AUTOMATIC GENERATION OF MULTILINGUAL SPORTS SUMMARIES

by

Fahim Hasan
B.Sc. in C.S.E., BRAC University (Bangladesh), 2007

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

In the
School of Computing Science

SIMON FRASER UNIVERSITY

Summer 2011

APPROVAL

Name: **Fahim Hasan**

Degree: **Master of Science**

Title of Thesis: **Automatic Generation of Multilingual Sports Summaries**

Examining Committee:

Chair: **Dr. Richard T. Vaughan**
Associate Professor – Computing Science

_____

Dr. Fred Popowich
Senior Supervisor
Professor – Computing Science

_____

Dr. Veronica Dahl
Supervisor
Professor – Computing Science

_____

Dr. Anoop Sarkar
Internal Examiner
Associate Professor – Computing Science

Date Defended/Approved: May 31, 2011

Declaration of
Partial Copyright Licence

ABSTRACT

Natural Language Generation is a subfield of Natural Language Processing, which is concerned with automatically creating human readable text from non-linguistic forms of information. A template-based approach to Natural Language Generation utilizes base formats for different types of sentences, which are subsequently transformed to create the final readable forms of the output. In this thesis, we investigate the suitability of a template-based approach to multilingual Natural Language Generation of sports summaries. We implement a system to generate English and Bangla summaries making use of a pipelined architecture to transform data in multiple stages. Additionally, we demonstrate how the automatically generated summaries differ from human generated summaries. We show that by using a template-based approach the system can generate acceptable output in multiple languages without requiring detailed grammatical knowledge, which is important for languages such as Bangla where computational resources are still scarce.

Keywords: Natural Language Processing; Natural Language Generation; Bangla; Template; Pipeline.

To my family

ACKNOWLEDGEMENTS

I would like to thank my supervisors, Dr. Fred Popowich and Dr. Veronica Dahl for their guidance and motivational comments on my work. I also thank my family and friends for their support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

Natural Language Generation (NLG) is a subfield of Natural Language Processing (NLP) that utilizes techniques from Artificial Intelligence (AI) and Computational Linguistics (CL) to automatically generate human understandable Natural Language (NL) text. The generated text can be formatted as reports, explanations, help messages, etc. from non-linguistic (structured) representation of information as the input [1]. NLG systems typically use knowledge about the target language and the application domain to produce their NL output.

The objective of our research is to explore cross-lingual NL summary generation in the sports domain. We investigate the applicability of various NLG techniques and propose a template based approach to automatically generate sports summaries in multiple languages. We present an implementation of the approach consisting of separate subsystems responsible for specific generation tasks. The implemented system produces natural language text as output from structured data in non-linguistic format as input. A key aspect of our approach is to have a language independent system, i.e., the system would support having template files for each language and not require modification to the core components when adding new languages. We also provide a methodology to demonstrate how human and computer generated summaries differ and discuss outputs of our system, displaying its ability to extract the key semantic concepts of the input data and successfully summarize those in NL sentences.

1.1 NLG Systems

From a broad perspective, NLG systems can be categorized as either standard systems that utilize generic linguistic and grammatical information or template based systems that map the input directly to the surface structure without requiring deep grammatical knowledge or in depth analyses [2]. NLG systems usually have several different subsystems with well-defined interfaces to each other that are responsible for specific subtasks of generating text. A widely used architecture for NLG systems has three primary components: the document planner, the micro planner and the surface realizer. The document planner is responsible for the tasks of content determination and document planning that specify the content and structure of the output text. The tasks of the micro planner are sentence aggregation and referring expression generation that determine which words and syntactic structures should be used to realize the content and structure chosen by the document planner. Finally, the surface realizer is responsible for linguistic realization, i.e. mapping the abstract representation created by the micro planner into actual text as the final output of the system [1].

This specific architecture described above is sometimes referred to as the pipelined architecture since the different modules of the system are connected to each other in a one-way pipeline. That is, the output of the document planner acts as the input to the next module, which is the micro planner. In the same way, the output of the micro planner acts as the input to the surface realizer. The process is called one-way since a module in the pipeline can only have an

influence on the operation of the modules that come after it. For example, since the micro planner is placed after the document planner and before the surface realizer, it can only control the outcome of the surface realizer since it produces the input for the realizer, and not the document planner. Graphically, the pipeline can be represented as shown in Figure 1.1, which is based on the depiction provided by Reiter and Dale in [1].

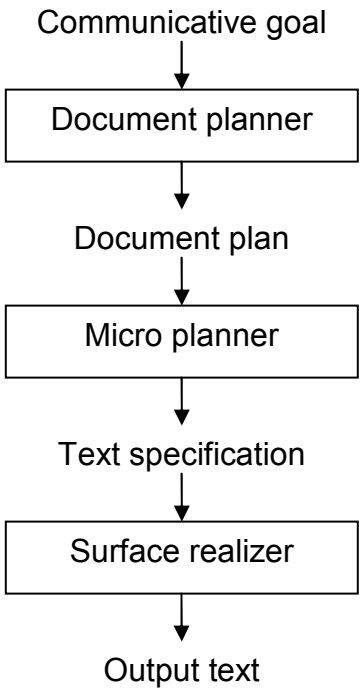

Figure 1.1: The pipelined architecture of NLG systems

1.2 Motivation

For our research, we chose to focus on generation in two languages: English and Bangla. As for the domain within sports, we decided to use Cricket, since it is one of the most popular sports in the Commonwealth.

