TripCom

Triple Space Communication

FP6 – 027324

Deliverable

D7.2

Ontology of EDIFACT Syntax and Semantics

doug foxvog [NUIG]

April 1, 2008

Executive Summary

The aim of the Ontological Infrastructure work package (WP 7) is to offer a means for a semantically rich definition of business processes for the express purposes of overcoming heterogeneity problems. The concrete goal of Work Package 7 is to ontologize a significant portion of the current EDIFACT EDI standard as a basis for business-to-business process integration. The main objective of this document is to present the developed sets of ontologies for major EDIFACT subsets. After a review of the selection process for the subsets (Chapter 2), we present the created ontologies for the syntax (Chapter 3) and semantics (Chapter 4) of the selected subsets. Chapter 5 presents the lessons learned from the ontologization process. Chapter 6 explores possible future uses of the developed ontologies and suggests fruitful avenues for furthering this work.

Document Information

IST Project Number	FP6 – 027324	**Acronym**	TripCom
Full Title	Triple Space Communication		
Project URL	http://www.tripcom.org/		
Document URL			
EU Project Officer	Werner Janusch		

Deliverable	**Number**	7.2	**Title**	Ontology of EDIFACT Syntax and Semantics
Work Package	**Number**	7	**Title**	Ontological Infrastructure for Business Processes and Data

Date of Delivery	**Contractual**	M24	**Actual**	15-3-08
Status	version 0.1		final ☐	
Nature	prototype ☐ report ☒ dissemination ☐			
Dissemination Level	public ☒ consortium ☐			

Authors (Partner)	doug foxvog (NUIG)			
Resp. Author	doug foxvog (NUIG)		**E-mail**	doug.foxvog@deri.org
	Partner	NUIG	**Phone**	+353 (91) 495 150

Abstract (for dissemination)	The aim of the Ontological Infrastructure work package is to offer a means for a semantically rich definition of business processes for the express purposes of overcoming heterogeneity problems. The concrete goal of the Work Package is to ontologize a significant portion of the current EDIFACT EDI standard as a basis for business-to-business process integration. The main objective of this document is to present the developed sets of ontologies for major EDIFACT subsets. After a review of the selection process for the subsets, we present the ontologies and knowledge bases created for the syntax and semantics of the selected subsets. The lessons learned from the ontologization task are analyzed and fruitful avenues for furthering this work are suggested.
Keywords	Ontology, EDI, EDIFACT, TripCom, business standards, triple space, triples

Version Log			
Issue Date	**Rev No.**	**Author**	**Change**
2007-01-15	1	doug foxvog	Initial version created
2007-03-15	2	doug foxvog	Initial version completed for QA

Project Consortium Information

Partner	Acronym	Contact
Leopold Franzens University Innsbruck http://www.sti-innsbruck.at	LFUI	Prof. Dr. Dieter Fensel Semantic Technology Institute (STI) Innsbruck, Austria E-mail: dieter.fensel@sti-innsbruck.at
National University of Ireland, Galway http://www.deri.ie	NUIG	Dr. Laurentiu Vasiliu Digital Enterprise Research Institute (DERI) Galway, Ireland Email: laurentiu.vasiliu@deri.org
University of Stuttgart http://www.iaas.uni-stuttgart.de/	USTUTT	Prof.Dr. Frank Leymann Inst. für Architektur von Anwendungssystemen (IAAS) Stuttgart, Germany E-mail: frank.leymann@informatik.uni-stuttgart.de
Vienna University of Technology http://www.complang.tuwien.ac.at/	TUW	Prof.Dr. eva Kühn Institut für Computersprachen Vienna, Austria E-mail: eva@complang.tuwien.ac.at
Free University Berlin http://www.ag-nbi.de/	FUB	Prof. Dr.-Ing. Robert Tolksdorf AG Netzbasierte Informationssysteme Berlin, Germany E-mail : tolk@inf.fu-berlin.de
Ontotext Lab, Sirma Group Corp. http://www.ontotext.com/	ONTO	Atanas Kiryakov, Vassil Momtchev, Ontotext Lab, Sirma Group Corp. Sofia, Bulgaria E-mail: vassil.momtchev@ontotext.com
Profium OY http://www.profium.com/	Profium	Dr. Janne Saarela Profium OY Espoo, Finnland E-mail: janne.saarela@profium.com
CEFRIEL SCRL. http://www.cefriel.it/	CEFRIEL	Davide Cerri CEFRIEL SCRL. Milano, Italy E-mail: cerri@cefriel.it
Telefonica I+D http://www.tid.es/	TID	Noelia Pérez Crespo Telefonica I+D Madrid, España E-mail: tripcom@tid.es

Table of Contents

1 Introduction

The aim of the Ontological Infrastructure work package (WP 7) is to offer a means for a semantically rich definition of business processes for the express purposes of overcoming heterogeneity problems. The concrete goal of Work Package 7 is to ontologize (use ontologies to encode) a significant portion of the current EDIFACT Electronic Data Interchange (EDI) standard as a basis for business-to-business process integration. The main objective of this document is to present the developed sets of ontologies for major EDIFACT subsets and the process of creating the ontologies.

1.1 Overview

The EDIFACT system, as described in Deliverable D7.1 [5], is the major inter-business electronic data exchange system in Europe. The system defines a fixed set of message types and specifies a format for messages of each type. These formats define the message structure in terms of

- the order, optionality, and limitations on repetitions of data segments and loops of data segments,
- the structure of data segments in terms of composite and simple data elements,
- the structure of the composite data elements in terms of simple data elements, and
- the formats of the simple data elements (text string, integer, decimal value, or coded value).

Permissible codes and their meanings are also specified in the descriptions. Restrictions on the presence or absence of one component (or component value) based on the presence or absence of another component (or component value) are also part of the syntactical description.

In addition to the syntax description, the meanings (semantics) of the components are described – both in themselves and as related to each other. Sometimes the meaning of a component expresses how two other components are related. For example, the meaning of one component may be a warehouse, while the meaning of a second component is whether the warehouse should be the location which a product being shipped should be shipped from, shipped to, or where it should be switched from one form of transit, e.g. truck, to another, e.g. ship.

The process of ontologizing EDIFACT messages covers using ontologies to encode both their syntax and semantics, while including any inter-component restrictions on either format or meaning. This ontologization process was discussed in detail in the original TripCom work plan (Annex I) [1] and in the Detailed Work Plan (Deliverable 10.1) [4].

For the purposes of this document, an *ontology* is a formal definition of terminology and relationships among the terms in a computer-processable form, while a *knowledge base* stores information about a particular situation, through encoding that information using one or more ontologies. Thus, one set of ontologies contain the terms and relations for describing the syntax and semantics of EDI messages and a second set of

ontologies was created to handle the topics referred to by the messages, but the encodings of the syntax and semantics of the message types and their parts were stored in knowledge bases.

1.2 Applicability of EDIFACT to TripCom

Although the subsets of EDIFACT to be ontologized were chosen to be useful for the use cases and the use of EDIFACT messages pointed out by WP 7 was considered by each of the use cases, eventually neither use case decided to use the messages, nor to have their messages anywhere near the complexity of the EDIFACT messages.

Work Package 8B researchers examined the ontologies and knowledge bases developed for the MEDRPT (Medical Service Report), MEDREQ (Medical Service Request), and MEDPRE (Medical Prescription) messages while designing their scenario and the messages which they would include. However, they decided to align with different existing standards: the Continuity of Care Record (CCR [8]) and the Clinical Document Architecture (CDA [6]).

Work Package 7 discussed EDIFACT messages with Work Package 8A researchers, including the ontologies and knowledge bases under development. Work Package 8A used a number of the concepts from the messages, but decided not to align its communications with the EDIFACT messages, nor to use the ontologies being developed by WP 7.

