conceptualization [3]. The term is borrowed from philosophy, where an ontology can be a methodical account of existence. The meaning depends on our understanding what "exists" is that which can be represented wherever the terms "specification" and "conceptualization" is a description (like a formal specification of a program) of concepts and describable relationships between them that can reflect in the representational vocabulary with which an abstraction of a program. Typically by using UML, we can represent the abstract of a program. For example, A UML object o is the object of class C is drawn by using UML, as given in Fig. 1.

In ontologies, the concepts of the program are represented in a set of axioms that depict the specification of a conceptualization [4], and the relationship among concepts as properties. Therefore, the UML Class C is mapped to OWL 2 as

Declaration (Class(C))

In the OWL the UML object o of class C is called an individual and is expressed as a class assertion:

ClassAssertion(C o)

TABLE I. UML THINGS, RELATIONSHIPS AND DIAGRAMS

UML Category	UML Elements	UML Elements
Things	Structural Things	Classes Interfaces Collaborations Use Cases Active Classes Components Nodes
	Behavioral Things	Interactions State Machines
	Grouping Things	Packages
	Annotational	Things Notes
Relationships	Structural Relationships	Dependencies Aggregations Associations Generalizations
	Behavioral Relationships	Communicates Includes Extends Generalizes
Diagrams	Structural Diagrams	Class Diagrams Component Diagrams Deployment Diagrams Use Case Diagrams
	Behavioral Diagrams	Sequence Diagrams Communication Diagrams Statechart Diagrams Activity Diagrams

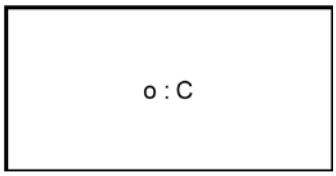

Fig. 1. A UML Object o of Class C.

C. OWL and UML

The Ontology Definition Metamodel (ODM) [5] is derived from Meta-Object Facility (MOF) [5,6] and based on the Object Management Group (OMG) [7] specifications that permit integrating ontology engineering into concepts of OMG modeling. The ODM follows an identical hierarchy just like the one mentioned in a four-layer OMG modeling hierarchy [8].

The ODM was primarily developed to support the ontology structure [5]. It contains classes, associations, and constraints.

MOF defines an abstract framework and language for constructing, managing, and specifying technology-neutral metamodels. It is the foundation for defining any modeling language like UML. Consequently, The UML is also derived from the MOF and fundamental form of ODM. Therefore, UML notations are also used for ontology modeling [6, chapter 7].

The ODM provides metamodels for several knowledge representation languages such as OWL and RDF [8]. In our technique, ODM metamodel OWL is used to represent the MOF / UML based models. In our technique, we use a decidable fragment of OWL 2 DL.

Whereas the ODM and UML are derived from MOF [5,6], therefore there exist common features, as well as there, are also different features. The common features are shown in Table II. The comparison between UML and ODM aforementioned is given in terms of UML and OWL 2 DL. Table III shows the features in UML which do not have equivalent OWL 2 elements.

TABLE II. UML ELEMENTS THAT HAVE THE DL EQUIVALENT OWL 2 DL ELEMENTS

UML Elements	OWL Elements
Class	Class
Instance	Individual
Enumeration	Oneof
Multiplicity	Min/max/exact cardinality
Datatype	Datatype

TABLE III. UML ELEMENTS THAT HAVE THE DL EQUIVALENT OWL 2 DL ELEMENTS

UML Elements	OWL Elements
Ordering	Not available
Messages	Not available
Operations	Not available
Guards	Not available
Fragment	Not available
Operands	Not available

D. Reasoners

Reasoning is a critically important capability for the Semantic Web application development. A reasoner plays a vital role in developing that automatically infers logical consequences from a collection of logical facts or axioms such as Pellet, FaCT++, HerMiT, etc.

Pellet [9] provides functionality to check consistency and infer subsumption of ontologies and Semantic Web Rule Language. Pellet is a complete and capable OWL-DL reasoner, which is written in Java and is open source. Based on these criteria, we have chosen Pellet [9] which satisfies our requirements to reasoner processes the ontology and generates a validation report.

III. UML Sequence Diagrams into OWL 2 DL

In order to create better OWL 2 ontologies from a UML sequence model, we explain in this section our understanding of UML sequence diagram concepts and show how to migrate these concepts into OWL 2.

The UML sequence diagrams capture the dynamic behavior of a system. The sequence diagram provides the behavioral interface of object instances and the sequence of messages that they send to each other over time by using the vertical axis of the diagram to show time what messages are sent and what.