Bangla is the sixth most spoken language in the world [3], but work on Bangla NLP is still not widespread compared to other languages such as English. In fact, while studying the related research papers for this work, we found few works on Bangla NLG [4, 5, 6] (except within Machine Translation systems). And these were also quite different than our target area since [4] focuses on morphological synthesis of word classes; [5] discusses sentence fusion techniques by identifying clauses and types of input sentences and [6] investigates appropriate discourse marker generation and aggregation using grammatical knowledge, which makes our system the first of its kind since Bangla is one of the target languages of the system. Although we will focus on Bangla generation in one specific domain, we will consider situations that may be useful or adaptable to other domains as well.

As input to an NLG system, we considered medical records of patients, weather forecast data and game score cards, all of which are widely available as structured data and also allow us to build on the ideas used by some of the previous research in the area which will be examined in Chapter 2. We decided upon game data because of its wide availability and more standardly structured nature, compared to medical records or raw weather forecast data. Also, human written reports are available alongside the actual game data, which can be utilized for demonstrating the output differences of the generation system and humans to some extent. We specifically selected score cards of Cricket games as the input data to our system due to its structured nature and vast availability in similar formats across different sources. A second reason for selecting Cricket

score cards is the availability of domain knowledge. As mentioned previously, NLG systems utilize domain knowledge or language specific information such as grammars to generate output text. Since the proposed system would not have access to sophisticated grammatical resources, proper utilization of domain knowledge, which we are already familiar with in the Cricket domain, holds significant importance for producing meaningful output. Furthermore, since the output of the proposed system would be fairly simple sentences, it would also be possible to use the output text as input to a Text to Speech (TTS) system in the future. This has been done for several applied NLG systems previously [7, 8, 9]. Furthermore, a popular Bangla newspaper is already using TTS technology to provide spoken version of its news through the Internet [10] and the proposed system could be used alongside that to create audio bulletins of game results.

1.3 Contributions

The research in this paper will advance both the state of the art with respect to generation of sports summaries, and the work being done in Bangla language processing.

We present a template based cross-lingual approach to NLG utilizing the pipelined architecture, which is widely used to build generation systems due to its simplicity, flexibility, high cohesion and low coupling. The system is capable of creating short game summaries of one-day international (ODI) Cricket matches in Bangla and English from game scorecards in non-linguistic form.

We verify the hypothesis that a template based approach to NLG facilitates a decreased language dependency. We show that by utilizing templates to design the system, it is possible to generate output in multiple languages by only creating new templates and without requiring modifications to the core components of the system.

Our system also demonstrates the fact that using templates allows generation, even though in a constrained manner, without requiring embedding of detailed grammatical knowledge within the system. Removing the need for detailed grammatical knowledge is important for languages like Bangla where computational resources are not widely available yet.

Another contribution of the presented work is that we discuss a methodology to assess the performance of a generation system. We describe how the output of an automatically generated summary could be compared meaningfully to a human authored game report for the Cricket domain. Based on the discussion, we present comparison results demonstrating that the system accurately extracts and realizes the key semantic concepts, i.e. the most important information from the input data as compared to human authored reports.

1.4 Organization

The rest of this thesis is organized as follows. In chapter 2, we discuss the different approaches to NLG and some of the previous work on generation tasks such as creating textual summaries of ontology concepts, software engineering

test cases written in Z, weather forecasts and soccer game results. In chapter 3, we present our NLG system and discuss in detail the different aspects of the system such as the general architecture and module specifics of content determination, aggregation and cross-lingual surface realization. In chapter 4, we discuss why evaluation in general is a difficult task for NLG and as such, present the method that we followed to demonstrate the performance of our system. We summarize the presented work in chapter 5 and subsequently provide our conclusions.

CHAPTER 2: RELATED WORK

In this chapter, we discuss previous approaches to NLG related to our research, along with their benefits and drawbacks, and then discuss some applied NLG systems. We examine their features, evaluation methods and how the underlying ideas used to build those systems motivate the design of our NLG system.

2.1 Shallow versus In-Depth Generation Techniques

According to Buseman and Horacek in [11], language generation techniques can be categorized as either shallow or in-depth. An in-depth approach is knowledge based and theoretically motivated whereas a shallow approach is opportunistic and only models parts of the language that are necessary for the task at hand. As stated by Reiter et al. in [12], a significant amount of domain or language related background knowledge is required for NLG systems to produce quality output comparable to that written by humans. However, the process of knowledge acquisition for NLG tasks is by no means easy due to the complexity, novelty, poorly understood nature and ambiguity of the task. Therefore, since the shallow NLG methods can vastly differ in complexity depending on the requirement of the generation task, they are appropriate when the in-depth methods are not well understood or are less efficient [11].

Based on the above reasoning, Buseman and Horacek present a *flexible shallow method* called the TEMSIS (Trans-national Environmental Management Support and Information System) application for automatically generating reports on the quality of air [11]. According to the authors, the approach requires little linguistic information and can be adapted to new domains easily. It uses templates, canned text as well as a detailed grammatical model for realizing sentences. There are two components, the text organizer and the text realizer in the application as well as an internal representation of information that is realized in a language independent manner. The advantages of such an approach are that it allows reusability of the modules, offers language-modelling flexibility and has better processing speed than in-depth methods since processing complex grammatical rules is not necessary. In addition, there is no language dependency, which allows porting to a new language without requiring considerable effort. Therefore, for the cross-lingual summary generation problem, it suggests that the use of a template based approach might be a good idea since it allows language flexibility and does not require grammars. However, the authors also acknowledge the shortcomings of shallow approaches, i.e. domain dependency of the realizer and internal representation, which prohibits reuse of the internal representation in a new domain without requiring modification. Since for our generation task, we focussed on reducing language dependency rather than domain dependency, these drawbacks did not pose significant problems. The authors also present a qualitative evaluation of the method by discussing its

advantages and disadvantages. However, evaluation of the actual generated output is not included in the paper.