1.3 Ontology Tools

DERI Galway started to ontologize a different EDI system (ANSI X12) in 2004. They designed ontologies for expressing both the format (syntax) of EDI messages [2] and templates for the meaning encoded in messages [3]. Separate – but interrelated – ontologies were created for computer data structures in general, EDI message formats in particular, and for templates expressing how semantics is encoded in EDI messages.

These ontologies are general enough to be used for specifying EDIFACT formats as well as X12 formats and were selected for use in specifying the formats of our selected subsets of EDIFACT and the templates for their meanings in knowledge bases (KBs). During the process of encoding the messages, constraints were discovered which were not covered by the system devised for X12, and the tool ontology was enlarged so that these constraints could be expressed. This is reported in more depth below.

Separate knowledge bases created for describing individual ANSI X12 messages and their components were not applicable for ontologizing EDIFACT and so were not used.

1.4 Overview of the Rest of the Deliverable

Chapter 2 reviews the selection process for the subsets of EDIFACT we chose to ontologize. Chapter 3 presents the EDIFACT syntax, Chapter 4 presents the knowledge bases created for encoding the syntax of the selected subsets, while Chapter 5 presents the process of encoding the semantics. Chapter 5 presents the lessons learned from the ontologization process. Chapter 6 explores possible future uses of the developed ontologies and suggests fruitful avenues for furthering this work.

2 EDIFACT Subset Selection

Due to the vast number of message types and data segments defined in EDIFACT and the generality of such components to serve the diverse needs of every industry, most industries and organizations using the system have developed specialized standards based on EDIFACT, focused in their business domain. The resulting definitions of subsets or sub-standards of EDIFACT are typically called Implementation Guidelines.

Each Implementation Guideline limits the message types it includes, the formats of the included message types, and the permitted values for codes for coded data elements. They often specify additional codes for data elements which are not part of the generic EDIFACT standard.

Such substandards are often only suitable for narrow applications in contrast with the the original idea of a common standard for all electronic data interchange.

2.1 Range of EDIFACT Subsets

As discussed in Deliverable D7.1[5], *Analysis of EDIFACT and Other Standards*, there is a wide range of industry-specific EDIFACT subsets. Industries which have defined their own subsets include (among others) the chemical, accounting, the electronics and computer, library supply, appliances, medical, telecommunications, goods supply and demand, and travel bookings industries.

2.2 Criteria for Selecting EDIFACT Subsets to Ontologize

The EDIFACT standard is so large; the number of subsets is so great; and the need for EDI messages is so limited for TripCom use cases, that we chose to ontologize selected subsets of the standard as discussed in Deliverable D7.1.

The EDIFACT subsets were selected on several grounds:

- Overlap with other subsets – A type of message used in several industries was deemed to be likely to be more useful than one used in a single industry.
- Utility of message types for TripCom use case in WP 8A (Enterprise Integration)
- Utility of message types for TripCom use case in WP 8B (European Patient Summary)
- Use of subset by Partners' industries (IT, Telecommunications)
- Being actual subsets of EDIFACT, not merely using EDIFACT format – Message types of interest in non-subset variants are likely to have similar message types in standard EDIFACT and its subsets.

This resulted in our decision to build ontologies for the EANCOM (goods supply and demand), EDIFICAS (Accounting), EDIFICE (Electronic goods), EMEDI (Medical), and ETIS (Telecommunication) standards.

Each substandard is based on a fixed version of EDIFACT, thus they are unaffected by the semi-annual updating of the main standard.

3 EDIFACT Syntax

3.1 Structure of an EDIFACT message

Each EDIFACT message is composed of three sections (see Figure 3.1): header, detail and summary. Each section is made up of an ordered arrangement of data segments and loops of segments (segment groups), with restrictions on repetitions of each component. Each *segment group* is made up of an ordered arrangement of data segments and, frequently, further nested segment groups. Each *data segment* is made up of an ordered arrangement of data elements which may be composite. Each *composite data element* is made up of an ordered arrangement of non-composite data elements. Each non-composite *data element* has a maximum length and a format of text string, integer, decimal number, or occasionally binary value (e.g., for encrypted data). The text strings may be codes from specified groups of codes called code sets.

The Implementation Guidelines specify the maximum number of repeats for each component.

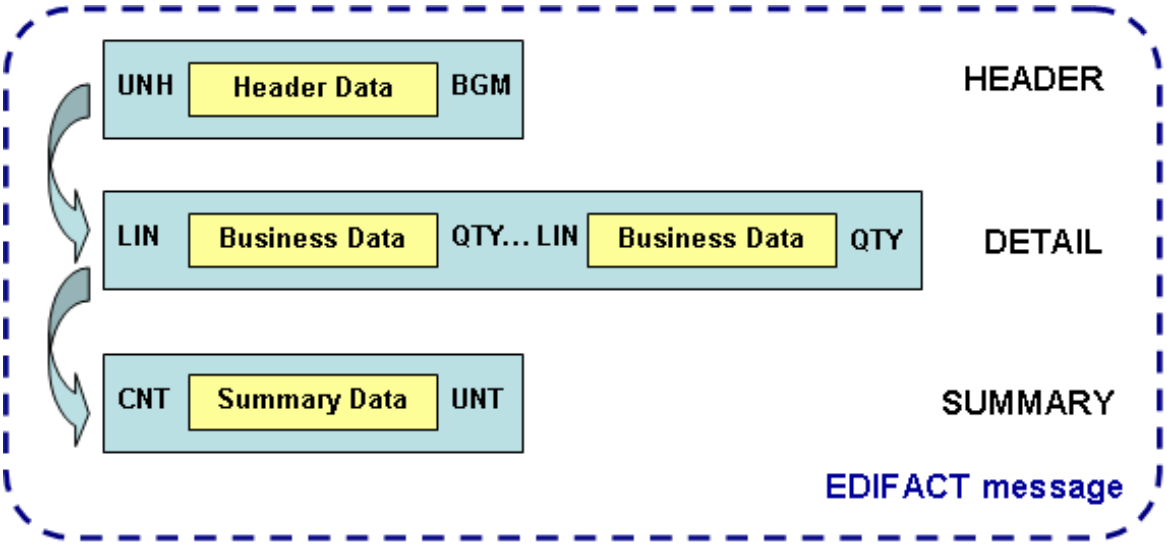

Figure 3.1: EDIFACT Message Structure.

The header section is a group of segments containing information which relates to the entire message. The complete header section can not repeated, though it might contain segments or segment groups which may be individually repeated.

The detail section is comprised of one or more groups of segments carrying information which relates to repeating elements in the message, such as order lines or invoice lines. By analogy with paper documents, in many of the simpler standard transaction messages each repeat of the detail section is referred to as a line. In more complex messages the structure of the detail section may involve nested repeats at several levels.

The summary section is a group of segments containing totals or control information, e.g. invoice total amount or number of lines in a purchase order. Like the header section, it occurs only once per message.

The same segment type may occur in more than one of the message sections, e.g. in the header and in the detail section, and/or more than once in the same section.

Some components are mandatory at a given position in a message, while others are optional. Some may be repeated a certain number of times at their specific location in the message. The status (mandatory or conditional) and the maximum number of repetitions of components are defined for each subset.

Each version of the EDIFACT standard specifies these formats for each message type using a message specification document. The specification document provides a branching diagram for the message and segment tables which define the structure of each segment.

Within a message, specific groups of functionally related segments may be repeated; these groups are referred to as segment groups. The maximum number of repetitions of a particular segment group at a specific location is indicated in the message specification.

A segment group may be nested within other segment groups, provided that the inner segment group terminates before any outer segment group terminates.