A. Classes and Objects

A concept that grouped multiple objects that have the same features and share the same behaviors is commonly known as a class in UML. The concept of class in UML is equivalent to the concept of class in OWL 2 because both concepts are similar. A UML Class C is mapped to OWL 2 as:

Declaration(Class(C))

An object is an instance (or element) of a class. In UML, objects have the behaviors of their class. Each object in a sequence diagram belongs to a specific class in a class diagram. A UML object o of class C is mapped into ClassAssertion axiom in OWL 2 called an individual:

ClassAssertion(C o)

Furthermore, By default in a UML sequence diagram, every object is different from another. However, OWL 2 follows the open-world assumption, so we must mention that all individuals are different from each other. For example, objects o1, …, on in a sequence diagram, we use the DifferentIndividuals axiom in OWL 2:

DifferentIndividuals(o1 … on)

B. Sequence of Messages

Messages that depict the call of operations belong to a specific class are shown horizontally in Fig. 2. They are sent from a source object that is defined as a caller and received by a target object is defined as a receiver in a sequence diagram. A vertical position indicates the sequence of the messages of the sequence diagram, wherever the first message is always shown at the top in the diagram. Next, subsequent messages are added to the sequence diagram little down from the earlier message.

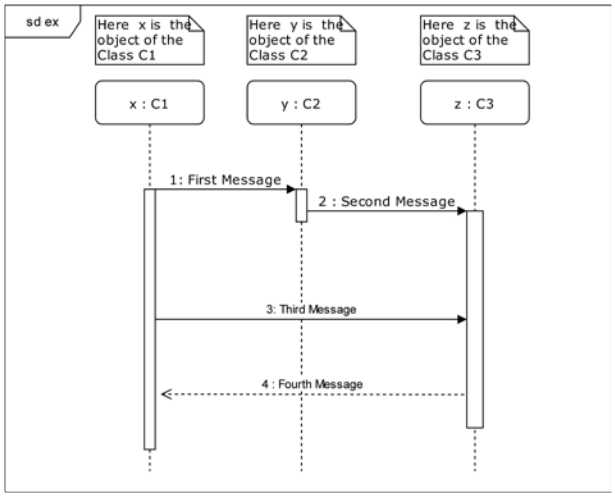

Fig. 2. A Simple UML Sequence Diagram.

A Sequence of messages is a number of messages that come one after another in a particular order as shown in Fig. 2. OWL 2 contains no axioms specifically for defining sequence or ordering [10]. However, OWL 2 has axioms that can be used to model sequence. Our technique describes a design pattern for modeling a sequence of messages using OWL 2 axioms, as summarized in the diagram in Fig. 3.

Before starting the conversion, we create a class OWL called “Fragment” to represent the sequence of messages in the UML sequence diagram. The class “Fragment” represents the kind of fragment in the UML sequence diagram. The object property “hasOperand” connects operands to the class “Fragment” which contains the operands of it. The class “Operand” represents the sequence of elements in the UML sequence diagram that can have multiple elements that is executed by the class “Element” and the object property “hasMessage”/”hasFragment”. The class “Operand” specifies a particular size refers to the number of messages in the operand. As shown in Fig. 3, the object property “hasNext” connects an individual of the class “Element” to exactly another one. Consequently, the object property “hasNext” supports the sequence of elements. Furthermore, “hasNext” is accompanied by its related transitive properties and inverse. Two object properties are defined, “firstElement” and “lastElement”, to determine which are the first and the last elements in the class “Operand”, as sub-properties of “element”. The elements connected with these two properties cannot be respectively preceded or followed by another element. Moreover, the class “Element” represents the type of elements in the operand this is executed by the class “Message” and “Fragment” respectively with the object property “hasMessage” and “hasFragment”. The class “Message” specifies a particular type that refers to the type of a message in the UML sequence diagram. In order to identify which are the caller and the receiver objects of a message, two object properties are defined caller and receiver. The object property “next message” connects an individual of the class “Message” to exactly another one. Two objects, properties are defined, “firstMessage” and “lastMessage”, to determine which are the first and the last messages in an operand and a fragment, as sub-properties of “hasMessage”. Consequently,

the object property “nextOperand” supports the sequence of operands. Two object properties are defined, “firstOperand” and “lastOperand”, in order to determine which are the first and the last operand in the class “Fragment”.

This class and the properties are related to this class are defined as follows:

Class: Fragment
SubClassOf:
kind exactly 1,
firstOperand exactly 1 Operand,
lastOperand exactly 1 Operand
DisjointWith: Operand, Element, Message
ObjectProperty: hasOperand
Domain: Fragment
Range: Operand
SubPropertyChain: hasOperand o nextOperand
InverseOf: operandOf
ObjectProperty: followedBy
Characteristics: Irreflexive
ObjectProperty: precedeBy
Characteristics: Irreflexive
InverseOf: followedBy
ObjectProperty: firstOperand
Characteristics: Functional
SubPropertyOf: hasOperand
Domain: Fragment
Range: precedeOperand exactly 0 Operand
ObjectProperty: lastOperand
Characteristics: Functional
SubPropertyOf: hasOperand
Domain: Fragment
Range: nextOperand exactly 0 Operand
Class: Operand
SubClassOf:
size exactly 1 xsd:nonNegativeInteger,
guard max 1 xsd:string,
firstElement max 1 Element,
lastElement max 1 Element
DisjointWith: Fragment, Element, Message
DataProperty: guard
Domain: Operand
Range: xsd:string
DataProperty: size
Characteristics: Functional
Domain: Operand
Range: xsd:nonNegatveInteger
ObjectProperty: nextOperand
Characteristics: Functional
SubPropertyOf: followedBy
Domain: Operand
Range: Operand
ObjectProperty: precedeOperand
Characteristics: Functional
SubPropertyOf: precedeBy
InverseOf: nextOperand
ObjectProperty: hasElement
Domain: Operand
Range: Element
SubPropertyChain: hasElement o hasNext
InverseOf: elementOf
ObjectProperty: firstElement
Characteristics: Functional
SubPropertyOf: hasElement
Domain: Operand
Range: hasPrecede exactly 0 Element
InverseOf: firstElementOf
ObjectProperty: lastElement
Characteristics: Functional
SubPropertyOf: hasElement
Domain: Operand
Range: hasNext exactly 0 Element
InverseOf: lastElementOf
Class: Element
SubClassOf: inverse hasElement some Operand
DisjointWith: Fragment, Operand, Message
ObjectProperty: hasNext
Characteristics: Functional
SubPropertyOf: followedBy
Domain: Element
Range: Element
ObjectProperty: hasPrecede
Characteristics: Functional
SubPropertyOf: precedeBy
InverseOf: hasNext
ObjectProperty: hasFragment
Domain: Element
Range: Fragment
ObjectProperty: hasMessage
Domain: not Message
Range: Message
InverseOf: messageOf
ObjectProperty: firstMessage
Characteristics: Functional
SubPropertyOf: hasMessage
ObjectProperty: lastMessage
Characteristics: Functional
SubPropertyOf: hasMessage
Class: Message
SubClassOf:
index exactly 1 xsd:positiveInteger,
type exactly 1,
caller max 1 not Message
receiver max 1 not Message
DisjointWith: Fragment, Operand, Element
DataProperty: index
Characteristics: Functional
Domain: Message
Range: xsd:positiveInteger
ObjectProperty: caller
Characteristics: Asymmetric
Domain: Message
Range: not Message
ObjectProperty: receiver
Characteristics: Asymmetric
Domain: Message
Range: not Message
DisjointWith: caller
ObjectProperty: nextMessage
Characteristics: Functional
SubPropertyOf: followedBy
Domain: Message
Range: Message
ObjectProperty: precedeMessage
Characteristics: Functional
SubPropertyOf: precedeBy
InverseOf: nextMessage

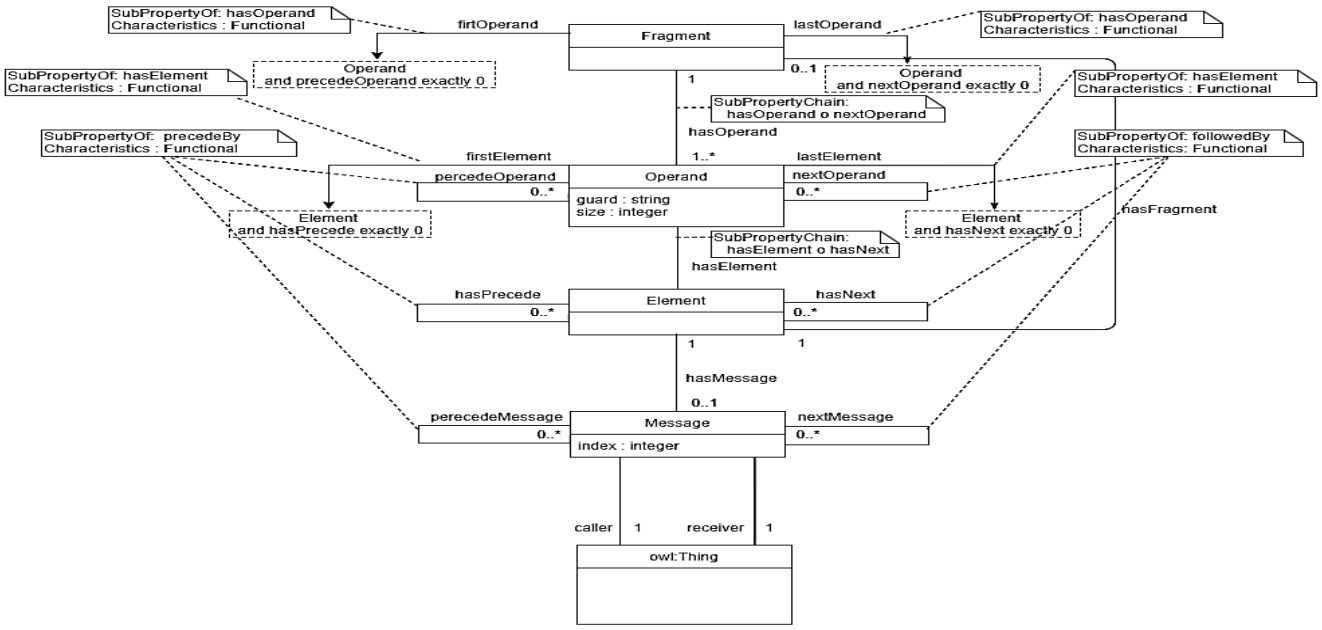

Fig. 3. Diagram Summarizing the Class "Fragment" and Its Related Classes and Properties.

