A Multi-channel Communication Framework

Michal Nagy

University of Jyväskylä, P.O. Box 35 (Agora), FI-40014, Jyväskylä, Finland
michal.nagy@jyu.fi

Abstract. We present a modular framework for a business-to-customer communication service integrating several communication channels. Using such a service it is possible to find hidden relationships between messages and thus collect more customer-related data. The framework core is a message-conversion engine capable of converting channel-independent abstract messages into concrete messages and vice-versa. The conversion process is context-dependent. The context consists of formally described communication channel characteristics and user preferences. The framework is based on semantic technologies due to a balance between their expressive power, reasoning properties, and existence of production-quality tools. This chapter describes the multi-channel communication framework with relation to its components. Among others we discuss message conversion and channel selection.

Keywords: communication, ontology, context-awareness, B2C.

1 Introduction

The introduction of electronic media into businesses changed the way businesses communicate with their customers and between each other. A typical business has nowadays several options how it can reach its customers and how the customers can reach the business. We call these means of communication communication channels. With the increasing number of communication channels the problem of consolidation and integration becomes more complicated.

We will illustrate this problem on an example of a consumer electronics seller trying to communicate with a potential customer. The seller has a web page, Facebook page, Twitter account, phone number and an email address. These are the communication channels. The seller may start a campaign selling a new model of a TV. The seller sends a new message to all followers on Twitter and Facebook. The next day a customer asks a question about the new product through the seller's web page. The seller replies by sending an email to the email address that the customer provided in the web form. Then the customer buys the product using a form on the seller's web page. The example illustrates communication through several communication channels – social networks, web forms and email. It would be useful for the seller to understand that the customer asking the question through the web form is the same person that later bought the product. This way the seller may build up a database of all communication

V. Ermolayev et al. (Eds.): ICTERI 2012, CCIS 347, pp. 72–88, 2013.

happening between the business and a particular customer no matter what communication channel is being used. The seller may for example find out that the customer prefers email communication in 90% of cases and that he/she prefers to buy the products in person in the brick-and-mortar shop instead of buying them online. This may differ from another customer that prefers communication through Facebook and always buys product electronically through the electronic shop.

In order to make such a system we have to overcome several issues. In [7] we present five issues related to multi-channel communication – message conversion, communication channel selection, information element extraction, goal/purpose modeling and context modeling. In this publication we propose a framework that tries to solve these issues. The framework is mostly based on semantic technologies and utility functions. The emphasis was given to the real-world implementation.

Section 2 talks about the motivation behind this effort. Section 3 describes the framework and briefly discusses the main framework components. Section 4 defines ontologies that are used by various components of the framework. Section 5 describes engine algorithms used for message conversions and channel selection. Lastly, section 6 concludes the chapter.

2 Motivation

A multi-channel communication system brings several advantages to businesses. Firstly, it allows them to collect more meaningful information about their customers. Nowadays, information systems can collect a significant amount of data related to B2C communication. However, more data does not necessarily mean more information. Isolated facts become more useful once they are linked with each other given a certain context. This is the motivation of the Linked Data initiative as well [1]. This way hidden links between data can be detected.

Every communication effort initiated by the business should be rational and should be done with a goal and purpose (e.g. promoting a product with the goal of selling it). Better aimed communication increases the chances of goals being fulfilled. In order to make these rational decisions, one must be well informed. There should be a system capable of collecting information about every customer – past actions, preferences, financial profile, etc. Instead of sending a message to concrete customers, a business may decide to send a message to a certain type of customer. For example if the business tries to advertise a TV, it should advertise it one way to home cinema enthusiast, but other way to customers interested in products with low energy consumption.

There is a significant amount of business-relevant information on the Internet. The problem is however that the information is difficult or even impossible to understand by a machine [4]. The situation is changing. Technologies such as RDF (Resource Description Framework), OWL (Web Ontology Language) allow people and businesses to publish this information in a machine-readable form on the Internet. There are many publicly available RDF storages that contain useful

information (e.g. DBpedia [3], data.gov.uk portal [2], etc.). It could be beneficial to utilize this information in business' decision making process.

3 Multi-channel Communication Framework

3.1 Framework Overview

The framework consists of two main components – knowledge base and message conversion engine. The goal of the knowledge base is to store information about five main areas – messages, communication channels, customers, commodities and business actions. All these partial knowledge bases are linked. The framework user is free to extend the knowledge base and corresponding knowledge schemas (ontologies) to fit the business needs. The message conversion engine is responsible for message conversion. It converts message received from customers to abstract message trying to extract information elements that are relevant to the business. Then it interprets them in form of actions. The framework user may specify several message templates to send message to the customers. The engine converts message templates into concrete messages that can be sent to the customer. It is also responsible for proper channel selection that may depend on customer's personal preferences and other circumstances. The framework overview is available in Figure 1.

