Enabling Email-Based Conversational Interface to Business Applications

Shefali Bhat
Tata Consultancy Services ltd, 249 D&E Udyog Vihar Phase IV, Gurgaon, Haryana, 122016, INDIA
and
C Anantaram
Tata Consultancy Services ltd, 249 D&E Udyog Vihar Phase IV, Gurgaon, Haryana, 122016, INDIA
and
Hemant Jain
Tata Consultancy Services ltd, 249 D&E Udyog Vihar Phase IV, Gurgaon, Haryana, 122016, INDIA

ABSTRACT

With email becoming the standard mode of communication among people, an email-based conversational interface to business application systems opens a new and convenient mode of interaction with such systems. Users should be able to send emails stating their requests to the system and the system should be able to respond appropriately. As the content of an email is usually in free-format text, meaningful information has to be extracted from it. Semantic web technology plays an important role in this process by enabling domain ontology creation, capturing domain-specific semantics and facilitating retrieval of answer to the email. We describe the architecture of an email-based natural language conversational interface, called Natas. We use semantic web-based ontology of the domain, to aid in the retrieval of relevant data and concepts from the system.

Keywords: conversational systems, natural language interface, user interface, semantic web, ontology, email.

1. INTRODUCTION

In today’s web-enabled world many transactions are being carried out through the internet. As a result a large number of end-users, from varied profiles (IT-professionals, IT-trained, IT-uninformed), log into a business application. With business applications becoming complex (e.g. banking companies offering banking and insurance services, or insurance companies offering insurance and health management services), users have to deal with various menus and options. This makes it complicated for some users, as they are unable to find the correct menus to get the data they want or carry out the task they desire. There is no facility to just send an email to a business application system and expect it to respond.

Email based interfaces can provide a mechanism for users of an application to interact with the system in a natural human language, such as English. Since emails are written in natural language and do not possess structure in their content (i.e. the body), an email-based interface would entail natural language processing and response mechanism in business applications. Natural language is one of many “interface styles” (or “interaction modalities”) that can be used in the dialog between a user and a computer. There is a significant appeal in being able to address a system and direct its operations by using the same language as we use in everyday human-to-human interaction. By implementing an email-based interface to business applications, the user interaction can move from just pushing buttons and dragging options, to specifying operations and assessing their effects through the use of language.

A framework called NATAS has been developed, that enables users to interact with a business application through emails by posing questions and invoking tasks. NATAS analyses, interprets and evaluates user input and responds back appropriately. The architecture of NATAS uses Semantic Web based ontology of the domain to aid in the retrieval of relevant data and concepts from the application. The idea for the application is to carry out a conversation with the user, in order to drill down to what the user actually wants and then to identify the task(s) that would carry out the user’s requirement. The natural language interface interprets the text and calls appropriate APIs of the application to accomplish the requested tasks. The main advantage of such a system is that the user is free to enter any information in a raw form. It is then the job of the system to process the raw information and get whatever else is required. Our approach in NATAS is based on an explicit domain ontology that is described using Semantic Web technology, namely RDF (Resource Description Framework), rules in N3 notation, OWL (Web Ontology Language), and SPARQL. The ontology describes the main concepts of the domain and their inter-relationships with a <subject-predicate-object> structure for each of the concepts. With the widespread nature of domain knowledge, Semantic Web technology allows seamless integration of different resource definitions that semantically mean the same thing. This permits easier integration of domain knowledge, which in turn makes the natural language system more robust in answering queries posed by the user.
2. STATE OF ART

Although a number of attempts have been made to build natural language interfaces for applications and databases [9], there has not been any attempt to have an intelligent e-mail-based workflow interface to the business application. **Answers Anywhere** by Sybase Inc [5] has been designed to have natural language enabled interfaces to a business application (wireless phone, a handheld PDA, a customized console, or a desktop computer). The method is based on agents and networks. The network structure defines the communication path between agents, which in turn determine the way agents get requests and provide responses. While their approach is good it doesn’t handle workflow of the business application, semantic description of web resources, traversal of the ontology graph or ontology-based querying and retrieval.

**Domain-specific languages** (DSL) have been created to simplify the specification of a system for a particular domain. A DSL can be viewed as a programming language dedicated to a particular domain or problem [2] [4], providing appropriate built-in abstractions and notations. Normally such languages are designed for a programmer and not for an end-user. A natural language interface, on the other hand, provides a mechanism to interact with an application in natural sentences. Some efforts for carrying out conversations in common-sense knowledge domains, such as ELIZA [3], Cyc project [1] and Open Mind Common Sense project [7] have been carried out. However, none of these address email-based workflow interfaces.