The reflexive object properties "followedBy" and "precedeBy" (respectively super-"hasNext", "nextMessage" and "nextOperand", and "hasPrecede", "precedeMessage" and "precedeOperand") refers to all the elements that follow/precede a particular element. In OWL 2, no cycles are permitted. Acyclicity means that an element cannot follow or precede itself, i.e., an element e1 either follows or precedes e2, e2 follows or precedes e3 and e3 follows or precedes e1 is disallowed. A sufficient and necessary condition for acyclicity of followedBy/precedeBy is irreflexive closure of the property is transitive. We make followedBy/precedeBy irreflexive and transitive at the same time. However, to set those two properties as transitive is not possible [8] since the logic system would no longer be decidable, and we would keep the ontology in a DL framework. We map transitivity in Semantic Web Rule Language (SWRL). The transitivity of the properties" followedBy" and "precedeBy" is written in SWRL as

- followedBy(?e1, ?e2) ^ followedBy(?e2, ?e3)-> followedBy(?e1,?e3)
- precedeBy(?e1, ?e2) ^ precedeBy(?e2, ?e3) -> precedeBy(?e1,?e3)

The UML sequence diagram is mapped in a way that is possible to infer some implicit data. Leaving to a reasoner. For example, it is not necessary to specify all the operands of the fragment and all the elements of the operand. In fact, through the properties chain axiom defined in for the properties.

- hasOperand : hasOperand o nextOperand
- hasElement : hasElement o hasNext

It can specify the first (properties firstOperand and firstElement) and the last (properties lastOperand and lastElement) elements. In this technique, the reasoner will be able to infer all the remaining hasOperand and hasElement.

Moreover, the mixture of the above property chain can be very useful when combined with the subsequent SWRL rules:

- firstMessage(?o, ?m1) ^ index(?m1, 1) ^ size(?o, ?v) ^ index(?m2,?v) -> lastMessage(?o, ?m2)
- firstOperand(?f, ?o) ^ firstMessage(?o, ?m) -> firstMessage(?f,?m)
- elementOf(?e, ?o) ^ firstElementOf(?e, ?o) ^ hasMessage(?e,?m) -> firstMessage(?o, ?m)
- elementOf(?e, ?o) ^ firstElementOf(?e, ?o) ^ hasFragment(?e, ?f) ^ firstMessage(?f, ?m) -> firstMessage(?o, ?m)
- lastOperand(?f, ?o) ^ lastMessage(?o, ?m) -> lastMessage(?f, ?m)
- elementOf(?e, ?o) ^ lastElementOf(?e, ?o) ^ hasMessage(?e, ?m) -> lastMessage(?o, ?m)
- elementOf(?e, ?o) ^ lastElementOf(?e, ?o) ^ hasFragment(?e, ?f) ^ lastMessage(?f, ?m) -> lastMessage(?o, ?m)
- elementOf(?e, ?o) ^ hasMessage(?e, ?m) -> hasMessage(?o, ?m)
- hasOperand(?f, ?o) ^ hasMessage(?o, ?m) -> hasMessage(?f, ?m)
- elementOf(?e, ?o) ^ hasFragment(?e, ?f) ^ hasMessage(?f,?m) -> hasMessage(?o, ?m)
- hasNext(?e1, ?e2) ^ hasMessage(?e1, ?m1) ^ hasMessage(?e2, ?m2) -> nextMessage(?m1, ?m2)
- hasNext(?e1, ?e2) ^ hasMessage(?e1, ?m1) ^ hasFragment(?e2, ?f) ^ firstMessage(?f, ?m2) -> nextMessage(?m1, ?m2)