2.2 Template Based Approaches

Buseman and Horacek state that the main difference between template based and non-template based approaches is that the non-template based methods are linguistically motivated and utilize a layer of internal representation of information that is used by the surface realization component [11]. They argue that the advantages of both the linguistically motivated and the template-based approaches are limited and the former method is generally difficult to use whereas the latter is too inflexible. However, as will be seen in some of the systems discussed later, several template based approaches also utilize internal representation of information.

Cristiá and Plüss [13] present a prototype NLG system aimed at creating NL description of test cases written in a logical format called Z, created by Model Based Testing (MBT) tools. As displayed in Figures 2.1 and 2.2, and stated by the authors, test cases written in Z have to be converted to NL descriptions in order for humans to be able to understand what operation is described in the test case.

$$
\begin{aligned}
&\mathtt{processingTC} = \mathtt{yes} \land \mathtt{adds} = \varnothing \\
&\mathtt{blocks} = \{\mathtt{mid0} \mapsto \{1 \mapsto \mathtt{byte0}, 2 \mapsto \mathtt{byte1}, \\
&3 \mapsto \mathtt{byte2}, 4 \mapsto \mathtt{byte3}\}\} \\
&\mathtt{m?} = \mathtt{mid0} \land \mathtt{sa?} = \langle 1 \rangle \land \mathtt{len?} = \langle 2 \rangle
\end{aligned}
$$

Figure 2.1: An example test case in Z

Service (6,5) will be tested in a situation that verifies that:

- the state is such that:
  - the on-board system is currently processing a telecommand and has not answered it yet.
  - the service type of the telecommand is DMAA.
  - the set of sequences of available memory cells contains only one sequence, associated to a memory ID, which has four different bytes.
  - the set of starting addresses of the chunks of memory that have been requested by the ground is empty.
- the input memory ID to be dumped is the available memory ID, the input set of start addresses of the memory regions to be dumped is the unitary sequence composed of 1, the set of numbers of memory cells to be dumped is the unitary sequence composed of 2.

Figure 2.2: The corresponding NL description

As the authors describe, since the Z test cases are essentially a set of bindings between variables and values, a template based approach to generation is suitable for creating NL descriptions where a grammar is defined to describe templates for each test case. These templates are called NLTCT (Natural Language Test Case Templates) and each template defines how a specific test case in Z is converted to an NL description by including a parameterized description of the test case. This idea was utilized to some extent in our generation task, where the templates can be considered as parameterized descriptions of specific type of sentences and define how they should be realized in natural language.

The authors state that the parser they implemented that takes an NLTCT and a Z test case as input produces an adequate NL description of the test case. However, a limitation of the approach is that the solution is domain dependent and as such, would not generalize well to specifications written for other

domains. Also, it requires writing a template for each task, which might be difficult for testing complex systems with many test cases.

Reiter et al. [14] describe a template based system called IDAS (Intelligent Documentation Advisory System) that generates advanced help messages using canned text, hyperlinks and Object Oriented techniques. The system can produce answers to questions such as "*What is it*", "*Where is it*", "*What are its parts*" about the objects in the knowledge base. The knowledge base is used to store information such as domain knowledge, grammatical and content determination rules and words. The system is also used to generate technical documentation of automatic test equipment using information from the knowledge base. As the authors report, the system is successfully extended for multi-lingual support, which supports the hypothesis that a template based approach might be suitable for cross-lingual generation tasks. For evaluating the performance of the system, a user effectiveness study is done using three human subjects where the subjects are asked to answer several questions using the information obtained from the system and then fill out a questionnaire about its performance. Thus, according to the authors, the results despite being positive in general, should be considered suggestive and not statistically significant. However, as we will find later in this chapter, this kind of evaluation technique is often utilized when other methods may not be available.

In [7, 15, 16, 17], Wilcock discusses a pipelined approach to NLG that generates text later used as input for a speech generation system. The discussed method uses an XML based pipeline and XSLT templates for

transforming text plan trees (where the leaves are domain specific concept messages) to text specification trees (where the leaves are linguistic phrases). Their approach is similar to ours in its use of templates, the internal representation format, which is a set of concepts or facts, a transformation based scheme that modifies the input data in different stages and finally a tree based pipeline implemented in Java. A short example of how the system works, as described in the paper is included below.