3. ISSUES

Business Application systems often deal with voluminous data and it is this data that forms the basis of domain ontology. While domain ontology helps drive meaningful conversation for answering user questions and carrying out various tasks, this also means handling issues related to its creation and processing. We list some of the main issues.

- **Ontology of the domain:** how can the domain ontology be created from the application data? How can it be seeded automatically?
- **Concepts of the domain:** what should be the concepts of the domain? How should they be represented?
- **Parsing the input:** What should be the mechanism of identifying query(s)/task(s) and in-turn the concepts implied by them from the content of the e-mail? In case the query posed contains pronoun(s), what methods should be used to resolve them?
- **Handling queries of the domain:** what should be the mechanism for resolving the queries posed on the domain?
- **Application specific tasks:** how should the NL interface handle application specific tasks – especially tasks that require detailed application logic?
- **Context resolution and clarification:** Often, the system cannot fully identify user’s intent, in such a scenario what should be the ways and means of clarifying the user’s intent.

In the sections below, we present an architecture that uses semantic web technology to help address these issues.

4. THE ARCHITECTURE

The architecture of our system relies on a domain ontology created using OWL, N3 and RDF technologies. The system uses this ontology to derive facts and reason on them in order to respond to the user’s email. Since the body of an e-mail does not posses any structure, the content of the mail has to undergo natural language processing.

![Architecture of NATAS](image)

**Fig. 1. Architecture of NATAS**

The query and tasks identified after parsing of the e-mail is sent to the synonym handler, which does synonym replacement. The output from the synonym handler is then tagged and the domain ontology is traversed to identify concepts in the input sentence. Next, these concepts are given to the Task Manager. The Task Manager decides to fire a SPARQL query, or call an API function, or let the system traverse the ontology to obtain the answer. The answer obtained is then sent back to the user by the e-mail formulator. Figure 1 shows the broad architecture of NATAS. Note that the arrows in the figure describe the information flow of the system. The application data can reside in any of the databases (SFleX, MySQL, or Oracle).

4.1 Ontology Creation

The ontology of the domain describes the domain terms and their relationships. The application data (i.e. the database of the business application) forms a part of the domain terms and their relationships in the ontology. This helps forms the main concepts of the domain and their relationships with a <subject-predicate-object> structure for each of the concepts. Figure 2 shows the ontology creation process and the levels of ontology.

The **Seed Ontology** describes the basic relations that are applicable in the domain, for example in a Project Management domain where details about all the employees, projects and the relation between the employees and the projects are handled; facts like “project has a project number”, “an employee is a person”, etc populate the Seed Ontology. The Application Data (also termed as static facts) provides the actual data that is present in the system. The **Ontology Generator** takes in the Seed Ontology and
Application Data and creates an instance of the Seed Ontology populated by the application data. This is called the Application Ontology. Next, the rules of the application domain are then evaluated together with the Application Ontology by a Rule Engine, such as Closed World Machine (CWM), to create the Domain Ontology. Domain Ontology is used by the NATAS system to answer questions on / carry out the tasks of the domain.

In our work, domain-specific rules are defined on the ontology in W3C’s N3 format to state possible derivable facts about data. For example, “has_boss” can be inferred from the rule as shown below:

```
{x ds:NAME ?a. ?x ds:Role
ds: Role ds:TeamMember. ?x ds:
project ?z. ?y ds:project ?z. =>
{?b ds:has_boss ?a ).
```

Thus, If x=1, a = Ritesh, y=2, b = Rajat, z = TechPrgm
Then Rajat has_boss Ritesh

The ontology is used in conjunction with the domain rules on the data and new facts are derived based on the data of the domain. These new facts are called derived facts. The static and derived facts are then converted into Resource Description Framework (RDF) format. The RDF format has each fact (both static and derived) as a set of URLs in XML form.

![Ontology Traversal](image)

**Fig. 2.** Overall architecture of ontology creation process.

### 4.2 Concepts of the Domain

The RDF file is read and a <subject-predicate-object> graph structure is created in the memory. Once we have the domain ontology in memory, we can traverse it using the graph traversal functions to get the subject, predicate or object (or a combination of these). A set of class objects is created in memory to represent each subject, predicate and object of the <subject-predicate-object> structure as concept of the domain. Each concept has the concept name and its synonyms to help identify the concept from the natural language sentence that the user inputs. The synonyms are derived by posing the concept name to WordNet - an open source lexical reference system [8]. In WordNet, English nouns, verbs, adjectives and adverbs are organized into synonym sets, each representing one underlying lexical concept. Different relations link the synonym sets. The relevant synonyms are loaded with each concept in memory.