3.2 Structure of an EDIFACT Data Segment

An EDIFACT data segment consists of a sequence of data elements. Upon being instantiated in an instance of a message, additional formatting symbols are necessary to indicate boundaries between components. Hence, an instantiated data segment consists of:

- A three-letter segment tag, which identifies the segment type and is mandatory in every segment.
- Composite data element separators, which are optional.
- Nesting and repetition indicators, which are optional.
- Data element separators, which are mandatory.
- Simple or composite data elements, which are optional or mandatory as specified in a standard.
- A segment terminator, which is mandatory.

Data elements may be either simple or composite. Composite data elements contain two or more simple (non-composite) data elements, called component data elements when used inside a composite data element. Most data elements are components of only a single type of data segment.

Simple data elements can be defined as having fixed or variable (with some maximum) length. Data elements may be filled by coded values, numeric values, text strings, or occasionally (e.g., with encrypted data) binary data.

Together, these comprise the different EDIFACT structural layers (see Figure 3.2).

A data element can be qualified by another data element, the value of which is expressed as a code that gives specific meaning to the data. The data value of a qualifier is a code taken from an agreed set of code values.

Here is a simple EDIFACT segment example: `DTM+137:19940121:102'`

- `DTM` = segment tag, identifying the "Date/time/period" segment

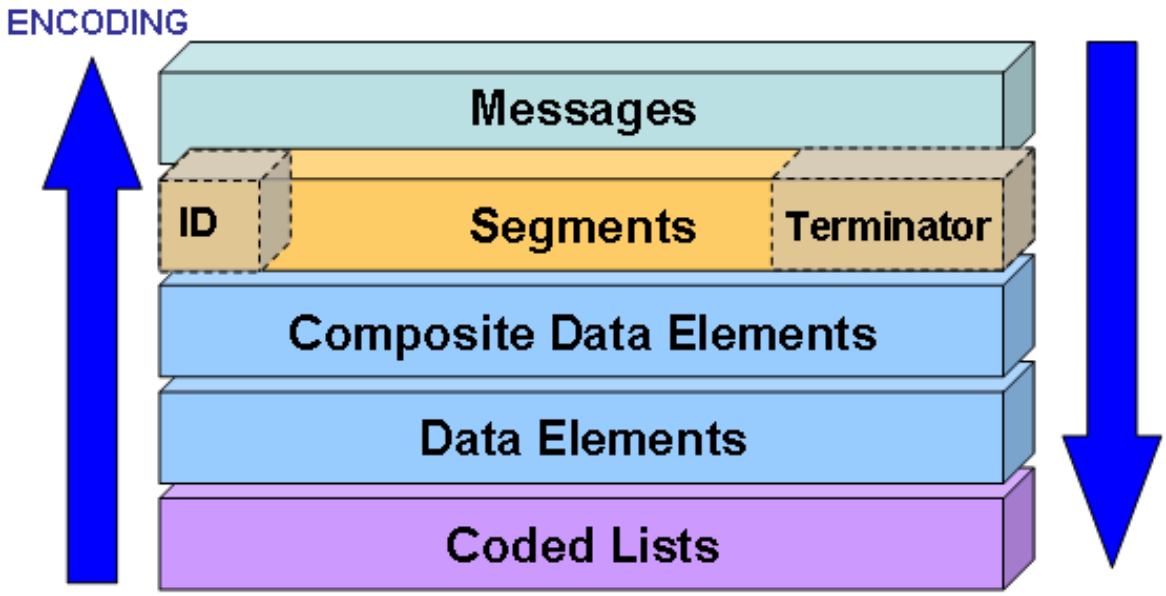

Figure 3.2: EDIFACT message layers.

- `+` = segment tag and data element separator
- `137` = date qualifier, code indicating that the date is the document/message date/time
- `:` = separator for component data elements within a composite (here, date qualifier and date)
- `19940121` = date, in the format specified by the date format qualifier
- `:` = separator for data elements within a composite (here, date and date format qualifier)
- `102` = date format qualifier, code indicating the format of the date (here, CCYYMMDD)
- `'` = segment terminator

3.3 Format Restrictions

Limitations on one part of a data structure due to features of another part of the data structure are deemed format restrictions. Examples are two subcomponents that must not appear as part of the same structure and a subcomponent that must appear if another subcomponent has a specified value. Restrictions on the presence or possible values of a component of a data segment due to the location of that segment in a message are common. Restrictions on values are normally restrictions on the codes from a code set that can be used in certain circumstances. A couple dozen types of format restrictions were defined in the previous work[2], but the need for several more appeared during the task of encoding the restrictions on the formats of the EDIFACT subsets.

The list of types of format restrictions, including the new ones *in italics*, appears below.

- Restrictions on the presence or absence of components
  - Forbidden – no instance of the indicated subcomponent may be present in an instance of the first structure.
  - Required – at least one of the indicated components is required in the structure being restricted.
  - Conditional – if the first of the indicated components is present in the structure being restricted, then the second one is required
  - List Conditional – if the first of the indicated components is present in the structure being restricted, then at least one of the others must be present.
  - Exclusive Use – only one of the indicated components may be present in the structure being restricted.
  - Loop Consistent – if any instance of the specified subcomponent exists for a structure, every instance of the structure will also have a value for the same subcomponent.
  - Max Subcomponents – there can be at most the indicated number of instances of the specified subcomponent.
  - Paired – if any of the indicated components is present in the structure being restricted, they all must be.
  - *Multiplicity Requires* – if more than one instance of the first indicated data element is present in the structure, the second indicated component must be present in each instance of its supercomponent which is present in the same structure.
- Restrictions on the presence of one component due to the specific value of another
  - Value Forbids – if the value of the first indicated data element is equal to one of the specified values, the second indicated component may not be present in the same structure.
  - Value Requires – if value of the first indicated data element is equal to one of the specified values the second indicated component must be present in the same structure.
  - *Values Require* – if values of the first two indicated data elements are among those listed for each component, the third indicated component must also be present in the same structure.
- Restrictions on the comparative values of one component relative to another
  - *Different Values* – no value of the first indicated component is equal to that of any instance of the second indicated component.
  - Equal Values – the value of the first indicated component is equal to that of the second indicated component.
  - Equals Sum – the value of the only instance of the first indicated data element is equal to the sum of all instances of the second indicated data element.

  - Exists Equal – if any instance of the first data element exists in an instance of STRUCTURE, an instance of the second data element with the same value will also exist in an instance of the same structure in the same message.
  - *Greater Value* – the value of any instance of the first indicated component is numerically greater than that of any instance of the second indicated component (at least one of which must exist). [Reverse the order of the indicated components to specify a lesser value.]
  - Sequential – values of instances of the specified component within the message have sequential integer values starting with the value "1". This is a more specific form of Unique.
  - Unique – every instance of the specified component within the message must have a unique (i.e. different) value. If more than one instance of the enclosing component is present, this means that each must have a value for the specified subcomponent – which, of course, must be unique.

- Restrictions on the specific value of a single component
  - Some Value – some instance of the indicated subcomponent must have one of the specified values for some instance of the larger structure. If only one instance of the subcomponent is permitted, this is the same as Value List.
  - Value – every instance of the indicated structure must have the specified value.
  - Value List – every instance of the indicated component must have one of the specified values.

- Restrictions on the specific value of one component due to the presence of another
  - Number Of – the value of the first indicated component is equal to the number of instances of the second indicated component within the same message.
  - Value When Absent – when the first indicated component is absent from the specified structure, the second indicated subcomponent must be present and all instances must have one of the specified values.
  - Value When Present – when the first indicated component is present in the specified structure, the second indicated subcomponent must be present and all instances must have one of the specified values.