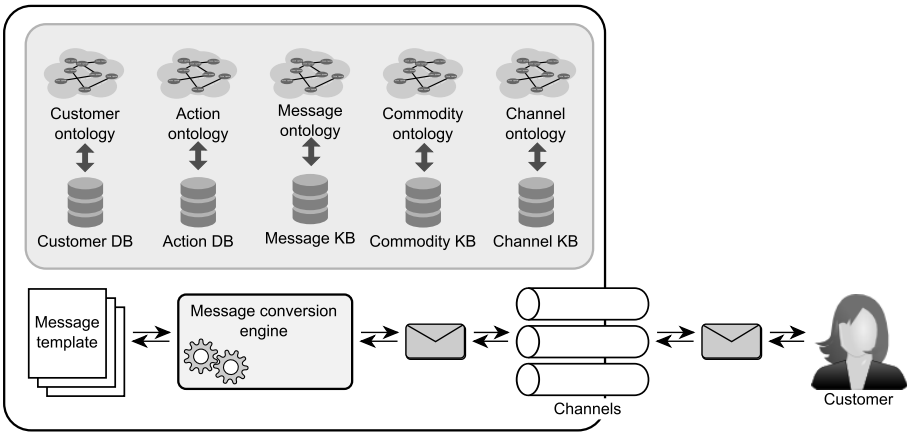

Fig. 1. Multi-channel communication framework overview

3.2 Knowledge Base

The system should be capable of storing different types of information about customers, channels and business. Some of this data is relatively static (e.g. product data) and some data is dynamic (e.g. customer information). However,

we believe that a single way of representation is favorable due to the need to integrate and reason about all these kinds of data together. Reasoning will take place mostly in two cases – when a message is received and when a message is sent.

We need a way of knowledge modeling that would be expressive enough to describe relationships between heterogeneous entities such as customers, products, services, channels, etc. Moreover, the language must allow reasoning about the facts. This reasoning must be sound, so that it infers only valid facts. It should be complete as well, so that no valid fact is missing. If a valid missing was missing, it would result in a failure to send an outbound message or to properly annotate an inbound message. Lastly, this modeling framework must not be computationally too complex. As mentioned earlier, the discussed system should be usable in real-world production environment. Therefore the existence of a mature tool with abovementioned properties is important.

Based on these requirements we believe that Resource Description Framework (RDF) is the most reasonable way to model our data. It is expressive enough, proven and widely supported by a great amount of production-quality tools for modeling (e.g. Protégé [8]), storing and querying data (e.g. Jena framework [6], Sesame [12]). Also, there are many reasoners that are sound, complete and still computationally inexpensive (e.g. RacerPro [9], HermiT [5], etc.). RDF is built on top of wide-spread standards such as XML, XML Schema, etc. It is closely related to Web Ontology Language (OWL) which is used to formally define knowledge schemas called ontologies. Thanks to ontologies and reasoners it is possible not only to conclude new facts from existing ones, but to check data consistency as well.

4 Ontologies

4.1 Overview

In this section we present several ontologies that are needed for the framework's operation. Each ontology belongs to a different namespace due to logical division. Later in the text we use prefix names to refer to specific namespaces. They are described in Table 1.

Prefixes xsd, owl, rdfs and rdf represent well-known namespaces and ontologies. All the other prefixes represent framework-specific namespaces and ontologies. The commodity ontology represents all the products and services that the business is trying to sell. The message ontology represents abstract and concrete messages. The action ontology is about actions that the business or a customer can perform. The channel ontology is used to describe communication channels. Lastly, the customer ontology represents a customer schema including customer personal information, contact addresses, preferences, etc. In order to conserve space the ontologies are not described directly in OWL. For each ontology the class hierarchy is graphically depicted in a figure and the list of preferences is described in a table.

Table 1. Namespaces and prefixes used in this chapter

Prefix	Namespace	Ontology
b	`http://cs.jyu.fi/ai/OntoGroup/mmcc/commodity#`	Commodity ontology
a	`http://cs.jyu.fi/ai/OntoGroup/mmcc/action#`	Action ontology
m	`http://cs.jyu.fi/ai/OntoGroup/mmcc/message#`	Message ontology
ch	`http://cs.jyu.fi/ai/OntoGroup/mmcc/channel#`	Channel ontology
cu	`http://cs.jyu.fi/ai/OntoGroup/mmcc/customer#`	Customer ontology
xsd	`http://www.w3.org/2001/XMLSchema#`	XML Schema [15]
owl	`http://www.w3.org/2002/07/owl#`	Web Ontology Language [14]
rdf	`http://www.w3.org/1999/02/22-rdf-syntax-ns#`	Resource Description Framework [10]
rdfs	`http://www.w3.org/2000/01/rdf-schema#`	RDF Schema [11]

4.2 Commodity Ontology

In general the commodity ontology depends on the business domain and can comprise of a variety of goods and services. However, there is a single primitive generic ontology that can be extended based on the needs of the business. The generic commodity ontology together with an example of an extension can be seen in Figure 2. The extension is based on a scenario where the user of the framework is a small consumer electronics seller.