- nextOperand(?o1, ?o2) ^ lastMessage(?o1, ?m1) ^ firstMessage(?o2, ?m2) -> nextMessage(?m1, ?m2)
- hasNext(?e1, ?e2) ^ hasFragment(?e1, ?f1) ^ lastMessage(?f1, ?m1) ^ hasFragment(?e2,?f2) ^ firstMessage(?f2, ?m2) -> nextMessage(?m1, ?m2)
- hasNext(?e1,?e2) ^ hasFragment(?e1,?f) ^ lastMessage(?f,?m1) ^ hasMessage(?e2,?m2) -> nextMessage(?m1, ?m2)
- messageOf(?m1,?o) ^ messageOf(?m2, ?o) ^ index(?m1, ?v1) ^ add(?v2, ?v1, 1) ^ index(?m2, ?v2) ->nextMessage(?m1, ?m2)
- nextMessage(?m1, ?m2) ^ index(?m1, ?v1) ^ add(?v2, ?v1, 1) -> index(?m2, ?v2)
- Operand(?o) ^ firstMessage(?o, ?m) ^ lastMessage(?o, ?m) -> size(?o, 1)
- Operand(?o) ^ firstMessage(?o, ?m1) ^ index(?m1,1) ^ lastMessage(?o, ?m2) ^ index(?m2, ?v) -> size(?o, ?v)
- nextOperand(?o1,?o2) ^ lastMessage(?o1, ?m1) ^ index(?m1,?v1) ^ lastMessage(?o2, ?m2) ^ index(?m2,?v2) ^ subtract(?v3, ?v2, ?v1) -> size(?o2, ?v3)
- hasNext(?e1,?e2) ^ hasMessage(?e1, ?m1) ^ index(?m1,?v1) ^ hasFragment(?e2, ?f) ^ firstOperand(?f, ?o) ^ lastMessage(?o,?m2) ^ index(?m2,?v2) ^ subtract(?v3,?v2,?v1) -> size(?o, ?v3)
- hasNext(?e1, ?e2) ^ hasFragment(?e1, ?f1) ^ lastOperand(?f1,?o1) ^ lastMessage(?o1,?m1) ^ index(?m1,?v1) ^ hasFragment(?e2, ?f2) ^ firstOperand(?f2,?o2) ^ lastMessage(?o2,?m2) ^ index(?m2,?v2) ^ subtract(?v3, ?v2, ?v1) -> size(?o2, ?v3)

Let us introduce an example to show how to use our technique for describing a UML sequence diagram. Suppose one wants to describe the example, in Fig. 2. It is possible to model this scenario straightforwardly using our technique:

```
:example a :Fragment
      ; :kind “sd”
      ; :firstOperand :op1
      ; :lastOperand :op1 .
:op1 a :Operand
      ; :size “4”^^xsd:nonNegative
      ; :firstElement :el1
      ; :lastElement :el2 .
:el1 a :Element
      ; :hasMessage :first_message
      ; :hasNext :el2 .
:el2 a :Element
      ; :hasMessage :second_message
      ; :hasNext :el3 .
:el3 a :Element
      ; :hasMessage :third_message
      ; :hasNext :el4 .
:el4 a :Element
      ; :hasMessage :fourth_message .
:first_message a :Message
      ; :index “1”^^xsd:positiveInteger
      ; :type “synchronous”
      ; :caller :x
      ; :receiver :z .
:second_message a :Message
      ; :index “2”^^xsd:positiveInteger
      ; :type “synchronous”
      ; :caller :y
      ; :receiver :z .
:third_message a :Message
      ; :index “3”^^xsd:positiveInteger
      ; :type “asynchronous”
      ; :caller :x
      ; :receiver :z .
:fourth_message a :Message
      ; :index “4”^^xsd:positiveInteger
      ; :type “return”
      ; :caller :z
      ; :receiver :x .
```

C. Messages

A UML sequence diagram shows interactions with which messages are exchanged among a set of objects participate. It concentrates to determine the behavioral view of a system. There are two dimensions in the UML sequence diagram, a vertical dimension, and a horizontal dimension respectively representing time and objects participating in the interaction. The horizontal dimension also captures the message which can be a signal or a class operation call between two vertical dashed lines which are called lifetimes. Each lifetime indicates an individual participant over a time in the interaction.

A message is an abstract element that has a name. It specifies the kind of communication between two lifelines of an interaction. It does not specify only the sort of communication but also the caller and therefore the receiver. Caller and receiver are normally two occurrence specifications (points at the ends of messages). The message is shown from the caller message end to the receiver message end.

The types of communication of the message as defined in the UML are listed below.

- Synchronous Message: represents a class operation call. All other calls of the caller are blocked waiting for the receiver to have processed the message and returned.
- Asynchronous Message: the caller of message does not need to wait for a replay to continue. Like synchronous messages.
- Reply Message: it is also defined return message, used to refer to the receiver that has processed the message and returned a result to the message caller.
- Self Message: when a caller and a receiver are the same it means a caller sent a message to itself. is represented as a U shaped arrow.
- Create Message: the receiver of this message is created during the interaction by the message that is being sent.

- Delete Message: it destroys its receiver. Targets can be destroyed during the interaction by the message that is being sent.
- Found Message: is a message from an unknown caller to a known receiver.
- Lost Message: is a message from a known caller to an unknown receiver.

In OWL 2 DataOneOf axiom is suitable for defining a type of the message. “DataOneOf” defines a datatype with a set predefined value space.

DatatypeDefinition(type DataOneOf("synchronous" "asynchronous"…))

The message is a signal or a class operation. In a UML class diagram an operation op in class C can be one of the following:

op = { SimpleOp, ParOp }

- SimpleOp (simple operation): is an operation without parameters.
- ParOp (parameter operation): is an operation with parameters.