### 4.3 Parsing The Input

The NATAS system fetches e-mails from the email spooler and sends it to the e-mail parser which reads the contents and extracts information, such as “from”, “to”, “date”, “time” “subject”, and “body”. In case an e-mail consists of multiple queries or tasks, the parser processes them one at a time. The input sentence is parsed for identifying the parts-of-speech in the sentence. This process identifies the nouns, verbs, adjectives and adverbs in the sentence. We use open source software, MontyLingua [6], for carrying out the tagging. Using these identified parts of speech, the relevant concepts of the domain that are referred in the sentence, are marked (called raised concepts). The raised concepts are then used to identify that part of the ontology, which needs to be traversed. From the <subject-predicate-object > tuples, the system tries to generate an answer for the user’s input. This answer is then presented to the user.

### 4.4 Handling Queries Of The Domain

The query posed by the user is answered in one of the following possible ways:

- **Firing of queries through SPARQL**
- **Ontology Traversal**
- **Retrieving answers through API**

The queries that are executed are written in a generic form in SPARQL [10]. Since the domain ontology is in RDF format, the general structure of the query is (subject, predicate, object). We have identified seven types of queries for the subject-predicate-object (henceforth referred to as <s-p-o>) structure of our ontology; these are: s (only subject); p (only predicate); o (only object); s-p (subject and predicate); s-o (subject and object); p-o (predicate and object); s-p-o (subject, predicate and object specified). The actual query is formulated by binding the value of the raised concept in the input sentence to the generic SPARQL query of one of the above seven types, in order to formulate the precise query and retrieve the answer.

For example, let the question be “In which project is Ritesh a group leader?”. The concepts raised for this particular query are, “Project”, “Ritesh” and “Group Leader”. Post concept identification, the system will try to fire one of the above seven mentioned SPARQL query. In the above example, since s, p, and o are known, therefore SPARQL query corresponding to <s-p-o> is fired (shown below).

```
Select={"?f"}
where=GraphPattern({"?a",d{prd1},
d{val1}},{"?b",d{prd2},d{val2}},
{"?a", "?c", "?a"}, {"?b", "?c" 
,"?a"}, {"?b", "?d", "?e"},{"?e", 
"?d", "?e"}, {"?e", d{prd}, "?f"})
result = sparqlGr.query (select, where)
```

Where val1=Ritesh and prd1=ename, val2 = Group Leader and prd2 = role and prd = pname

This query states that for an unknown subject “a” having “ename” as predicate and “Ritesh” as object, there is a “b” having “role” as predicate and “Group leader” as object, also subject “b” is bound to objects “a” and “e” with predicates “c” and “d” respectively, also there exists some “e” such that this “e” having “pname” as a predicate gives us the value for the project “f”.

In case the query generation does not fetch an answer then the system traverses the RDF graph. Ontology traversal takes in concepts identified from the input sentence and determines which part of the ontology these concepts satisfy. That is, the concepts could be leaf nodes or some intermediate nodes in the ontology graph. Once this is established, the traversal tries to determine the relationship (direct or inherited) between the concepts identified in...
the graph structure. When the application data is transformed into the Domain Ontology, a graph structure is created in the memory. It is this graph structure that is traversed in order to obtain an answer for the query. The graph traversal figures out the node which is directly or indirectly connecting two or more different nodes. Thus questions like “what is common between X and Y?” which are quite cumbersome to be answered by a query or through an API function, are answered easily through ontology traversal. For example, if a user wants to know “what is common between Anubha and Shefali”, the query generation mechanism is not going to give an answer, whereas an ontology traversal may give the answer “Anubha and Shefali are in the VirginAtlantic project”.

4.5 Handling Application Specific Tasks

In case both SPARQL as well as ontology traversal does not lead to an answer, the system tries to pose the concepts to the application. This requires tasks to be identified that can be carried out by the application system. For each possible task of the application, a task description file describes the concepts that are required to perform the task. From the concepts that are identified from the input sentence, the task that has the maximum number of raised concepts is carried out. The answers generated by the application system are then fed to a response manager, which formulates the appropriate response and sends it to the user in natural language. The context of previous interactions is used to identify the next set of concepts.