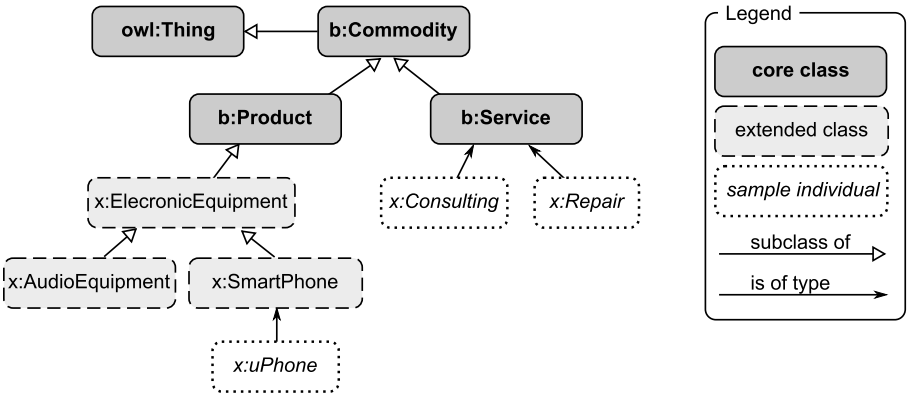

Fig. 2. Class hierarchy of a generic commodity ontology and its sample extension

The central point of the ontology is the class b:Commodity. A commodity represents either a product (b:Product) or a service (b:Service). Class b:Commodity also acts as an integration point to other ontologies, that means other framework ontologies refer to it.

4.3 Communication Channel Ontology

The communication channel ontology is relatively small. The user of the framework needs to extend it with communication channels he/she wants to use in his/her business. The central point of the communication channel ontology is the class ch:ComChannel. The class hierarchy is depicted in Figure 3.

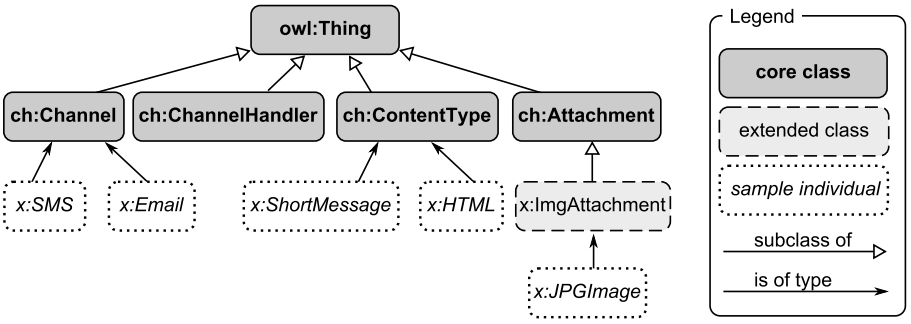

Fig. 3. Class hierarchy of a generic commodity ontology and its sample extension

Each communication channel (or just channel) is characterized by several object and datatype properties. The speed and reliability of the channel is determined by the property ch:speed and ch:reliability. Higher value represents higher speed/reliability. Property ch:cost represents any other considerations – e.g. financial cost. The capability to transfer attachments is expressed through ch:attachmentFeature object property. In general a channel might have no possibility to transfer attachments (e.g. SMS) up to the ability transfer several types of attachments (e.g. electronic mail). For the sake of simplicity we describe communication channel ontology properties in Table 2.

Table 2. Object and datatype properties of the channel ontology

URI	Min. card.	Max. card.	Domain	Range
ch:speed	1	1	ch:ComChannel	xsd:float
ch:reliability	1	1	ch:ComChannel	xsd:float
ch:cost	1	1	ch:ComChannel	xsd:float
ch:attachmentFeature	0	n	ch:ComChannel	ch:Attachment
ch:registeredHandler	1	1	ch:ComChannel	ch:ChannelHandler
ch:acceptsContent	1	n	ch:ChannelHandler	ch:Content
ch:conversionFunction	1	1	ch:ChannelHandler	xsd:string

In general, each channel uses different types of messages and handles them in a different way. Channel handler is a special class that represents the way messages are constructed and manipulated. It stores the list of content types that can be

transferred through the particular communication channel. This is expressed by linking the channel handler to a content instance by ch:acceptsContent property. The list of content types is configurable and fully depends on the framework user. There is a content conversion function associated with each handler. This function takes the content definition from the message template and converts it to a particular content suitable for this channel. Since each channel can deal with different types of content, each channel has its own handler.