- Restrictions on the specific value of one component due to the specific value of another
  - *Exclusive Values* – only one of the specified component's subcomponents may have a value from the value list. Values other than those on the list are not restricted by this condition. Subcomponents having values which are not in the list are not restricted. [See Unique Subcomponent Values]
  - Order – instances of the indicated subcomponent whose values are contained in the specified list should appear in with their values in the order in that list. Multiple data elements may share the same value (unless otherwise restricted); not every value is required (unless otherwise restricted); and values outside the list are permissible (unless otherwise restricted).

- *Paired Values* – if the sole value of the first indicated data element is equal to one of the specified values, the sole instance of the second indicated data element within the same structure must have one of the values in a second specified set. This is equivalent to a pair of Value Requires Only Value restrictions.
- *Unique Subcomponent Values* – only one of the specified component's subcomponents may have any specific value from the value list. Each value on the list is available only one time to a subcomponent of the specified component. Values other than those on the list are not restricted by this condition. Subcomponents having values which are not in the list are not restricted. [See Exclusive Values.]
- *Value Absence Requires Value* – if value of the first indicated data element is not equal to one of the specified values, the second indicated component must be in the second specified set of values.
- *Value Requires Only Value* – if the value of the first indicated data element is equal to one of the specified values, any instance of the second indicated data element within the same structure must have in a second specified set.
- Value Requires Value – if value of the first indicated data element is equal to one of the specified values, an instance of the second indicated data element within the same structure (or another instance of the same data element if a second one is not specified) must have one of the values in a second specified set.
- *Values Require Value* – if the values of the first two indicated data elements are equal to (one of the values in) value1 and value2 respectively, an instance of the third indicated data element within the same structure must have a value specified in a third set.

4 EDIFACT Syntax Knowledge Bases

In order to define the syntax (structure/format) of EDIFACT messages using a syntax ontology, the ontology has to be developed, a description of the syntax must be obtained, and that syntax needs to be encoded in a knowledge base using the syntax ontology.

4.1 Description of Syntax

The ontology designed by DERI Galway to encode EDI system formats uses the concepts `DataStructure`, `MessageType`, `DataSegmentLoop`, `DataSegment`, `DataElement`, and `ComplexDataElement`[1] to define the structures discussed above and auxiliary structures `MultiDataElement` and `MultiDataSegment` for ease in defining attributes. Formats are described using `Format`, `FormatCode`, `FormatDescriptor`, and `Optionality`. Format restrictions use the concepts `FormatRestriction` and `FormatRestrictionType`, while actual instances of messages and their parts are specified using `BusinessMessage`, `EDIMessagePart`, `DataSegmentLoopInstance`, `DataSegmentInstance` and `DataElementInstance`.

Attributes of these concepts are used to define the syntax of their instances [2].

For example, the format of the FCA (financial charges allocation) data segment is defined in WSML as follows:

```
instance ds#FCA_DS memberOf edi#DataSegment
  edi#hasFormat hasValue FCA_DS_Format
edi#hasFormatDescriptor hasValue FCA_DS_FD

instance FCA_DS_Format memberOf edi#Format
  nonFunctionalProperties
    dc#description hasValue "Format for EDIFACT FCA data segment"
  endNonFunctionalProperties
  edi#formatFor1stComponent hasValue fc#FC_M1
  edi#formatFor2ndComponent hasValue fc#FC_O1

instance FCA_DS_FD memberOf edi#FormatDescriptor
  nonFunctionalProperties
    dc#description hasValue "FormatDescriptor for FCA data segment"
  endNonFunctionalProperties
  edi#formats1stComponent hasValue de#DE_4471
  edi#formats2ndComponent hasValue cde#C878_CDE
```

This means that the format of the segment is a mandatory data element – data element 4471 (Settlement means code) – followed by an optional composite data element – C878 (Charge/allowance account). Only one repetition of each is permitted.

[1] for composite data elements

4.2 Development Process

Web documents were obtained for each of the selected subsets of EDIFACT, as well as the full standard. The full standard[2] is published on the web, from which any version from the second version of 1998 (D.98B) to the present (two per year) is available. The EANCOM standard[3] (based on UN/EDIFACT D.01B Syntax 4) is also available for web browsing. The EDIFICE[4], ETIS[5], and EDIFICAS [6] implementation guidelines were downloaded in files from the respective standard organizations' websites. EMEDI's medical messages are not used by any other industry, so were taken directly from the appropriate messages in the EDIFICE standard.

A set of knowledge bases using the EDI syntax-defining ontology was set up: a knowledge base for the set of data segments and their subcomponents at the general EDIFACT level and individual knowledge bases for the messages and more specific knowledge bases for the data segments and data elements for each of the selected subsets. The knowledge bases at the subset level applied more restrictive formats on the components defined at the EDIFACT level. For example, a component deemed optional at the EDIFACT level, might be required in one subset and forbidden in another.

4.3 Describing EDIFACT Subsets in Syntax Knowledge Bases

The descriptions of the messages for the EDIFACT standard and each of its subsets have a standard form: a message structure chart[7] describing the segments and segments loops in the message including their nesting, maximum repetitions, and optionality; a textual description of the components; and a segments layout chart[8] for each segment, showing the order, repetitions, and optionality of data elements in each segment, including the structure of any composite data elements.

The subset descriptions provide the standard EDIFACT definitions, along with their own restrictions of the format. Such restrictions may require or forbid the presence of specific subcomponents that the generic EDIFACT standard declares optional or reduce the permitted number of repetitions of a subcomponent.

The EDIFACT subsets normally provide a second basic chart for each message, called a branching diagram[9], which graphically depicts the structure in the message structure chart. The branching diagram, unlike the message structure chart, provides only the code, not the name, of the data segment and omits textual descriptions and explanations.

[2] http://www.unece.org/trade/untdid/directories.htm
[3] http://www.gs1.se/EANCOM$_2$002/*ean*02*s*4/*experts*/*part*2/*part*2*toc.htm*
[4] http://repository.edifice.org/migs/index.htm
[5] http://www.etis.org/activities/ebg.asp
[6] http://www.edificas.org/edificas_ftp_x-tdfc.htm
[7] e.g. http://www.ean.se/EANCOM_2002/ean02s4/experts/part2/invoic/041.htm
[8] e.g. http://www.ean.se/EANCOM_2002/ean02s4/experts/part2/invoic/059.htm
[9] e.g., http://repository.edifice.org/migs/invoic/bd1.htm

4.3.1 Knowledge Base by Structural Level

The same data segment types are used in many different message types. Some data element types are used in multiple data segment types, although most appear in a single type of data segments.

For these reasons, the message types and data segments have their syntax ontologized separately, while the data elements are normally ontologized with their associated data segments.

4.3.2 Knowledge Base by Standard EDIFACT Subset

The different EDIFACT subsets vary enough in the segments and segment loops present or absent in a message that it was not deemed useful to describe a message format at the EDIFACT level and then modify it for each subset. Thus, although the message types are defined at the EDIFACT level, the format of each message is defined – in terms of data segments and segment loops – at the subset level.

In most cases, a separate knowledge base is used to store the encoded format for each data segment and its data segment loops. However, some of the smallest loops (of one, two, or three data elements) occur in a number of different message types in a given EDIFACT subset, so the formats for these standard data segment loops are ontologized in a separate KB for the associated subset.

The formats of the data segments and composite data elements, however, are not nearly so complex as those of the messages, and therefore have been defined at the EDIFACT level, with the formats more narrowly specified at the subset level as needed.

At the EDIFACT level, components of the message, data segment loops, data segments, and composite data elements are either mandatory or optional; at the subset level, components deemed optional by EDIFACT may be left optional or made "required" or "forbidden" for that subset. Another subset might make the opposite assignment. EDIFACT may specify a maximum number of repetitions of a component that is more than is needed by the industry of a given subset.