For each SimpleOp, we create a data property by respectively its domain and range associating with the class corresponding to the operation and the XSD type corresponding to the type of the operation in the UML class diagram.

Simple operations have a multiplicity restriction to max one value. The simple operations of a UML class in a class diagram are mapped in OWL 2 as a “DataProperty”. The range of the data property is datatype. The datatype can be xsd:string, xsd:int and other datatypes [11].

Simple operations that use basic types are mapped by declaring a data property with the operation's name. An operation is a required component of its class. Consequently, the data properties describing operations have a max cardinality of one. The Simple operation SimpleOp of the UML class C having any of the above-mentioned DataType is translated in OWL 2 as

Declaration(Class(C))

Declaration(DataProperty(SimpleOp))

SubClassOf(C DataMaxCardinality(1 SimpleOp))

DataPropertyDomain(SimpleOp C)

DataPropertyRange(SimpleOp DataType)

An operation with parameters is mapped to:

- an OWL class (named className) with data property for every additional parameter and one of its datatype, the data properties have an exact cardinality of one.
- and object property between the new class and the class contains this operation, the object property has an exact cardinality of one.

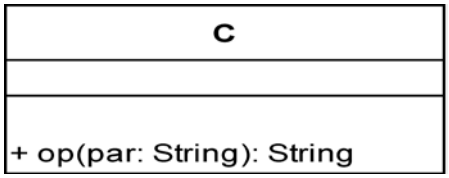

Fig. 4. A UML Operation op with Parameter par of Class C.

Fig. 4 shows an example of an operation op with parameters of class C is drawn by using UML.

The example of an operation op with parameters of class C is mapped into OWL 2 as

Declaration(Class(C))

Declaration(Class(Op))

Declaration(DataProperty(par))

SubClassOf(Op DataExactCardinality(1 par))

DataPropertyDomain(par Op)

DataPropertyRange(par String)

Declaration(DataProperty(datatype))

SubClassOf(Op DataExactCardinality(1 datatype))

DataPropertyDomain(datatype Op)

DataPropertyRange(datatype String)

Declaration (ObjectProperty(Op_C))

SubClassOf(Op ObjectExactCardinality(1 Op_C))

ObjectPropertyDomain(Op_C Op)

ObjectPropertyRange(Op_C C)

DisjointClasses(Op C)

D. Combined Fragment

A Combined fragment is an interaction fragment using an interaction operator to define the semantics of the combined fragment, such as alternative, option, and loop. Each combined fragment contains at least one interaction operand that is like a UML sequence diagram that can contain interaction fragments and messages together to model conditional behavior in a UML sequence diagram. An interaction operand illustrates the interactions between classes or object instances and the ordering of messages according to time. An interaction operand may have interaction constraints also called guards, which is a boolean conditional expression. A guard is a semantic condition that specifies the condition under which the interaction fragments and messages will be performed inside the interaction operand.

Interaction Operators as defined in the UML sequence diagram [12] are listed below.

- sd: abbreviation for sequence diagram, has one operand used for framing an entire sequence diagram.
- alt: abbreviation for alternatives, means that the combined fragment represents alternative or choice

paths of execution. Only the one whose guard is true will be chosen to execute.

- opt: abbreviation for option, equivalent to an alt only with one path with a guard, and the option operand is executed if the guard is true.
- loop: has one operand with a guard, means that the combined fragment represents a loop. The operand will be repeated at least the minimum count and no more than the maximum count as long as the guard is true.
- break: has one operand with or without a guard that is performed instead of the remainder of the enclosing interaction fragment.
- par: abbreviation for alternatives, means that the combined fragment represents more than one operands which can be executed parallel. Operands can be freely interleaved. In any order, but must be according to the ordering imposed by each operand separately.
- critical: abbreviation for critical region, is a region cannot be interleaved by other occurrence specifications.
- ref: abbreviation for critical reference, refers to an interaction defined on another diagram.

There are other interaction operators, such as Strict Sequencing, Weak Sequencing, Negative, Ignore, Consider, and Assertion which are also defined in [12, 13].

In OWL 2 DataOneOf axiom is suitable for defining a type of a combined fragment. “DataOneOf” defines a datatype with a fixed predefined value space.

DatatypeDefinition(kind DataOneOf("alt" "opt"…))

We can use a number of value restriction infix operators with the guard constraint of the class “Operand”, such as =, >=, <=, > and <. The guard constraint is mapped in OWL 2 as a DataProperty. The value constraint of the guard is written in the UML sequence diagram as [Guard Op Value], where OP is the infix operator and Value represents the guard value. The map of the value constraint in OWL 2 is based on the infix operator used with the guard operator, such as “>=” is mapped using the OWL 2 axiom DatatypeRestriction and xsd:minInclusive, “>” is mapped using the OWL 2 axiom DatatypeRestriction and xsd:minExclusive,“<=” is mapped using the OWL 2 axiom DatatypeRestriction and xsd:maxInclusive, “<” is mapped using the OWL 2 axiom DatatypeRestriction and xsd:maxExclusive, “!=” is mapped using the OWL 2 axiom complementOf and the axiom DataHasValue, and “=” mapped using the OWL 2 axiom DataHasValue. For example, [Guard = Value] is a guard constraint, is mapped to OWL 2 as:

DataHasValue(Guard }Value}^^ datatype)

In this translation, Guard is the name of the guard, Value is the value of the guard, and datatype is the datatype of the guard Value.