4.4 Action Ontology

The main goal of the multi-channel communication framework is to allow message integration among various communication channels. However, messages have no business value unless they are properly interpreted. As mentioned earlier we assume that each message is sent for a reason and that it has a purpose. Therefore we believe that each message is associated with one or more actions. A message is a piece of information that needs to be interpreted under a certain context. Usually a message has a recipient, sender, subject and body. An action is something that a message represents. In our case we are interested in business actions. Some of these actions are related to certain commodities (products or services) such as a purchase, question about a product, ordering a service or a product return. Other actions are not related to any product – e.g. contact information change, change of preferences, etc.

We use class m:Message for representation of messages, class a:Action for representation of actions and class b:Commodity for the representation of commodities. The relationship between these classes is depicted in Figure 4. Object property m:represents represents a connection between a message and action(s). There are two subclasses of a:Action. Subclass a:GeneralAction represents an action that is not related to any commodity. Subclass a:RelatedAction represents an action related to a concrete product or service. This connection is expressed by object property a:regardingCommodity. The specific actions that businesses are interested in can vary and each business should have the ability to define their own types of actions. The ontology in Figure 4is a generic action ontology that can easily be extended by businesses

4.5 Customer Ontology

The customer ontology models information related to personal data of the customers together with their business preferences. Each customer can be reached through several channels. This is expressed through the class cu:Contact. A customer may have several contacts. Each contact is linked to a channel and has a datatype property representing customer's address in that particular channel. Also, the ontology defines a datatype property cu:preference for each contact. The value of the preference property is a real number between 0 and 1. This number expresses the customer's willingness to be informed using this contact. If the value is 0, the user never wants to be contacted. If the value is 1, the user always wants to be contacted. The list of customer ontology properties is

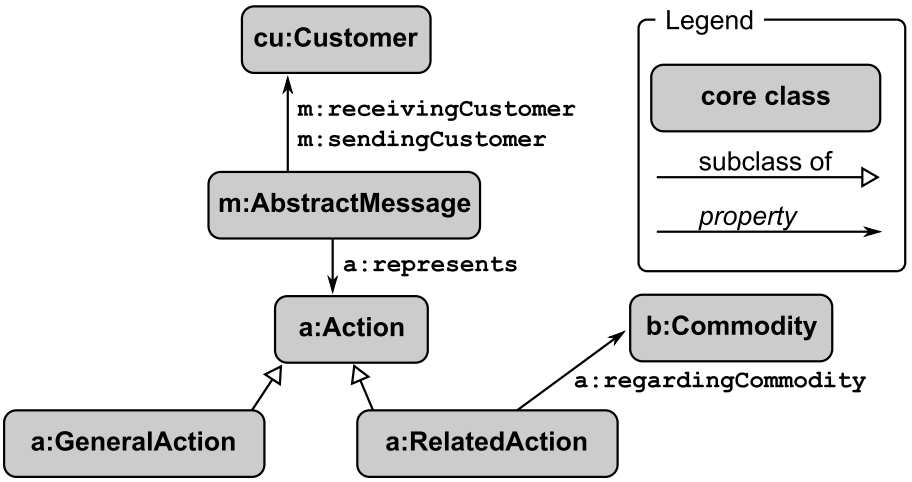

Fig. 4. Relationship between messages, actions and commodities

depicted in Table 3. We do not provide any class hierarchy figure, because both ch:Customer and ch:Contact are direct descendants of OWL class owl:Thing. The customer ontology contains only entities that are important for the proper framework operation. Personal information such as name or birth date are not included in the ontology. The framework user can provide them by extending the ontology.

Table 3. Object and datatype properties of the customer ontology

URI	Min. card.	Max. card.	Domain	Range
cu:hasContact	0	n	cu:Customer	cu:Contact
cu:correspondingChannel	1	1	cu:Contact	ch:Channel
cu:contactAddress	1	1	cu:Contact	xsd:string
cu:preference	1	1	cu:Contact	xsd:float

4.6 Message Ontology

There are two basic types of messages – inbound and outbound. An inbound message is a message sent by a customer to the business. An outbound message is a message originating from the business and sent to one or more customers. According to another criterion a message can either be abstract or concrete. A concrete message is a message sent through a concrete channel from/to a concrete address with a concrete content. In case of a concrete message no information about the action or customer is included. An abstract message is a message on

a higher thought level. It is linked to a concrete customer from the customer database in case the customer is the receiver or sender of the message. It is also linked to the proper action that was supposed to be achieved through it. Both abstract and concrete messages can be either inbound or outbound. That makes it four types of messages. Apart from abstract and concrete messages the framework distinguishes a term called message template. A message template is a prescription for an outbound abstract message. In the message template the recipients, the content and the communication channel are known in a form of a query. Such a template can be converted into an outbound abstract message by resolving these queries. This concept is similar to the idea of executable knowledge by Terziyan [13] and will be described in detail in the next section.

The message ontology consists of four main classes – m:ConcreteMessage, m:AbstractMessage, m:InboundMessage and m:OutboundMessage. The class hierarchy is depicted in Figure 5. The property descriptions are in Table 4.