Assuming the input question is "*Which bus goes to Miami?"* the system is supposed to provide the answer "*Number 74*". The input to the generation system is called an agenda, which is a set of concepts for realization as determined by a separate component called the dialogue manager. The agenda consists of the following XML description:

```
<agenda id="1">
  <concept info="Topic">
   <type>transportation</type>
   <value>bus</value>
  </concept>
  <concept info="Topic">
   <type>destination</type>
   <value>Malmi</value>
  </concept>
  <concept info="Topic">
   <type>bus</type>
   <value>exists</value>
  </concept>
  <concept info="NewInfo">
   <type>busnumber</type>
   <value>74</value>
  </concept>
</agenda>
```

Figure 2.3: An input agenda

Using the agenda as input, the system performs content determination by extracting the concept nodes and creating a text plan. It also determines whether to generate the information as new information which has not been mentioned previously, i.e. a *NewInfo* or whether to wrap it using a link to a previous topic. The resulting text plan in XML is displayed below.

```
<TextPlan id="1">
  <Message>
   <type>NumMsg</type>
   <concept info="NewInfo">
     <type>busnumber</type>
     <value>74</value>
   </concept>
  </Message>
</TextPlan>
```

Figure 2.4: A text plan tree in XML

During the micro-planning stage, the text plan tree displayed above is transformed using XSLT templates to a text specification tree where the messages in the concept nodes are changed to phrase specifications and domain concepts are transformed into referring expressions, if necessary. The resulting text specification tree is provided in XML as shown in Figure 2.5.

```
<TextSpec id="1">
  <PhraseSpec>
   <subject cat="NP">
     <head>number</head>
     <attribute>74</attribute>
   </subject>
  </PhraseSpec>
</TextSpec>
```

Figure 2.5: A text specification tree

Finally, the realization stage processes the text specification tree as displayed in Figure 2.6, and creates output in Java Speech Markup Language (JSML) to be used as input data for the speech generation system.

```
<jsml lang="en">
  <div type="sent"> Number
    <sayas class="number">74</sayas>
  </div>
</jsml>
```

Figure 2.6: Output text in JSML

As stated, the author considers the approach not to be a template based generation method, but a system combining template based text planning and transformation based micro-planning. Even though the output of the method is reasonable, a formal evaluation of the system is not presented in the papers. However, they do provide clear examples of how the different stages of the pipeline of an NLG system could be designed in order to effectively perform the generation task.

As described in [18], Web Ontology Language (OWL) is a family of knowledge representation languages for authoring ontologies. Galanis and Androutsopoulos [19] present an XML based verbalizer for OWL ontologies implemented in Java, called the NaturalOWL system. NaturalOWL follows a pipelined architecture where generation is carried out in three stages. Content selection and document structuring, i.e. determination of the order of information to be realized, are done in the first stage called document planning. In the second stage known as micro-planning, an abstract sentence specification is created for each message and these are aggregated as necessary. Referring

expression generation is also done during this stage. Finally, in the surface realization stage, text output is created from the abstract sentence specifications. The system is similar to the one presented in [17], with the difference being instead of using XSLT templates for transformation, NaturalOWL uses Java for the same purpose. This provides clear separation of processing code with linguistic resources and also allows easier transformation of data since XML trees can be modified easily using available methods in Java rather than writing custom XSLT templates.

White and Caldwell discuss an object oriented and rule based framework named EXEMPLARS for Natural Language Generation in [20]. The system focuses on ease of use, extensibility and efficiency using schema like text planning rules that are called exemplars. These exemplars are used to determine the content and form of the generated text by using a condition and an action. The condition specifies when the rule should be applied given the state of the input and the discourse context whereas the action specifies what should be added to the output and how the discourse should be updated if the condition is satisfied. As a concrete example, application of the *IdentifyDate* exemplar from the paper is included below that generates a state of a task given its start and end dates. If the start date is same as the end date, the phrase *"same day"* is used instead of mentioning the date twice as displayed below in (2.1).

```
This task is scheduled to start next Thursday, June 25,    (2.1)
and to finish the same day.
```

The idea of conditionally applying exemplars or rules on parts of the template is utilized in our system, where the surface realizer determines whether to apply a phrase template (action) depending on the features selected by the content selector for realization (condition) that correspond to that template. The idea of tracking the discourse context and updating it based on what information is added to the output is also utilized.

A template based generation system called XtraGen is presented by Stenzhorn in [21]. As the author describes, XtraGen is implemented in Java and uses XML based interfacing for easy integration into real world applications and scenarios. XtraGen uses grammar templates for the generation task, an example of which is included in Figure 2.7.

```
<template id="String"
      category="String">
  <conditions>
   Condition*
  </conditions>
  <parameters>
   Parameter*
  </parameters>
  <actions>
   Action+
  </actions>
  <constraints>
   Constraint*
  </constraints>
</template>
```

Figure 2.7: A grammar template of XtraGen

The condition can be one or more in number that defines when a specific template can be applied. It can also be simple or complex, i.e. a combination of

several conditions connected by *And*, *Or, Not* phrases. As the author describes, the parameters are used to specify a preference of style or rhetorical structure for generation. For example, the template in Figure 2.8 is preferred when generating text for the expert level users as specified by the parameter named level.

```
<template id="explainExpert"
      category="explain">
 <parameters>
   <parameter name="level"
      value="expert">
   <parameter name="verbosity"
      value="low">
 </parameters>
 ...
</template>
```

Figure 2.8: Use of parameters in XtraGen

The actions specify what should be generated, e.g. static or inflected text when a condition of a template is fulfilled. Finally, the constraints are used to apply morphological rules on the generated output. An example of the output produced by XtraGen is included in (2.2).

```
The number of documents is 37, divided into 2 different
categories. The results have been produced using 3 fold-
cross-validation which means that the data-set is divided
into 1/3 test-set and 2/3 training-set.
```
(2.2)

As described previously, this condition-action styled transformation technique, albeit in a simpler manner than discussed in the paper is utilized for realizing the templates in our system. As stated by Stenzhorn, XtraGen has been evaluated by successfully integrating it into an existing system called X-Booster, which is a binary classifier. XtraGen is used to generate natural language text to

explain the learning phase of X-Booster in multiple languages such as English, German and French.