4.3.3 Knowledge Base by Related Message type

In several cases, two message types have almost the same structures, with only slight variations. One example is the purchase order (ORDERS) and blanket purchase order (BORDERS) pair of messages. In such cases, the formats of the two message types are defined in the same knowledge base, allowing the easy reuse of shared subcomponent loops.

4.4 Created Ontologies and Knowledge Bases

The EDI ontologies were used to define the 66 message types present in at least one of the subsets we have selected. Separate knowledge bases were created to define the message types, data segments, composite data elements, and simple data elements. One knowledge base was created at the EDIFACT level to encode the general formats for the 91 data segment types and 116 composite data element types present in these messages.

Restriction Type	# Uses
Absence Requires	3
Conditional	86
Different Values	2
Equal Values	6
Exclusive Use	80
Exclusive Values	2
Forbidden	16
Greater Value	26
Multiplicity Requires	1
Number Of	2
Paired	12
Paired Values	1
Required	158
Requires Values	24
Sequential	9
Some Value	23
Unique	2
Unique Subcomponent Values	0
Value	791
Value Absence Requires	6
Value Absence Requires Value	14
Value Forbids	41
Value Forbids Value	10
Value List	1056
Value Requires	49
Value Requires Absence	1
Value Requires Only Value	1
Value Requires Value	56
Value When Absent	10
Values Require	3
Values Require Value	3

Table 4.1: Format Restriction Usage

Subset-specific knowledge bases were created in for EANCOM, EDIFICAS, EDIFICE, EMEDI, and ETIS to specify more specific formats for the data segments and composite data elements present in the messages of each subset. Knowledge bases were created for encoding data formats for the message types used by each subset at the subset level. These knowledge bases reference (logically include) the similar knowledge base at the EDIFACT level.

44 knowledge bases were created to specify the formats of the 52 EANCOM message types. EDIFICE had 16 KBs created for 26 message types; The other EDIFACT subsets, EDIFICAS, EMEDI, and ETIS, respectively had six, seven, and two KBs created – each for a single message type.

The syntax knowledge bases at the subset levels encoded not just the sequence, optionality, and number of repetitions of components, but also inter-component format

restrictions. Table 4.1 presents the number of each kind of inter-component restriction were asserted, The vast majority of these are assignments of permissible coded values to appear in a given point in the structure. Usually these are subsets of code sets that are applicable for a given data element. These could have been coded as separate code sets, but the choice was made to assert them as format restrictions.

The complete set of ontologies and knowledge bases is published on the TripCom website at http://tripcom.org/ontologies/EDI/EDIFACT/.

5 EDIFACT Semantics Ontologies and Knowledge Bases

In order to define the semantics (meaning) of EDIFACT messages using an ontology, the ontology for encoding the semantics of EDI messages has to be developed, a description of the meanings of the message types and their components must be obtained, those meanings need to be represented in an ontology (or set of ontologies), and those meanings need to be linked to the EDIFACT messages using the semantics ontology.

The first of these steps was handled by using an existing ontology for encoding EDI semantics.

5.1 Development and maintenance processes

Meaning occurs in a standardized message at a number of different levels. In analyzing such meaning, one must distinguish between a standardized type of message (a "message type" or "abstract message") and an individual message that conforms to that type (an "instantiated message"). Of course, the individual parts of an instantiated message have their own meanings. Various components of the abstract message can be instantiated with message parts whose meaning is restricted to certain types of things. This can be modeled as the meaning of those abstract components being those types to which their instantiations are restricted.

We decided to use a system of templates[3] for encoding the meaning of EDI messages and their parts which was developed by DERI, Galway. This ontology is available in the WSML language. The system represents meaning at different levels.

5.2 Message Templates

A template for an information structure provides all the information necessary for generating a sentence in the logical language which is implicit in an instantiation of that structure. A complex information structure such as an EDI message will have multiple templates for different implicit sentences. The template specifies the predicate and each argument for the sentence. Each of these values may be derived from the instantiated information structure or be explicitly fixed in the template.

The system selected, encoded in WSML-Flight, contains a set of relations and classes for expressing such templates as described below. Their definitions can be found at http://www.wsmo.org/TR/d27/v0.2/ontologies.

WSML was used because it is a Semantic Web language which permits ternary relations and allows for classes and relations as arguments (in WSML-Flight and more expressive versions).

The meanings of the simplest data elements may be simple data values, individuals, classes, or relationships among any of these other types. This type is associated by a template statement to the data element type. For an specific message, this meaning of an instantiated data element may either be looked up in a code set or be a known or new instance of the type specified in the template. If the meaning does not come from a code set, the relationship between the message text and the represented thing needs to be defined, such as the name or a ship or a number of grams. The main referent of a more complex message component is taken to be its basic "meaning". Templates

for complex message components normally specify the meanings for their slots and relations that interrelate the meanings of various components.

These templates are stated, not using rules, but by relations which can take classes and relations as arguments.

5.2.1 Template Classes

The main classes defined for encoding the meaning of EDI messages are for templates, code sets, formats, and positions in formatted structures. The formats used for specifying EDI syntax described in [3] are used. Classes defined for these templates (which were not previously defined for encoding EDI message formats) include:

CodeSet: A mapping between short text strings and a set of things.

ComponentTemplate: A structure for determining a statement encoded in an instantiation of an Information Structure.

ComponentTemplate_Matching: A ComponentTemplate in which the generated statement must already exist in the data base. This can be used to bind a term being generated or restrict the range of the assertion to an existing element in the database. If all terms are already bound, it is a request for verification that the statement is known.

FormulaArgPosition: A position in a formula, specifically in a formatted structure. Attributes indicate the position relative to a given structure. The position may be the nth subcomponent, the structure itself, any level of surrounding component, or any numbered subcomponent of any of these, recursively.

MultiCodeSet: A CodeSet in which a code has different parts, each of which has its own mapping. E.g., a MultiCodeSet for paper type might have its first conjoined code for paper size, its second for paper color, and its third for paper weight.

5.2.2 Template Relations

The relations applicable to these template classes specify the types of things meant by the fillers of components of an information structure, how these types are encoded, details about code sets, and how to specify statements encoded in the message using the message components. Table 5.1 shows the basic types of template relation, along with the argument types for each relation. The meaning of each of these relations is specified below.

During the ontologization process, it was discovered that EDIFACT messages had features which had not be detected in the ANSI X12 messages for which the EDI ontologizing system was designed. Some of these were features of the X12 messages which were not adopted into the ontology, while others were just not encountered. Relations to specify these features were added to the ontology to enable templates to use them. These relations are listed with the preexisting relations, below, but are distinguished by being printed in *italic* type. EDIFACT did not demonstrate MultiCodeSets, so they were not used in the ontologization process and special predicates related to them are not discussed below.

componentOfType is used to indicate that an instantiation of the specified information structure means an instance of the specified class.

componentOfEncodedType is used to indicate that an instantiation of the specified information structure means an instance of the class encoded in a given argument position.

componentOfSubtype is used to indicate that an instantiation of the specified information structure means a subclass of the specified class.

subcomponentOfType is used to indicate that the filler for the Nth position of an instantiation of the specified information structure means an instance of the specified class, N being the number given as the second argument.

subcomponentOfSubtype is used to indicate that the filler for the Nth position of an instantiation of the specified information structure means a subclass of the specified class, N being the number given as the second argument.

subcomponentRelationArity is used to indicate that the filler for the Nth position of an instantiation of the specified information structure is a relation of the specified arity. E.g., an arity of 2 indicates a binary relation.

subcomponentSubrelationOf is used to indicate that the filler for the Nth position of an instantiation of the specified information structure is a relation which is a subrelation of the relation specified in the third argument.

hasSameMeaningAs is used to indicate that the filler for the first specified position in the structure has the same "meaning" as the filler for the second. The same term will be generated for each in the derived sentences.