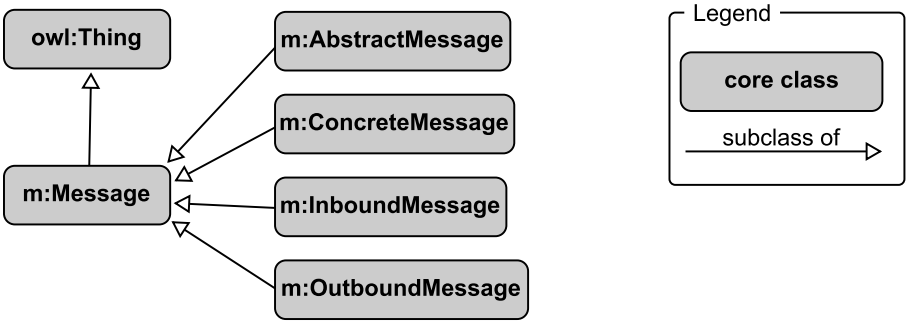

Fig. 5. Class hierarchy of the message ontology

A concrete message has datatype properties annotating the sender, receiver, subject and content. All these properties expect string as their values. As mentioned earlier, a concrete message is not mapped to a concrete customer URI. It only stores the information about the concrete address (string) it was sent to or received from. It is the job of the engine to do the conversion between an abstract and a concrete message. A concrete message also has two object properties. The one connects it to the communication channel that was used to send/receive the message. The other object property points to attachments that were sent or received together with the message. The user can extend the message ontology and specify subclasses of concrete messages such as text message (SMS), email, tweet, etc. Some of these subclasses do not specify certain properties. For example tweets and SMS messages do not have subjects. Some concrete messages may contain attachments and some may not.

Table 4. Object and datatype properties of the message ontology

URI	Min. card.	Max. card.	Domain	Range
m:to	1	n	m:ConcreteMessage	xsd:string
m:from	1	1	m:ConcreteMessage	xsd:string
m:dateReceived	1	1	m:Message	xsd:dateTime
m:subject	0	1	m:Message	xsd:string
m:content	0	1	m:Message	xsd:string
m:channel	1	1	m:Message	c:Channel
m:hasAttachment	0	n	m:Message	m:Attachment
a:represents	0	n	m:AbstractMessage	a:Action
m:receivingCustomer	0	1	m:AbstractMessage	cu:Customer
m:sendingCustomer	0	1	m:AbstractMessage	cu:Customer

An abstract message describes the sender and receiver in form of a URI identifying a concrete individual from the knowledge base. As mentioned in the action ontology description, an abstract message includes the information about an action that the message represents.

5 Message Conversion Engine

5.1 Message Template Description

A message template is a prescription for an abstract message. Naturally, the abstract message generation process requires data that the template should be filled with. The data needed for the generation process is expressed in a form of a query. During the conversion the query is executed by the engine and the result of the query represents that working data set that is used to fill the template with data. A message template is a prescriptive element, not a descriptive one. Therefore we believe that RDF is not the most suitable way of expressing it. In Figure 6 we present a simple language to describe a message template.

The template description consists of four main parts – query, recipient specification, message content and channel specification. The recipient specification defines who should receive this message. It can be a single person or several people. The message content is a piece of information that will become the content of the concrete message after the conversion. Use of variable is permitted in the message content specification. Lastly, channel specification is a description of a channel that should be used to send the concrete message.

The query defines the working dataset that the conversion engine will work with during the conversion process. This query is written in SPARQL (SPARQL Protocol and RDF Query Language), which is a standard query language for RDF data. The other elements describe the other three main parts of the message by referring to the data in the working dataset. Recipients of the message are defined as an enumeration of concrete customers through their URIs or