McRoy et al. discuss an approach called YAG (Yet Another Generator) in [22] to improve template based text generation by increasing the flexibility of the templates. According to the authors, this can be achieved by augmenting the templates with linguistic or grammatical information and thereby making them more flexible and reusable across different applications than traditional templates. The advantages of using an augmented template based approach are speed and robustness, i.e. the ability to realize even with errors such as subject-verb disagreement in the input, and coverage, which is the ability to realize any kind of sentence if an appropriate template exists. A simplified example of a template for the output text "*John walks*", as described in the paper is included in Figure 2.9. In this example, *John* is evaluated as the *agent*, *walking* as the *process* and the subject is *null.*

```
((EVAL agent)
 (TEMPLATE verb-form
  ((process ^process)
   (person (agent person))
   (number (agent number))
   (gender (agent gender))) )
 (EVAL object)
 (PUNC "." left) )
```

Figure 2.9: Template rule for the *clause* template

Although we did not directly use the augmentation technique described in the paper during the current stage of development of our system, we designed the templates in such a way so that augmentation would be possible without

requiring major changes to the system as discussed in the future research directions section in chapter 5.

Theune et al. discuss the GoalGetter system in [8], based on a previous system called D2S (Data to Speech) that combines language and speech generation techniques. GoalGetter generates spoken reports for football matches in Dutch. For the text generation part, as the authors describe, it utilizes syntactically enriched templates along with knowledge about the discourse context. A major difference of GoalGetter with most other NLG systems is that it does not use a pipelined architecture as can be seen in Figure 2.10.

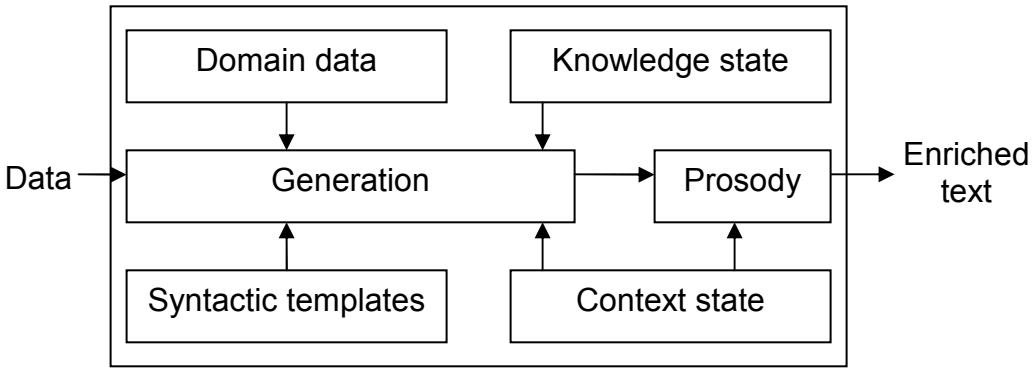

Figure 2.10: System architecture of GoalGetter

In GoalGetter, the generation module contains the algorithms for creating text that are applied to the input data using the syntactic templates. The syntactic templates are tree structures that contain variable slots, which can be filled up by appropriate values from the input data. The domain data contains background knowledge about teams and players, e.g. nationality and play position that are used to enhance the output quality by adding variation to it. The knowledge state stores which part of the input has already been processed and considered to be

known to the user, whereas the context state stores the current context and discourse, i.e. which objects have been mentioned to be used for referring expression generation. Some of these ideas from GoalGetter, such as the use of domain data and context state in order to increase variation have been utilized in designing our system. A sample output of GoalGetter is displayed in Figure 2.11.

```
Go Ahead Eagles / visited Fortuna Sittard // and drew ///
The duel ended in two // - all ///
Four thousand five hundred spectators / came to 'de Baandert'
///
<new-par>
The team from Sittard / took the lead after seventeen minutes
/ through a goal by Hamming ///
One minute later / Schenning from Go Ahead Eagles / equalised
the score ///
After forty-eight minutes / the forward Hamming / had his
second goal noted ///
In the sixty-fifth minute / the Go Ahead Eagles player
Decheiver brought the final score to
two // - all ///
<new-par>
The match was officiated by referee Uilenberg ///
He did not issue any red cards ///
Marbus of Go Ahead Eagles / picked up a yellow card ///
```

Figure 2.11: Output of the GoalGetter system

From the above discussions, it can be stated that in general, the templates in NLG tasks can be considered to be a collection of condition-action rules that specify when a parameter of the template (which may itself be a template) can be applied for realizing some property in the input data. And this is the definition that we use in designing the templates for our system where a sentence template is a collection of phrase templates where the applicability of the sentence template is determined by the type of the concept being realized. Similarly, the applicability of the parameters of the sentence templates, i.e. the phrase templates is

determined by the realization features specified by the content selection component.

2.3 Other Approaches

Bontcheva and Wilks present a system called MIAKT (Medical Imaging and Advanced Knowledge Technologies) in [23] for automatically generating documentation from ontology concepts encoded using the semantic web standard. The proposed method is not template-based and according to the authors, it provides better output standard than template-based methods and uses information from the ontology in different stages of generation. This claim by the authors is in line with the notion discussed by Deemter et al. in [2] that NLG approaches not based on templates are in general superior to their template based counterparts since they provide variation, better output quality and are well-founded based on linguistic theories. However, as discussed by the authors, and also apparent from the examples in the previous section, template based methods can indeed afford variation in the output and are also able to use linguistic knowledge, which makes this kind of distinction between the two approaches increasingly blurred [2].