4.6 Context Resolution and Clarification

One of the issues is the need for context resolution and clarification. This need arises in case of ambiguous statements, such as “what is his allocation percentage?” The framework tries to relate the pronoun/determinant in the input sentence with the corresponding context and resolves it using gender-specific pronoun resolution or based on the previous answer in case the object doesn’t contain a gender. Concepts are identified from previous conversations and resolved with respect to the current questions. However, if the question posed by the user is ambiguous or the system is unable to resolve the pronoun, it responds back to the user through e-mail and seeks clarification. To understand this better, consider the following conversation between a user and an e-mail based application.

**User:** Who all are allocated to Technology Program?

**System:** Employees in TechnologyProgram are. Puneet

Shefali

Ritesh

**Rajat**

**User:** What are their roles?

**System:** The roles are

Puneet ProjectLeader

Shefali TeamMember

Ritesh ModuleLeader

Rajat TeamMember

**User:** What is his allocation percentage?

**System:** We are talking about 3 men 'Puneet', 'Ritesh' and 'Rajat' when you say 'his' whom do you mean? Your answer:

**User:** Puneet

**System:** Puneet’s allocation percentage is 25 percent

The moment the user clarifies the content, Puneet in this case, the system fetches the answer and replies back.

## 5. EXAMPLES OF IMPLEMENTATION

### 5.1 Project Management System

We consider a logical subset of the Project Management System for an organization. In this example the system consists of a number of tables containing data about the projects, the various costs associated with the projects, employees and their allocations in different projects. The important tables are named ProjectDetails, Employees, Cost Details and Allocations.

**Table 1. ProjectDetails**

<table>
<thead>
<tr>
<th>ProjectNumbe r</th>
<th>Name</th>
<th>...</th>
<th>Type</th>
<th>Status</th>
<th>Startdate</th>
<th>Enddate</th>
</tr>
</thead>
<tbody>
<tr>
<td>100582</td>
<td>TechPro</td>
<td>...</td>
<td>SWO</td>
<td>Active</td>
<td>01-01-05</td>
<td>31-12-05</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>200315</td>
<td>Bechel</td>
<td>...</td>
<td>WON</td>
<td>Create</td>
<td>10-10-05</td>
<td>31-12-06</td>
</tr>
</tbody>
</table>

**Table 2. CostDetails**

<table>
<thead>
<tr>
<th>Cost No</th>
<th>ProjectNo</th>
<th>...</th>
<th>Costing</th>
<th>Billing</th>
<th>Revenue</th>
<th>Realized</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>200234</td>
<td>...</td>
<td>25</td>
<td>25</td>
<td>75</td>
<td>20</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>10</td>
<td>100582</td>
<td>...</td>
<td>70</td>
<td>56</td>
<td>1000</td>
<td>60</td>
</tr>
</tbody>
</table>

**Table 3. Allocations**

<table>
<thead>
<tr>
<th>AllocNo</th>
<th>ProjectNo</th>
<th>...</th>
<th>EmpNo</th>
<th>Startdate</th>
<th>Enddate</th>
<th>Alloc %</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>100582</td>
<td>...</td>
<td>160</td>
<td>01-09-05</td>
<td>31-12-05</td>
<td>100</td>
<td>Team Member</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>100</td>
<td>100582</td>
<td>.</td>
<td>180</td>
<td>01-01-05</td>
<td>01-10-05</td>
<td>50</td>
<td>Project Leader</td>
</tr>
<tr>
<td>0</td>
<td>100582</td>
<td>.</td>
<td>180</td>
<td>01-01-05</td>
<td>01-10-05</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>

**Table 4. Employees**

<table>
<thead>
<tr>
<th>EmpNo</th>
<th>Name</th>
<th>Age</th>
<th>...</th>
<th>Gender</th>
</tr>
</thead>
<tbody>
<tr>
<td>160992</td>
<td>Shefali</td>
<td>24</td>
<td>...</td>
<td>Female</td>
</tr>
<tr>
<td>160784</td>
<td>Anubha</td>
<td>25</td>
<td>...</td>
<td>Female</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>107890</td>
<td>Rajat</td>
<td>31</td>
<td>...</td>
<td>Male</td>
</tr>
<tr>
<td>180180</td>
<td>Ritesh</td>
<td>28</td>
<td>...</td>
<td>Male</td>
</tr>
</tbody>
</table>