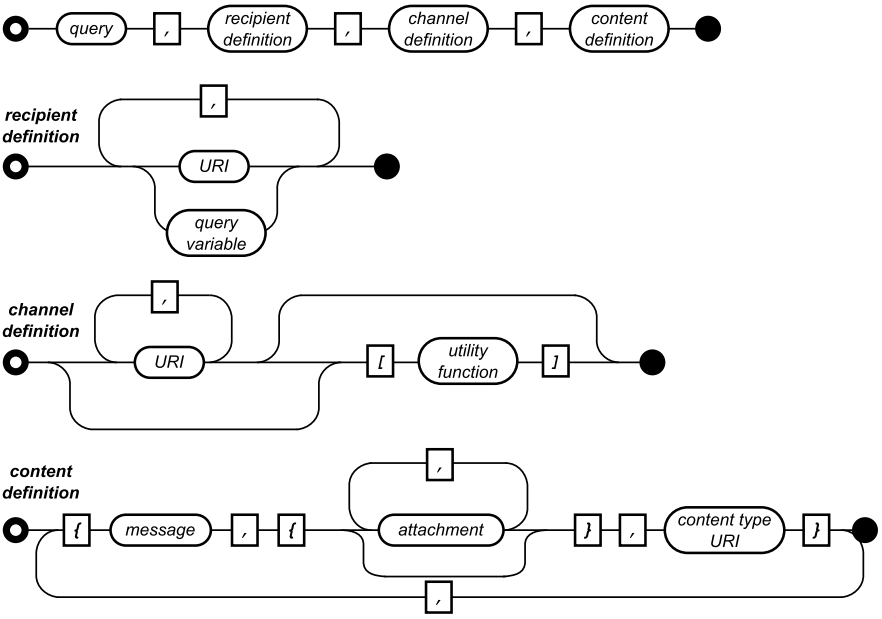

Fig. 6. Message template description language

variable names from the query. Message content can be a combination of textual messages, images, attachments, etc. Each message content specification is accompanied with a content type specification. These are used to determine which communication channel can be used to transfer them. For example if the message is a simple short text, it can be used as an email, SMS or tweet. However, if it is a text with images and some PDF document as an attachment, it can be specified as an email message only. Each abstract message can have several message content specifications with accompanying content type definitions. An example of a message template can be seen in Figure 7.

The query portion of the abstract message above describes a dataset that contains information about customers and products. The query is looking for customers that have bought a consumer electronics product after March 1st 2012. It also looks for the product name, product service file URL and product service file physical location. The query is used later in the message conversion process. The recipients of the message will be customers specified by a variable. The content of the message are two triples. The first triple contains some text with variables, attached service file and content type description specifying that it is an HTML text with an attachment. The second triple consists of some text, empty attachment set and content type description indicating that it is a short text. We will use this abstract message later to describe the message conversion process.

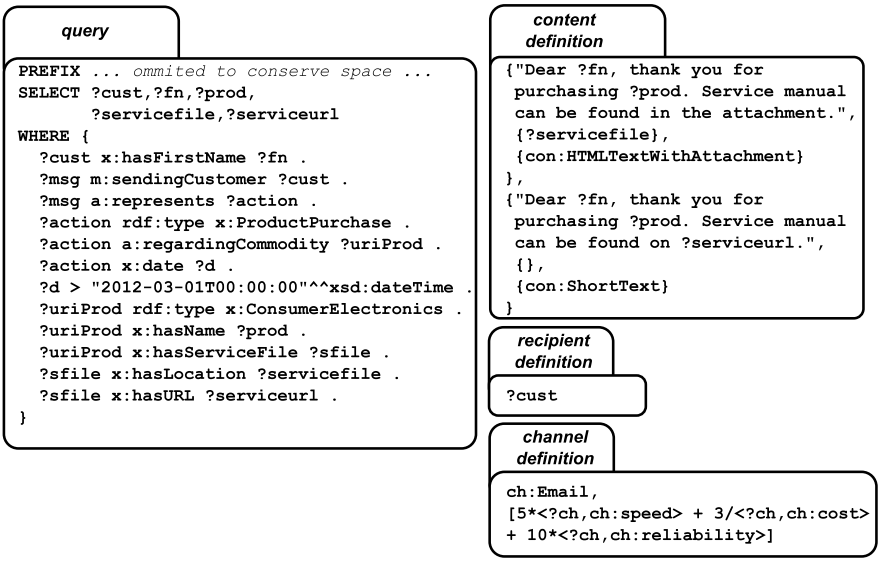

Fig. 7. An example of a message template

5.2 Message Conversion Process

The goal of the message conversion process is to produce a set of concrete messages by applying a message template on top of the contextual data. The message conversion process is described in a function called ConvertMessage (see Algorithm 1). It takes a message template and applies it on top of the data by performing the SPARQL query described in the query portion of the template.

The message conversion algorithm first performs a SPARQL query defined by the query portion of the abstract message definition. The result of the query is a binding table where each binding is represented by a row. A sample binding table is shown in Table 5. The sample binding table contains five variables representing the customer (?cust), customer's first name (?fn), product name (?prod), service file location (?servicefile) and URL of the service file on the producer's web page (?serviceurl). Then the algorithm iterates through all lines and for each line it determines the recipient and content. The channel definition is taken as it is provided in the message template. The channel selection process is described later in the chapter. The recipient is relatively straightforward to determine, since it is represented by a single variable. In the example from Figure 7 the recipient is represented by variable ?cust. The content is determined by substituting the variables in the template text with the values from the binding table. In the example from Figure 7 these variables are ?fn, ?prod, ?servicefile and ?serviceurl.