directlyEncoded is used to indicate that the "meaning" of an instantiation of the Nth slot in an information structure is the same as that of whatever fills it. For example, the "meaning" of a directly encoded data element might be a character string, while the "meaning" of a non-directly encoded data element might be the city whose name is the character string. The meaning of a slot in a data segment is often directly encoded as the meaning of the data element which fills it. However, sometimes that data element can be used multiple ways, so that its meaning may remain a text string, while in the context of a data segment, its meaning is clear.

functionalPredicateEncodes indicates that the "meaning" of a filler for an instantiation of the specified position relative to the specified information structure is the unique thing that would be related to the filler by the specified predicate, e.g., hasSocialSecurityNumber is a functional predicate that can be used to uniquely identify people given their social security numbers. This is a functional predicate.

componentIsBinaryPredicate is used to indicate that the meaning of the component is a binary predicate.

stringValuedComponent is used to indicate that the meaning of the component is a text string.

integerValuedComponent is used to indicate that the meaning of the component is an integer.

integerValuedComponent is used to indicate that the meaning of the component is a decimal number.

stringValuedSubcomponent is used to indicate that the meaning of the nth component is a text string.

integerValuedSubcomponent is used to indicate that the meaning of the nth component is an integer.

Predicate	Argument Types
componentOfType	InfoStruct Class
componentOfEncodedType	InfoStruct PosInteger
componentOfSubtype	InfoStruct Class
subcomponentOfType	InfoStruct NonNegInteger Class
subcomponentOfSubtype	InfoStruct NonNegInteger Class
subcomponentRelationArity	InfoStruct NonNegInteger NonNegInteger
subComponentSubrelationOf	InfoStruct NonNegInteger Relation
hasSameMeaningAs	InfoStruct FormulaArgPos FormulaArgPos
directlyEncoded	InfoStruct NonNegInteger
functionalPredicateEncodes	InfoStruct FormulaArgPos BinaryPred
usesCodeSet	InfoStruct CodeSet
subcomponentUsesCodeSet	InfoStruct PosInteger CodeSet
subcomponentUsesEncodedCodeSet	InfoStruct PosInteger PosInteger
encodedAs	Thing CodeSet String
encodedAsInverse	Relation CodeSet String
componentIsBinaryPredicate	InfoStruct Boolean
stringValuedComponent	InfoStruct Boolean
integerValuedComponent	InfoStruct Boolean
decimalValuedComponent	InfoStruct Boolean
stringValuedSubcomponent	InfoStruct PosInteger
integerValuedSubcomponent	InfoStruct PosInteger
decimalValuedSubcomponent	InfoStruct PosInteger
templateForComponent	ComponentTemplate InfoStruct
templateForNthInstance	ComponentTemplate PosInteger
templateRelation	ComponentTemplate Predicate
templateRelationEncoded	ComponentTemplate FormulaArgPos
templateArg[N]Encoded	ComponentTemplate FormulaArgPos
templateArg[N]Value	ComponentTemplate FormulaArgPos
templateArg[N]	ComponentTemplate Thing
templateSubcomponentMustBeRelation	ComponentTemplate PosInteger
templateSubcomponentMustBeOfType	ComponentTemplate PosInteger Class
templateSubcomponentMustBeOfSubType	ComponentTemplate PosInteger Class

Table 5.1: Template Relations

integerValuedSubcomponent is used to indicate that the meaning of the nth component is a decimal number.

usesCodeSet indicates that the specified code set maps the filler for an instantiation of the specified information structure to its "meaning". This is not a functional predicate – sometimes codes may come from any one of several code sets.

subcomponentUsesCodeSet indicates that the specified code set maps the filler for an instantiation of the specified slot to its "meaning". Some Data Elements can be encoded by more than one code set. The actual code set used depends on its enclosing Data Segment or context. This is not a functional predicate.

subcomponentUsesEncodedCodeSet indicates that the meaning of a code in instantiation of the specified slot should be looked up in the code set referenced in the

second specified slot. Only if the code at this point in the structure is not in the specified code set, the default code set(s) for this component is used. This is a functional predicate.

encodedAs is a ternary predicate which indicates that the specified thing is encoded in the specified code set with the specified code.

encodedAsInverse is a ternary predicate which indicates that the inverse of the specified binary relation is encoded in the specified code set with the specified code. The arguments should be reversed when a statement is generated from the template.

templateForComponent indicates a template for a logical sentence encoded by instantiations of a given information structure. An information structure may have multiple templates. templateRelation indicates the predicate for a given template. This is a functional predicate.

templateForNthInstance indicates a template is only applicable for the nth instance (within the smallest subsuming data structure) of the component to which it is applied.

templateRelationEncoded indicates that the predicate for each use of a given template is the one encoded at the specified position of the information structure which the template is for. This is a functional predicate.

templateArg[N]Encoded (for N being 1, 2, or 3) indicates that the Nth argument for each use of a given template is the value encoded at the specified position of the information structure which the template is for. This is a functional predicate.

templateArg[N]Value (for N being 1, 2, or 3) indicates that the Nth argument for each use of a given template is the value present (not encoded) at the specified position of the information structure which the template is for. This is a functional predicate.

templateArg[N] (for N being 1, 2, or 3) indicates that the Nth argument for each use of a given template is the specified value. This is a functional predicate.

templateSubcomponentMustBeRelation means that for the template to be used, the value of the Nth slot of the component to which it is applied must be a relation.

templateSubcomponentMustBeOfType means that for the template to be used, the value of the Nth slot of the component to which it is applied must be an instance of the specified type.

templateSubcomponentMustBeOfSubType means that for the template to be used, the value of the Nth slot of the component to which it is applied must be a subclass of the specified type.

5.3 Describing EDIFACT Subset Semantics using Ontologies

The descriptions of the components of EDIFACT messages in the message structure charts provided for each EDIFACT subset generally provide a line describing the meaning of the component. Often there are notes explaining the usage of the the term in greater depth. A paragraph or two describe data segments, and discussion of data segments is distributed through the document.

In order to encode the meaning of the EDIFACT messages, several things are necessary:

- The types (classes) of things encoded in the message are defined in an ontology.

- The relations among things encoded in the message are defined in an ontology.
- Any instances of things regularly used in the message, e.g. currencies, are defined in an ontology or knowledge base.
- Any sets of possible coded values of specific data elements are represented as code sets.
- Each code set is populated with the mappings between each of its codes and their meanings.
- Each data element, data segment, data segment loop, and message must have its basic meaning defined in a knowledge base.
- Each implicit relationship among components of must be encoded in a template in a knowledge base.

In order to create the ontologies of terms needed to express the meaning of message components, we first went through each of the code sets provided for the data elements of the messages in our selected EDIFACT subset. Subsequently, the meanings of the uncoded data elements were considered, along with the relationships that are implicit among different components of each message.

In parallel with creating ontologies of the terms derived from the message types, we created (Code Sets) knowledge bases linking those terms to the data element codes and holding the templates describing other relationships.

5.3.1 Generic Ontologies

The ontologies created to define the concepts and relations referred to in the messages were considered "generic" ontologies, since they were about the things the messages referred to, not about EDIFACT itself. To assist in constructing these ontologies, a search of preexisting ontologies was made and the OpenCyc ontology[1] was found to already have many of the concepts used by EDIFACT. In order to construct the generic ontologies for the EDIFACT messages and tie them together, they were tied to the OpenCyc terms, with appropriate snippets of the OpenCyc ontology being attached to connect the generated ontologies together.

The range of meaning of the data element was first considered and defined as one or more classes in an ontology. Then the meaning each code was determined and was added to the ontology as a subclass or instance of the basic class(es) for the data element. As these were added, a subclass hierarchy of the basic class(es) was developed, sometimes with the new class a sub- or superclass of an already defined class. Often, a set of terms in a code set were determined to naturally comprise a subclass of the basic class for the data element, in which case that subclass was created and inserted appropriately into the growing ontology.