IV. Case Study

There has been substantially related work on mapping UML diagrams into Ontology has been discussed by several authors in the past. For instance, B. Bouchra. Author in [2] discusses the conversion method by building an e-learning ontology from its UML class diagram. In their approach. They have recourse to the Collection Ontology to map the composition relationship and the Value Partitions Design Pattern to map the inheritance. Moreover, the approach presented in [4] discusses the migrating UML class diagrams to Ontology. In their approach, the model information is stored in the XMI document by using a Power Designer tool then creating an ontology by passing this XMI document as the input of their mapping algorithms. The approach presented in [8] describes conversion rules from UML diagrams to Ontology containing multiple class, object and statechart diagrams. However, the goal of his work is analyzing the consistency and satisfiability of models. Moreover, they do not discuss the mapping UML sequence diagrams into Ontology.

We present a summary of our technique that we expound with a running example. Fig. 5 shows a UML sequence diagram that describes the withdrawal cash scenario of an ATM system, where Messages are numbered top-down. It exposes the object of each class and messages that can be invoked on them. It consists of four classes, namely, “User”, “ATM”, “Bank” and “Account”. To evaluate our technique we need to first map this diagram into OWL 2 by following the mapping discussed in the previous sections, after mapping the diagram we pass the OWL 2 ontology to the OWL 2 reasoner. Fig. 6 and 7 illustrate the diagram after is mapped.

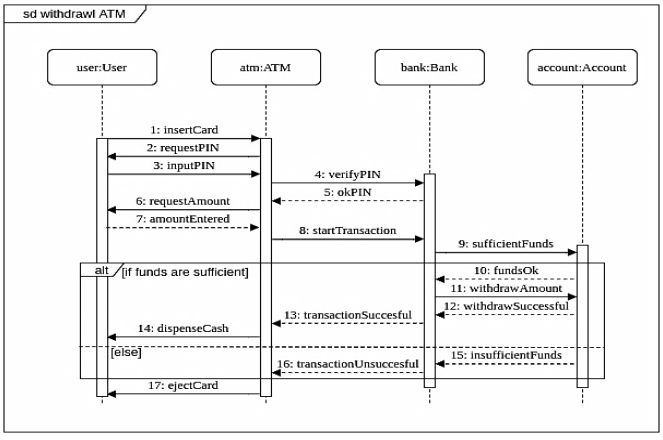

Fig. 5. Withdrawal Cash Scenario of an ATM System.

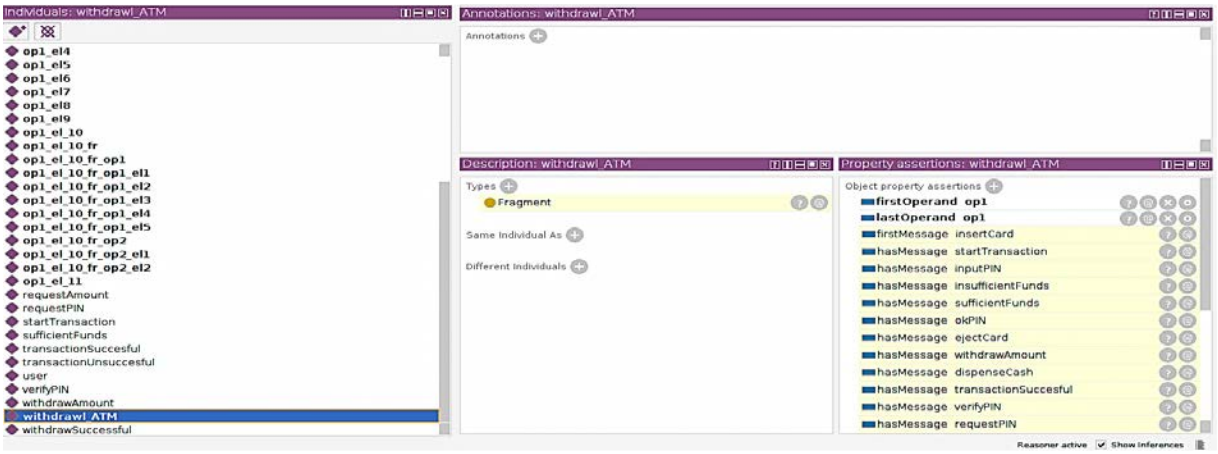

Fig. 6. The Withdrawal ATM Diagram after is Mapped.