Recipient, channel, and content are three parameters that define an abstract message. Please note that in case of an abstract message the recipient is determined by a list of user URIs. Similarly, channel is determined by a list of

Algorithm 1. ConvertMessage

```
input : MsgTemplate msgtemp
TableRow[] rows = performSPARQLQuery (msgtemp.query);
ConcreteMessage[] concrMsgs;
foreach row in rows do
    URI recipient = row.getColumn (msgtemp.recipientVariable);
    ChannelDefinition[] channel = row.getColumn (msgtemp.channelVariable);
    ContentDescription[] content = msgtemp.contentDescription;
    concrMsgs.add(ConvertAbstractToConcrete (recipient, channel, content));
end
foreach cMsg in concrMsgs do
    SendConcreteMessage(cMsg);
end
```

Table 5. Sample binding table based on a fictional customer and product database

?cust	?fn	?prod	?servicefile	?serviceurl
u:id5	John	Sonic X52	/mans/X52.pdf	http://sonic.com/X52.pdf
u:id3	Jane	Jogman 2000	/mans/Jog2k.pdf	http://tony.com/J2000.pdf
u:id2	Bill	Pear uPhone	/mans/PuP.pdf	http://pear.com/uPhone.pdf

channel URIs or utility functions with resolved variables. The content is represented as a list of triples (message content, attachments, content type) with resolved variables. Thanks to the use of URIs, an abstract message can be linked to concrete users and channels. Figure 8 contains an example of an abstract message based on the first row of the binding table. In the example we can see that the first member of the content list is a triple with a text for customer named John, attachment containing one PDF file and marked as a content type con:HTMLTextWithAttachment. The second member of the content list contains just text and content type, but it does not contain any attachment. In the channel section we can see the same content as in the channel section of the message template. One channel was given statically and one is described using a utility function (see Figure 7). The channel selection process is explained later in the chapter.

However, an abstract message does not contain enough information for a message to be sent to a concrete user. Firstly, it is still lacking a concrete address (e.g. email address, phone number, social network account, etc.). Secondly, in case utility functions are used to specify the communication channels, it is also lacking information about the concrete channel. Lastly, the content is specified as a list of content triples describing various content types. Therefore it has to be converted it into a concrete message. A concrete message then contains a concrete address, concrete message text with concrete attachments and concrete channel that it should be sent to. The conversion from an abstract message to a concrete message is done by calling ConvertAbstractToConcrete function (see Algorithm 2).

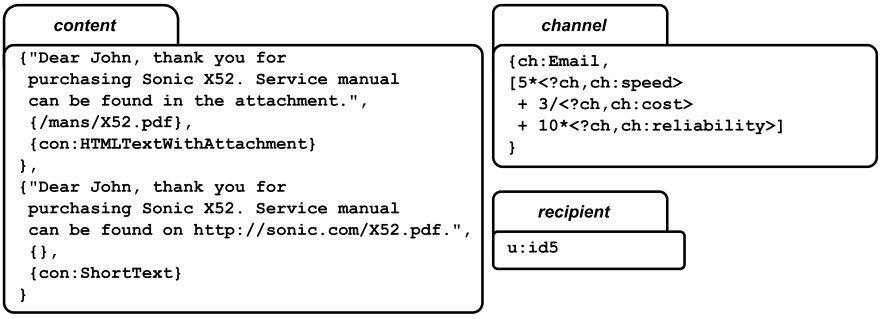

Fig. 8. An example of an abstract message

Algorithm 2. ConvertAbstractToConcrete

```
input  : URI customer,
         ChannelDefinition[] channelDefs,
         ContentDefinition[] contentDefs
output: ConcreteMessage[] concrMsgs
ConcreteMessage[] concrMsgs;
URI[] channels;
foreach chDef in channelDefs do
    switch typeOf (chDef) do
        case URI
            channels.add(chDef);
            break;
        case UtilityFunction
            channels.add(determineChannel(chDef, customer));
            break;
    endsw
end
foreach ch in channels do
    ChannelHandler chh = getChannelHandler(ch);
    foreach content in contentDefs do
        if channelAccepts (channel, content.contentType) then
            Message msg = chh.convertMessage(content.msg,
            content.attachmentList, content.contentType);
            Address addr = chh.extractAddress(content.user);
            concrMsgs.add(channel, addr, msg);
        end
    end
end
return concrMsgs;
```

ConvertAbstractToConcrete function takes an abstract message as the input and produces a set of concrete messages as the output. Please note that the three input parameters of ConvertAbstractToConcrete function correspond to recipient, channel, and content parameters of the abstract message mentioned above. The first step of the conversion is to obtain the channel handler based on the channel URI. The handler is capable of message content generation and extracting addresses from user descriptions. Then the function it iterates through the list of content triples and for each triple it checks if the particular channel accepts the particular content type from the triples. For example channel ch:SMS can accept content type con:ShortText, but it cannot accept content type con:HTMLwithAttachment. If the channel can accept the content type defined in the abstract message description, then a concrete message is created with the help of a channel handler. If the channel cannot accept the content type, the algorithm proceeds to the next content description triple. As a result a set of concrete message is created.