For the MIAKT project, the input is medical data of patients encoded in semantic web standards, specifically the medical ontology, description of a medical case in RDF [24], and the MIAKT lexicon. The job of the generator is to generate textual descriptions given the semantic input. The lexicon is custom built to provide mapping of medical terms to ontology concepts and is similar to the lexicons used in our system that contain a mapping of player names and

other language specific constructs from their internal representation format to the corresponding surface forms in the different languages.

It is worth mentioning that the realizer in the MIAKT system is not template based since it does not use a fixed structure where arguments are inserted to realize the surface format. Instead, as the authors describe, the input is an RDF statement and a concept which will be the subject of the output sentence. The realizer treats the RDF statement as a graph. The input concept is considered as the starting point and a path is traced through the RDF visiting all properties and their arguments.

In the paper, the authors describe the evaluation process of the robustness of the system, i.e. how it responds to and handles errors in the input. Plans of doing a qualitative evaluation are mentioned but not included in the paper which seems interesting considering the previous claim by the authors that non-template based approaches provide better output quality than template based methods.

Another approach to generate descriptions from RDF representations is discussed by Sun and Mellish in [25]. According to the authors, the presented method is domain independent and does not require a lexicon. However, the quality of output text depends on the amount of linguistic information in the RDF data and also its structure.

The approach takes a group of RDF triplets that represent a connected RDF graph as input, determines the connections between them based on common values and outputs the description in natural language. An example

input and the corresponding output text is provided in Figures 2.12 and 2.13 respectively.

```
RDF triples
( LongridgeMerlot, RDF:type, Merlot)
( LongridgeMerlot, locatedIn, NewZealandRegion)
( LongridgeMerlot, hasMaker,Longridge)
( LongridgeMerlot, hasSugar, Dry)
( LongridgeMerlot, hasFlavor, Moderate)
( LongridgeMerlot, hasBody, Light)
```

Figure 2.12: An example RDF triplet

```
LongridgeMerlot is a kind of Merlot with dry
sugar, moderate flavour and light body. It is
located in the New Zealand Region and it has
maker Longridge.
```

Figure 2.13: The corresponding output text

It has been stated by the authors that evaluation of the method is not carried out and included in the paper due to the fact that the output quality depends upon numerous factors such as the generation process as well as the conversion quality of an input RDF graph for processing. However, a blackbox evaluation technique that checks the output of the system as a whole could be utilized in such cases.

A stand-alone surface realizer for natural language generation called RealPro is discussed by Lavoie and Rambow in [26]. According to the authors, RealPro offers several advantages to generation such as the ability to run as a separate server and also provides APIs in multiple languages. The input format of RealPro is an unordered tree called a deep syntactic structure and is human readable. The linguistic resources that it uses such as grammar and lexicon are

stored in ASCII formatted text files and hence can be easily modified. However, RealPro does not provide grammars for languages other than English and French, which makes it difficult to use with languages such as Bangla where developing a grammar might be a problem due to lack of resources. Also, the ASCII formatted files might be troublesome for storing data of some languages that require extended encoding formats such as Unicode support.

Hewlett et al. present a domain independent approach to generate descriptions of concepts defined in OWL ontologies in [27]. The system generates a parse tree structure of the input representing the connections between a given class in the ontology and other entities. This tree is then traversed starting from the root to create output sentences. The discussed method is different from [19] where the system supports multiple languages and utilizes templates for simplicity instead of creating parse trees. The difference with [25] is that the method discussed there generates output from RDF input data, whereas the current approach is capable of handling OWL, which is significantly more complex than RDF. An example output is provided in (2.3).

```
A  Beaujolais is a Wine that:                                    (2.3)
is made from at most 1 grape, which is Gamay Grape
   • has Delicate flavor
   • has Dry sugar
   • has Red color
   • has Light body
```

The authors mention that even though the approach is domain independent, it uses a Part of Speech (PoS) tagger to determine the word class of a property, which is not very flexible and has language dependency. Evaluation of the approach is described in the paper where the output of the

system is compared to that of two other systems by five users who always selected the output of the presented system for being better than the others based on correctness, readability and clarity. However, it is not mentioned how these factors are measured, which can vary considerably between subjects.

In [28], Bontcheva discusses the ONTOSUM system that generates textual summaries from Semantic Web Ontologies. The approach, which does not utilize templates, is an extended version of the work presented in [23], i.e. the MIAKT system and uses customized versions of existing set of tools to generate the output text. ONTOSUM follows a pipelined architecture where the input consists of a set of triplets from the ontology in RDF or OWL format. The input is first pre-processed to remove repetition of information. Next, ordering and aggregation schemes are applied on the input statements by the summary structuring module. In the following stage, the data are converted into graphs and provided to the realizer for generating output text.

Demir et al. present an approach for generating textual descriptions of bar-chart graphics using a bottom up approach in [29]. The system takes the bar-chart data, i.e. the number of bars and height of each bar as an XML description. It then applies several manually crafted content selection rules on the graph to determine the contents of the message conveyed by the graph. An example of a rule from the paper is provided in (2.4).

```
If (message category equals increasing trend) then          (2.4)
include (propositions conveying the rate of increase of
the trend).
```

This idea of developing content determination rules based on data analysis is utilized in developing the content selection module of our system as discussed in chapter 3 where we present a set of content selection rules created by analyzing raw game data and the corresponding human authored news bulletins.