Let us assume that we want to find out, “What is the revenue of the project in which Ritesh is a Group Leader?” Clearly from the
above give data this query involves joins over four tables in order to extract answer. NATAS fetches answer for this query in the following manner. The domain ontology is loaded into the memory as RDF triples <subject, predicate, object>. An example of the domain ontology follows:

```
<rdf:Description
  rdf:about="http://Demo:Demo@172.21.107.145:8080/SfleXProd/#180180">
  <age rdf:resource="http://Demo:Demo@172.21.107.145:8080/SfleXProd/#27"/>
  <dob rdf:resource="http://Demo:Demo@172.21.107.145:8080/SfleXProd/#08-01 1979"/>
  <eName rdf:resource="http://Demo:Demo@172.21.107.145:8080/SfleXProd/#Ritesh"/>
  <empNo rdf:resource="http://Demo:Demo@172.21.107.145:8080/SfleXProd/#180180"/>
  <sex rdf:resource="http://Demo:Demo@172.21.107.145:8080/SfleXProd/#Male"/>
</rdf:Description>
```

The primary key form the subject, the fieldname forms the predicate whereas the values of the fields form the object in the ontology file.

**SPARQL Query**

One of the seven query templates is chosen. In this example it is:

```
select = ("?g")
where =
GraphPattern(
[ ("?a", ds[prd1], ds[val1]),
  ("?a", "?c", "?a"),
  ("?b", "?d", "?e"),
  ("?b", ds[prd2], ds[val2]),
  ("?b", "?d", "?e"),
  ("?e", "?d", "?e"),
  ("e", ds[prd1], ds[val1])
])
result = self.sparqlGr.query(select, where)
```

This query when fired fetches the appropriate answer. The deduction is as follows:

If `empno 180180 (a)` has `name (prd1) Ritesh (val1)` & `180180 (a)` has `empno (c)` & `180180 (a)` & Some allocation `100(b)` has `empno (c) 180180 (a)` & Same allocation `100(b)` has `projectno(d) 100582 (e)` & `Same allocation 100 (b)` has `role(prd2) Projectleader (val2)` & `Some cost No (f)` has `projectno (d) 100582 (e)` & `Same cost No (f)` has `revenue (prd) 1000 (g)`

Hence deducing: **The revenue is 1000**

**Domain Rules**

Domain specific rules are written over the ontology to describe the facts that can be derived from the given set of static facts. The rules operate on the ontology provided and generate an RDF structure containing facts as well as derived facts. With the help of these rules all the queries regarding the data can be answered.

A typical rule is as follows:

```
  ?x ds:role ds:TeamMember .
  ?x
```

This rule when instantiated for a data item would derive the appropriate answer.

Anubha has manager Ritesh.

### 5.2 Retail Management System

In case of retail outlet, which has a number of products, lots of promotion offers and catering to a large customer needs. Consider the following query posed by the customer

**User:** Which camcorders have more than 20% discount?

**Concepts Identified:** Camcorders, more than, 20%, discount

**SPARQL Fired:**

```
Select = ("?f")
where.addPatterns(
[ ("?a", "?c", "?a"),
  ("?a", ds[prd1], "?f"),
  ("?b", "?d", "?e"),
  ("?b", ds[prd2], ds[val2]),
  ("?e", "?d", "?e"),
  ("e", ds[prd1], ds[val1])
])
result = self.sparqlGr.query(select, where)
```

**System:** The Camcorders are

- DXG 3MP Digital Camcorder - DXG-301V
- Panasonic Mini DV Camcorder
- Aiptek IS-DV2 Digital Camcorder
- Panasonic 2.8" LCD Digital Camcorder with 3CCD Technology- Silver (SDR-S150).

Below is the screen shot of the above query

![Sample output](Fig. 3)
6. CONCLUSIONS

We have described e-mail-based conversational user-interface as an alternate mechanism of interaction with various business applications. This not only enables user to carry out dialog with the business application but also provides the user with the facility of conversing in a language that one uses in their day-to-day conversation. The Task manager simplifies updations in the database as well as ensures the consistency in the database. The use of semantic web ontology allows the natural language system to interact "intelligently" with the user by traversing the domain ontology for both static as well as derived facts. Semantic Web technology allows seamless integration of different resource definitions that semantically mean the same thing. This permits easier integration of domain knowledge, which in turn makes the natural language system more robust in answering queries posed by the user. Thus, having an e-mail-based interface to business application may be a good way to provide alternate mode of interaction with the application.

7. REFERENCES