5.3 Channel Selection

The channel selection depends on the context data and the message template used to generate the message. From ConvertAbstractToConcrete function it is clear that the channel definition section of the message template may contain either a channel URI or a utility function. If a channel URI is found, it is simply added to the list channels that will be used to send the concrete message. However, if a utility function is found, the process is more complicated. In ConvertAbstractToConcrete function the process of utility function evaluation is hidden behind the determineChannel call.

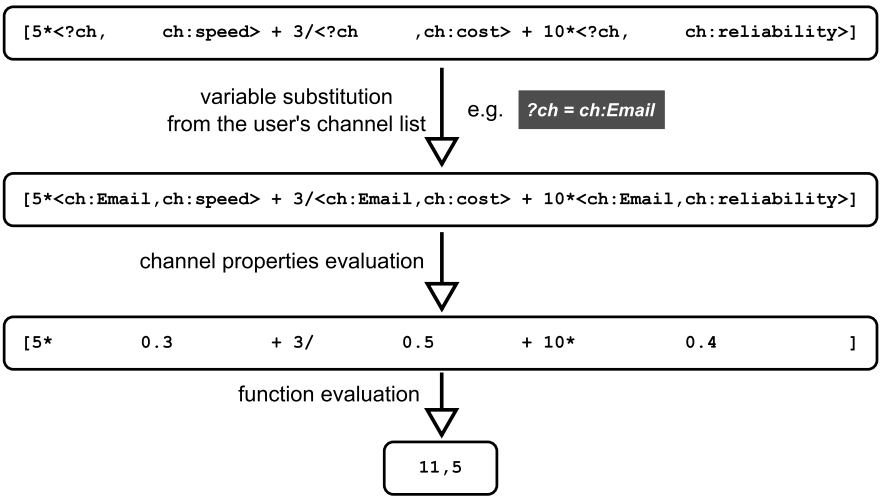

Fig. 9. An example of a utility function substitution and evaluation

The determineChannel function takes the recipient's URI and finds all channels that the recipient may use. Then, in an iteration each of them is used to substitute the ?ch variable in the utility function definition and the function is evaluated. The substitution and evaluation process is depicted in Figure 9. The <channel URI, property>pairs are replaced with values and the utility function is evaluated. The result is a real number representing the score of a given channel. Higher value represents more desirable channel. The channel with the highest value is considered the most suitable one and it is selected.

6 Conclusion

We presented a multi-channel communication framework capable of message integration from various communication channels. The framework consists of two main parts – knowledge base and message conversion engine. The framework's knowledge base is built on top of semantic technologies such as RDF, OWL, SPARQL, etc. The data is stored in form of RDF triples. The data schema is based on five main OWL ontologies. The commodity ontology describes products and services that the business is dealing with. The communication channel ontology is model for communication channels, channel handler and content types. The action ontology represents a variety of actions that the business or a customer can perform. The fourth ontology is the customer ontology dealing with customer's personal information, contact addresses and channel preferences. Lastly, the message ontology defines abstract and concrete messages and their relationships to other components. All the ontologies are extendable by the framework user in order to reflect the business needs.

The message conversion engine works with message templates that are being transformed into concrete messages. The message template is a prescription for a concrete message and it is defined by four parts – SPARQL query, recipient definition, channel definition and content definition. The engine performs the query and retrieves a working data set in form of a binding table. Then it chooses the best communication channel to send the message to. Lastly, it generates the message content based on the content specifications and communication channel's capability of accepting various content types.

Acknowledgements. The author would like to thank the industrial partner IPSS (Intelligent Precision Solutions and Services), members of IOG (Industrial Ontologies Group) and the TIVIT Cloud Software Program for supporting this work. Moreover, many thanks go to the reviewers for their helpful comments.

References

1. Berners-Lee, T.: Linked Data (2006), `http://www.w3.org/DesignIssues/LinkedData.html`
2. data.gov.uk, `http://data.gov.uk/`
3. DBpedia, `http://dbpedia.org`

4. Hendler, J.: Agents and the Semantic Web. IEEE Intelligent Systems 2, 30–37 (2001)
5. HermiT reasoner, http://hermit-reasoner.com/
6. Jena framework, http://jena.apache.org/
7. Nagy, M.: On the Problem of Multi-Channel Communication. In: Proceedings of ICTERI 2012, Kherson, Ukraine, pp. 128–133 (2012)
8. Protégé ontology editor, http://protege.stanford.edu/
9. RacerPro reasoner, http://www.racer-systems.com/
10. Resource Description Framework, http://www.w3.org/RDF/
11. Resource Description Framework Schema, http://www.w3.org/TR/rdf-schema/
12. Sesame framework, http://www.openrdf.org/
13. Terziyan, V., Kaykova, O.: From Linked Data and Business Intelligence to Executable Reality. International Journal on Advances in Intelligent Systems 5, 194–208 (2012)
14. Web Ontology Language, http://www.w3.org/TR/owl-features/
15. XML Schema, http://www.w3.org/XML/Schema