The proposed system also aggregates the selected propositions for realization and utilizes the SURGE [30] realizer for generating the final output text. Evaluation of the system is done by generating summaries using different combinations of system parameters. Then 15 participants are each given 4 summaries to rank according to the quality of conveying the message of the input graph to the output summary. It is reported that the output generated using the settings the authors used, i.e. using ordering and aggregation rules, and selecting the output rated highest by the evaluation metric is chosen most of the time (65.6%) by the evaluators to be the best generated summary.

Sripada et al. present a hypothesis on summary generation in [31]. Based on their Knowledge Acquisition (KA) experiments on several projects, the authors state that during the content determination stage of creating summaries, humans first form a qualitative mental overview of the input data. Then the overview, along with the input data itself is used to carry out content determination. However, all the information present in the overview may or may not be used in the actual output summary. This hypothesis is utilized in the text generation system following a pipelined architecture as discussed by the authors in [32] for generating marine weather forecasts for oilrigs. The system, called SumTime-

Mousam generates textual weather forecasts in four steps given time series data of weather prediction patterns as input. The output generated by the system is used to help form the mental overview and is post edited by human authors before passing on to the end-users. The architectural diagram from [32], along with brief information on how the system works is given below.

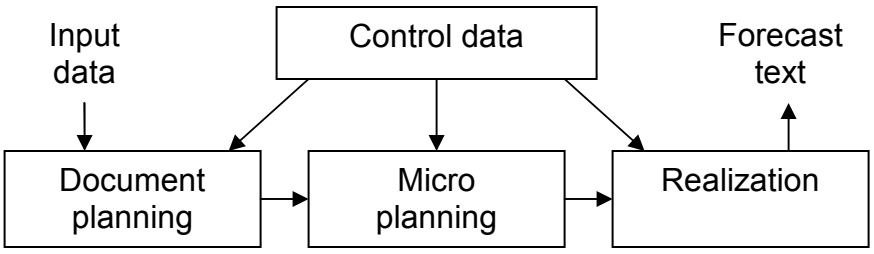

Figure 2.14: System architecture of SumTime-Mousam

As displayed in Figure 2.14, control data is a collection of external files that store end-user preferences. The data is used to customize the style and level of detail of the output as per end-user requirements. The document planner utilizes a bottom up segmentation algorithm primarily used in the KDD (Knowledge Discovery in Databases) community for data mining tasks in order to select the important facts from the input data that should be realized. In our system, instead of applying such learning algorithms, we followed a rule-based approach, the utility of which has been demonstrated by Demir et al. in [29].

In the micro planning stage, rules extracted from corpus analysis are used for lexical selection and omission or suppression of words. For example, given the input tuple *(0600, 8, 13, W, nil)*, the micro planner produces the output in (2.5).

```
Dir phrase 1: W                                                    (2.5)
Speed Phrase 1: 8-13
Time Phrase 1: by early morning
```

Finally, during the realization stage, the output from the micro-planner is ordered and augmented with punctuation markers to produce the textual forecast. An example output that has the previously mentioned tuple as part of it would look as shown in (2.6).

```
W 8-13 backing SW by mid afternoon and S 10-15 by                  (2.6)
midnight.
```

As mentioned previously, the output of the SumTime-Mousam system is edited by human authors before it is provided to the end-users. Thus, an evaluation of the system is carried out by counting the number of edits done on the forecast produced by the system. The system output is first segmented into phrases and aligned with the phrases from the human edited forecasts. Then the successfully aligned phrases are compared for exact matches where the result is found to be 43%. It is reported by the authors that 40% of the phrases did not match, i.e. were post edited by human authors and 17% of the phrases could not be aligned due to segmentation difference of the input data by humans and the system.

Yu et al. discuss the SUMTIME-Turbine system in [33], which generates textual summaries of time series data of gas turbines using knowledge based temporal abstraction (KBTA), pattern recognition and NLG techniques. After using KBTA methods, a summary is generated from the high level abstraction output of the KBTA module. The summary consists of two parts, the first being

some background information about the scenario and the second is the related interesting patterns extracted from the input data. A sample output of the system taken from the paper is provided in (2.7).

```
This scenario is about Fuel Valve subsystem which is
being monitored by channels: TNH, FSR, FSGR, FSROUT,
when the gas turbine is running in normal load
state from 21:03:41.00 28 Nov 99 to 00:03:41 29 Nov
99.
```
(2.7)

This structure of the output document is similar to that of our system where we start each section of the output with a single sentence that provides an overview and the succeeding sentences then provide related details.

For evaluating the quality of the summary generated by the system, plans of two methods are mentioned. The first one is to rank the system output by domain experts while the second one is to collect expert written summaries of the same scenario and compare them to the automatically generated outputs of the system, which is similar to our method to demonstrate the differences between human and system generated summaries as presented in chapter 4.

Belz discusses a probabilistic framework based approach to generating weather forecast text in [34]. The method utilizes a set of user defined generation rules as a context free language and estimates a probabilistic model based on a corpus containting example output sentences. As described by the author, apart from text quality comparable to that written by humans or generated by the SUMTIME system in terms of manual evaluation scores, the approach has benefits such as short development time and low computation time. However, the author also mentions that the human evaluators actually preferred the output

generated by the SUMTIME system in the experiments and the manually crafted component of the probabilistic system can significantly affect the overall output quality. Since such an approach would require users to define the base generator and also require a corpus of example outputs, which is difficult in our domain due to the extensive use of background information as will be discussed in chapter 3 and 4, we decided not to apply a probabilistic model in our system.