Separate ontologies were initially created for the terms from different code sets for simple data elements. In cases in which different data elements had closely related or overlapping concepts in their code sets, a single ontology was created for the terms from the set of code sets. This separation was done in order to prevent people working

[1] http://www.opencyc.org/

in different areas from interfering with each other's work. As the terms in each ontology are connected with the same set of more general concepts, later merging of the ontologies is facilitated. Some of this merging was performed during the ontologization of the code sets, while further merging was saved for later.

After the code sets were ontologized, several steps were taken concurrently:

- ontologizing the meaning of data elements for which a code set was not specified
- determining implicit relations among data elements (and more complex message components)
- ontologizing the discovered relations
- merging and consolidating previously created ontologies
- creating templates for statements implicit in the complex message components
- placing these templates in knowledge bases for the associated data segment or message type

Uncoded data elements are frequently associated with coded ones, being used to store names or ID strings for instances of the type of item which a coded data element represents. Specific relations are needed for many such data elements to specify address components, account numbers of various types, telephone numbers, and similar strings stored in those elements. Other uncoded data elements are numeric values for which the unit of measure is either implicit or indicated by another data element. Some uncoded data elements can be only roughly ontologized as they are designed for free text comments from the sender which are basically unrestricted.

Ontologizing these remaining data elements did not involve creating new ontologies, although they sometimes encouraged the merging of ontologies which had been created in the previous step.

As relations implicit in the data messages were discovered, they were defined in ontologies which had access to the concepts which were their argument types. This access could initially be done by "including" the ontologies in which the concepts were defined in those in which defined the relations were defined, with the intention of merging the ontologies later, or the ontologies could be merged at that time. Both techniques were used.

5.3.2 Knowledge Base by Structural Level – Data Element, Segment, Message

In order to assert the templates for statements implicit in data segments, knowledge bases were created to contain the templates for those segments. For templates that spanned data segments, knowledge bases were created for the message types along the lines of the syntax knowledge bases created earlier. In cases where the message syntax KBs handled multiple message types, the related message semantics KBs did also, since many of the templates valid for one message are valid for other messages whose formats significantly overlap.

The encoded data elements had separate knowledge bases defined for each of their code sets. Since different EDIFACT subsets often use different code sets – either different subsets of the standard EDIFACT code sets or included additional codes

which were only used for one industry's subset – more than one code sets, and thus knowledge bases, were created for many of the data elements.

It seemed useful neither to use a single knowledge base to specify the types of all the uncoded data elements (because they would need to access concepts from a wide array of generic ontologies before they were merged) nor to create separate ontologies for each (because there were so many), so their meanings were ontologized along with the data segments in which they are present.

Each of these knowledge bases must reference (in WSML, "include") the generic ontologies which defines the terms to which the knowledge bases refer as well as the ontologies defining the EDI template system.

5.4 Example Templates

We present example templates for the composite data element, C082 "Party identification details", taken from the developed knowledge bases. The complete set of developed knowledge bases is available at http://tripcom.org/ontologies/EDI/EDIFACT/.

- The C082 compound data element represents an organization or person:

```
relationInstance edis#componentOfType (cde#C082_DE, cyc#LegalAgent)
```

- The first data element of C082 is an identifying string ...

```
instance cde#C082_DE memberOf edi#ComplexDataElement
         stringValuedSubcomponent hasValue 1
```

- ... which is the same string as that encoded in its data element:

```
relationInstance edis#directlyEncoded (cde#C082_DE, 1)
```

- The third data element of C082 is a code set identified by a code set:

```
relationInstance edis#subComponentOfType
                 (cde#C082_DE, 3, edis#CodeSet)
relationInstance edis#subcomponentUsesCodeSet
                 (cde#C082_DE, 3, ecs#DE_3005CodeSet)
```

- The second data element of C082 represents a binary predicate ...

```
relationInstance edis#subComponentRelationArity (cde#C082_DE, 2, 2)
```

- ... which is a specialization of identificationStrings ...

```
relationInstance edis#subComponentSubrelationOf
                 (cde#C082_DE, 2, cyc#identificationStrings)
```

- ... and encoded using the code set specified by the third data element:

```
relationInstance edis#subcomponentUsesEncodedCodeSet
                 (cde#C082_DE, 2, 3)
```

- If there is no third data element or the code is not found in the code set that element refers to, the code is obtained from a default code set:

```
relationInstance edis#subcomponentUsesCodeSet
                 (cde#C082_DE, 2, ecs#DE_4405CodeSet)
```

For EDIFACT subsets, additional subsets should be used, such as ecs#DE_4405FCodeSet for EDIFICE, ecs#DE_4405ECodeSet for EANCOM, and ecs#UNRec24CodeSet if UN export codes are used.

- The relation in the second data element of C082 specifies a relation for the way that the party which is the meaning of the composite data element is identified using the ID in its first data element.

```
instance C082Template1 memberOf edis#ComponentTemplate
    edis#templateForComponent hasValue cde#C082_DE
    edis#templateRelationEncoded
         hasValue scp#SecondSubcomponentPosition
    edis#templateArg1Encoded hasValue scp#ThisFAPosition
    edis#templateArg2Value
         hasValue scp#FirstSubcomponentPosition
```

5.5 Created Ontologies and Knowledge Bases

The EDI ontologies were used to define almost 240 code sets for almost 150 coded data element types present in at least one of the subsets we have selected, resulting in the assignment of values for over 4680 codes. [This figure excludes the codes for the UN location codes (UNLOCODES) which are used in EDIFACT and defined for over 100,000 locations on Earth.] In cases in which different subsets used different codes for the same data element, the generic EDIFACT codes were encoded in a code set for the data element, while separate code sets for the distinct subsets only held codes that were not common.

Although there are frequent instances in which codes for different data elements (and thus in different code sets) represented and were mapped to the same thing, this still left well over 4000 different terms defined and incorporated in the ontologies. In addition to the explicitly referenced terms, natural groups of these concepts were defined as superconcepts of each of the members of the group, and these general concepts in turn were often defined as subconcepts of even more general concepts. These additional concepts were either found in the OpenCyc ontology or created for the EDIFACT ontologies.

In all, over 2400 classes and around 850 relations were created for the these ontologies and over 1000 classes and about 90 relations used from the OpenCyc ontology. About 730 of the OpenCyc terms exactly matched codes in the code sets, while the others were used either for non-coded data elements, (or more complex structures) or were generalizations of the concepts and relations which exactly matched.

In total, over 4000 terms were used from the OpenCyc ontology. This included around 2900 “individuals” (items neither classes nor relations; including languages, countries, cities, currencies, and units of measure).

Creating templates for the statements implicit in the EDIFACT messages took more effort than originally estimated. Because the templates being created turned out not to be useful to the use cases, the focus turned to developing the ontologies of classes and relations needed to express the meanings of EDIFACT messages, and away from creating the templates. As the task of the work package is to create ontologies so that what is stated syntactically in EDIFACT messages can be expressed semantically, and not to develop an adapter to convert existing EDIFACT messages into a semantic form, this was deemed to be a more appropriate use of limited resources.

6 Lessons Learned

The lessons learned from the EDI ontologization process deal with the possibility of storing complex syntactic messages semantically, the utility of using a massive preexisting ontology to ease the process of ontology creation, similarities and differences of approach for different complex syntactic message systems, and applicability of general message systems such as EDIFACT to newly designed business applications. The different phases of the project, ontologizing first the syntax and then the semantics of the EDIFACT system each contributed to these lessons learned.