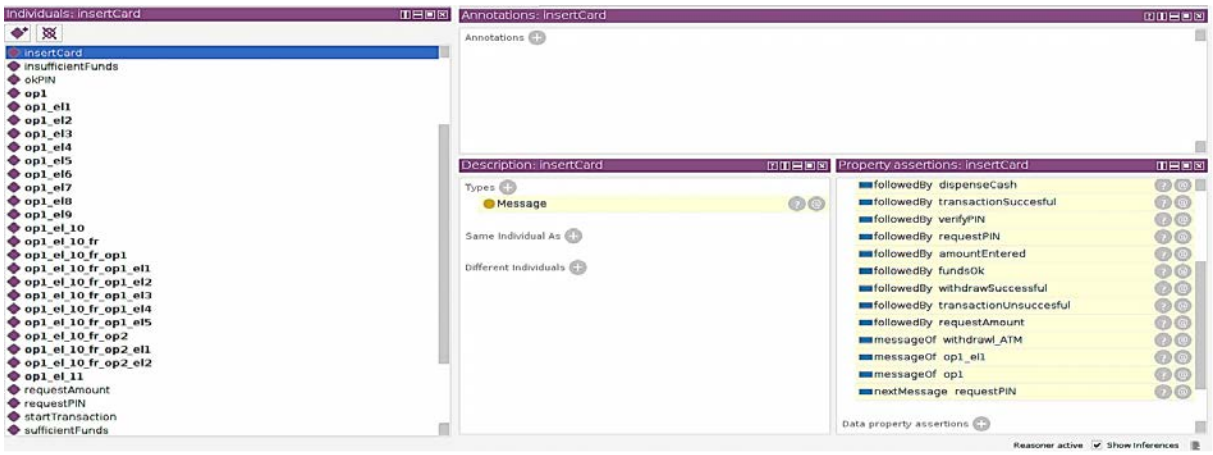

Fig. 7. The First Message insertCard after is Mapped.

V. Conclusion

UML sequence diagrams are used to describe the behavior of systems. In this paper, we have demonstrated a technique for mapping behavioral knowledge expressed in the UML sequence diagrams as an OWL ontology. The OWL DL and SWRL rules are used to formalize the semantics of the sequence diagrams model. We have analyzed similarities and differences among UML and OWL elements in-depth. With this knowledge, we have developed rules for addressing the issue of defining sequence in OWL. Furthermore, we formalized a suitable way to handle the fragment operator with an arbitrary number of operands, which is crucial when specifying transformations of sequence diagrams.

As our future work, we will continue to study additional cases in order to complete the set of rules. We plan to provide the support of other interaction constraints.

References

[1] P. Kumar K N, R. Kumar V, K. Raghuveer, “A Survey on Semantic Web Technologies for the Internet of Things,” In 2017 IEEE International Conference on Current Trends in Computer, Electrical, Electronics and Communication (CTCEEC), 2017, doi: 10.1109/CTCEEC.2017.8454974.

[2] B. Bouchra, B. Mohamed, “Building an e-learning system’s OWL ontology by exploring the UML model”, Journal of Theoretical and Applied Information Technology (JATIT),Vol.87,No.3,2016,pp.380-387.

[3] H. Martina, V. Bureš , "Formal Ontologies in Information Systems Development: A Systematic Review", Information, Vol. 11, No. 2, 2020.

[4] E. Oussama, A. Khadija, A. Larbi, B. Mohamed, "Mapping UML to OWL2 Ontology", Journal of Theoretical and Applied Information Technology (JATIT), Vol. 90, No. 1, 2016, pp. 126-143.

[5] OMG “Ontology Definition Metamodel Superstructure Version 1.1,” 2014, https://www.omg.org/spec/ODM/About-ODM/.

[6] K. John, “Model-Based Development and Evolution of Information Systems – A Quality Approach” , Springer, London, UK, 2012.

[7] Object Management Group Website (OMG), https://www.omg.org/.

[8] H.K. Ali, P. Ivan, “Consistency of UML class, object and statechart diagrams using ontology reasoners”, Journal of Visual Languages & Computing, Vol. 26, 2015, pp. 42-65.

[9] S. Evern, P. Bijan, C.G. Bernardo, K. Aditya, K.Yarden, “Pellet: a practical OWL-DL reasoner” , Journal of Web Semantics, Vol. 5, No. 2, 2007, pp.51-53.

[10] C. Paolo, P. Silvio, “The Collections Ontology: creating and handling collections in OWL 2 DL frameworks”, Semantic Web Journal (SWJ), Vol. 5, No. 6, 2014, pp. 515- 529.

[11] W3C, OWL Working Group, “OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax. W3C Recommend ation 11 December 2012,” https://www.w3.org/TR/owl2-syntax/.

[12] OMG, “Unified Modeling Language Superstructure Version 2.5.1,” 2017, https://www.omg.org/spec/UML/About-UML/.

[13] R. James, J. Ivar, and B. Grady, “The Unified Modeling Language Reference Manual 2nd Edition. Addison Wesley”, 2004.