6.1 Lessons Learned from Ontologizing Syntax

The lessons learned from ontologizing syntax primarily dealt with distinctions between two different EDI systems and thus with the applicability of a tool built (with generality in mind) for one EDI system to be used for other systems.

- The ontology developed for encoding ANSI X12 EDI message formats could be used without modification to encode the basic formats of the EDIFACT messages.
- Despite its vast size, a single EDI system does not demonstrate every type of inter-component format restriction used in complex syntactic message systems. X12 used some such inter-component restrictions that the EDIFACT subsets we selected don't use and EDIFACT uses some inter-component restrictions that X12 does not have.
- The ontology structure was such that defining new inter-component restriction types is simple.
- EDIFACT differs from ANSI X12 in using composite data elements to a far greater extent.
- EDIFACT differs from ANSI X12 in having far fewer data elements that are used in multiple data segments.
- The EDIFACT subsets covered do not use the multi-code sets occasionally found in X12 (in which a code for a single data element can be broken down into more than one component codes which are individually defined).

6.2 Lessons Learned from Ontologizing Semantics

- Ontologies of business-related concepts can be developed from data element code sets.
- EDIFACT sometimes uses multiple instances of a structure such that they bear a different relationship to other components depending on their repetition number (first, second, third) instead of coding that relationship within the component.
- Individual EDIFACT subsets may use some data elements for different purposes.

- Data element code sets often contain several disparate classes of meaning for the different purposes to which the data element is put.
- Defining templates as objects in their own right allowed the system to be expanded define conditions as to when a given template is applicable. These include:
  - when a given component is an integer;
  - when a given component is a relation;
  - when a given component is an instance of a given class;
  - when a given component is a subclass of a given class; and
  - that the template only applies to the nth repetition of a data structure.
- OpenCyc had most of the general concepts for data elements and data segments, which provided a good basis on which to build more detailed ontologies for the terms in the code sets, not only providing terms, but helping to structure them into overlapping hierarchies.
- The relations implicit in the messages (or even generalizations of them) were often missing from OpenCyc.

7 Discussion and Further Work

7.1 Use of Developed Ontologies in TripCom

7.1.1 Further Efforts in Work Package 7

In Deliverable D7.3, "Relationships among Ontology Modules within EDIFACT and with External Ontologies", the numerous small ontologies created for this deliverable will be integrated with each other, further with OpenCyc, and with other external ontologies. In Deliverable D7.4, "Evaluation and Refinement of Ontologies", missing aspects of the merged ontology will be filled in and the quality of the knowledge bases describing the message and component semantics will be examined and improved.

7.1.2 Use by Other Work Packages

Although both use cases reviewed the EDIFACT messages identified as applicable to their domains, they decided to design their own formats and ontologies, using the EDIFACT formats and the ontologies developed in WP 7 only for reference. Work Package 8A used the EDIFACT messages to inform their design of RDF messages targeted for their application. Work Package 8B selected different existing standards – the Continuity of Care Record (CCR [8] and the Clinical Document Architecture (CDA [6]) – and built ontologies based on them, after considering the EDI medical messages and the ontologies built for them.

Although the knowledge bases are encoded in WSML-Flight, WSML can not be used to express rules for templates having relations as arguments since it disallows rules using variables (or similar constructs) to represent relations. In order to use the templates and format descriptions to convert EDIFACT messages to and from sets of WSML statements, a more general computer language (such as Java) would have to be used to process the templates to generate WSML translations from EDIFACT input.

Work Package 7 was not scheduled to create data mediators or adapters to convert actual EDIFACT messages from EDIFACT syntax to a semantic language or vice-versa. With the use cases choosing not to interact with outside systems using EDIFACT, Work Package 4 found no reason to develop such an adapter.

7.2 Future Uses of Developed Ontologies

Although a computer ontology is supposed to be a formalised shared description of a domain [7], the amount of ontology reuse is not nearly as extensive as it could be. Even huge ontologies, such as OpenCyc, are normally not referenced when a new project needs an ontology. Standard practice is to design one's own ontology, even if the area has been covered numerous times in the past.

This bodes ill for any ontology made available to the public, including those generated in this Work Package, to be reused by groups unassociated with its construction. In order to be widely reused, it is likely that an ontology would have to be made a visible part of a successfully marketed software package.

Work Package 7 has developed general ontologies to cover the concepts of EDIFACT messages and knowledge bases defining the syntax of those messages and semantics

of their components. The most likely way for these ontologies and knowledge bases to be reused would be for a new project to create industrial quality adapters which would use the ontologies and templates to actually convert business messages between a semantic format and EDIFACT. An EDI software vendor might find doing this to be of value for enabling the automatic translation of business messages from one encoding system to another, with the semantic language being used as an interlingua among a set of diverse standards.

Other than this, the ontologies themselves would stand a chance of general reuse if translated into CycL (which would be a relatively mechanical process) and added to the existing OpenCyc ontology.

Promoting the set of ontologies as Yet Another Broad Ontology would be unlikely to achieve significant reuse.

The individual partners could decide to adopt the developed ontologies for internal use and the business partners could incorporate them into their products.

7.3 Conclusion

We conclude that large ontologies constructed with generality in mind can be reused by new projects. This is on the basis of our reuse of both the NUIG ontology built for ANSI X12 EDI and the OpenCyc ontology.

On the other hand, a large general system, almost by definition, contains a lot of material not applicable to an individual project that chooses to use it. Individual industries create their own subsets of EDIFACT, throwing most of the standard away. In encoding the EDIFACT system's semantics, we selected only a small subset of the OpenCyc system for modeling it. Work Package 8A determined that it did not need most of the content of EDIFACT messages this work package was encoding for catalogs, orders, and purchases, and so built their own small ontology using the larger ones only for reference.

EDIFACT was chosen for TripCom because it is designed to have broad coverage – to be able to accommodate any version of almost any type of business message. However, because this necessitates that any message description be filled with optional components, picking and choosing what is useful becomes difficult and it is difficult for a designer not familiar with the system to determine how to encode what s/he knows should be included in the message. Thus it becomes easier to design one's own ontology and message type than to adapt the heavily adaptable "standard" message.

References

[1] TripCom Consortium. Annex I – Description of Work. http://tripcom.org/internal/document/25, 2006.

[2] d. foxvog and C. Bussler. Ontologizing EDI: First Steps and Initial Experience. Proceedings International Workshop on Data Engineering Issues in E-Commerce (DEEC 2005), 2005.

[3] d. foxvog and C. Bussler. Ontologizing EDI Semantics. Proceedings First International Workshop on Ontologizing Industrial Standards (OIS 2006), 2006.

[4] d. foxvog, D. Cerizza, D. de Franscisco, A. Ghioni, V. Momtchev, M. Murth, L. Nixon, E. Simperl, T. Scheibler, D. Wutke, A. Polleres, and A. Carpentier. TripCom Deliverable D10.1: TripCom Detailed Work Plan. http://tripcom.org/internal/document/400, 2006.

[5] d. foxvog, J. P. Palacios, and E. P. B Simperl. TripCom Deliverable D7.1: Analysis of EDIFACT and other Standards. http://tripcom.org/docs/del/Tripcom-D71.pdf, 2006.

[6] R. H. Dolin, L. Alschuler, C. Beebe, P. V. Biron, S. L. Boyer, D. Essin, E. Kimber, T. Lincoln, and JE. Mattison. The HL7 Clinical Document Architecture. Journal of the American Medical Information Association, v. 8, pp. 552 - 69, 2001.

[7] T. R. Gruber. A Translation Approach to Portable Ontology Specifications. Knowledge Acquisition, v. 5, pp. 199 - 220, 1993.

[8] D. C. Kibbe, R. L. Phillips, and L. A. Green. The Continuity of Care Record. American Family Physician, v. 70, pp. 1220 - 3, 2004.