In [35], Sripada et al. describe how the Gricean Maxims [36] are applied for automatically generating textual summaries of time series data from different domains as discussed in [31, 32, 33]. This discussion motivated us to apply the Gricean Maxims as a general guideline in designing different components of our system. Below we present an overview of the Gricean Maxims, as described in [35]. In chapter 3, we discuss their application in the design and output generated by our system.

The Gricean Maxims are defined as a set of rules that specify the behaviors that a human hearer expects from a speaker. The four maxims can be briefly described as follows:

Maxim of quality:

1. Do not say what you believe to be false.
2. Do not say that for which you lack adequate evidence.

Maxim of quantity:

1. Make your contribution as informative as is required.
2. Do not make your contribution more informative than is required.

Maxim of relevance:

Be relevant.

Maxim of manner:

1. Avoid obscurity of expression.
2. Avoid ambiguity.
3. Be brief.
4. Be orderly.

As discussed by the authors of [35] the Gricean Maxims could be considered a key element of data analysis and so their application is important for effectively communicating information to human users. Hence we also apply them in different stages of generation in our system, as will be seen in the next chapter.

2.4 Chapter Summary

We have introduced some template based systems, identifying aspects that might be useful in our own approach. While many of these systems have not been formally evaluated, what evaluation has been done suggests that a template based approach might be more flexible in terms of extension to a new language and suitable for languages that have scarcity of computational resources. We also saw some other approaches that did not use templates, but gave us insight into automatic generation processes.

CHAPTER 3: THE GENERATION SYSTEM

Having introduced the research problem of investigating the suitability of a template based approach to cross-lingual NLG in the sports domain in chapter 1; we now present our approach to solve the problem. Specifically, we discuss the NLG system we built utilizing ideas from previous methods described in chapter 2, which is capable of generating short summaries of Cricket games in both Bangla and English. First we provide input and output examples, followed by the system architecture and an overview of how the input is transformed in different stages to produce the output. Then we look at the system processes in more detail and discuss the different operations carried out during generation. We conclude the chapter with a discussion on how the Gricean Maxims [35, 36] are applied during different phases of generation in our system.

3.1 Overview

We start with an example of a plain text input and the corresponding human written news bulletin. We discuss why automatically generating such output is difficult. Then we present the system architecture, the generated output and provide an overview of how the input data is transformed in various stages in order to generate the output.

The plain text input is a standard format of Cricket scorecards that is found to be similar across different sources, e.g. newspaper articles and websites, with

minor variations (for example, some of the sources may not include the number of minutes played or the wicket losing order of a batsman) in format.

We used cricinfo [37], which is one of the largest Cricket related websites, as the source of our input data. The scorecards retrieved from cricinfo are saved as plain text files and used as the unformatted input data. This unformatted data is converted to a structured format by the system, which can itself be considered as a parsing task, the details of which will be discussed later in the chapter. Figure 3.1 provides a shortened example of a plain text input to the system, created from the scorecard of a one day international Cricket match [38].

```
http://www.espncricinfo.com/the-ashes-2010-11/engine/current/match/446968.html
ODI 3098
Australia v England
England in Australia ODI Series - 7th ODI
Played at Western Australia Cricket Association Ground, Perth

Australia won by 57 runs
Toss Australia, who chose to bat
Series Australia won the 7-match series 6-1
Player of the match AC Voges (Australia)

Australia innings
TD Paine      lbw b Plunkett             5      16     7      0      0      71.42
BJ Haddin†    c Finn b Yardy             27     89     58     1      1      46.55
CJ Ferguson   c Strauss b Anderson       15     21     23     3      0      65.21
CL White*     c & b Yardy                24     70     47     1      0      51.06
DJ Hussey     c Bell b Plunkett          60     76     60     5      1      100.00
AC Voges      not out                    80     106    72     4      0      111.11
MG Johnson    c Prior b Anderson         26     31     25     2      0      104.00
JW Hastings   c Wright b Anderson        6      10     4      1      0      150.00
JJ Krejza     not out                    6      9      4      0      0      150.00
Extras        (lb 11, w 19)              30
Total  (7 wickets; 50 overs; 219 mins)   279 (5.58 runs per over)
Did not bat   SW Tait, DE Bollinger

Bowling
JM Anderson   10     1      48     3      4.80   (2w)
LE Plunkett   10     0      49     2      4.90   (6w)
ST Finn       10     1      57     0      5.70   (5w)
LJ Wright     9      0      47     0      5.22   (3w)
MH Yardy      10     0      59     2      5.90   (2w)
IJL Trott     1      0      8      0      8.00

England innings
AJ Strauss*   b Tait                     1      2      0      0      0      0.00
…
Extras        (lb 3, w 19, nb 5)         27
Total  (all out; 44 overs; 196 mins)     222 (5.04 runs per over)

Bowling
SW Tait       8      1      48     3      6.00   (1nb, 9w)
…
```

Figure 3.1: Plain text input

The first 2 paragraphs of the human authored game bulletin corresponding to the input data in Figure 3.1 are included in Figure 3.2. The whole bulletin is available in appendix 2.

```
Australia rounded off their international summer in style
with a commanding 57-run victory in Perth. It wasn't a
high-quality match, with the exception of the batting from
Adam Voges and David Hussey, as a long season drew to a
close with two patched-up sides on show. However,
Australia's depth came to the fore again as Voges hit a
career-best 80 before England's mentally-finished top order
was blown away to end hopes of a face-saving win.

Nothing will compensate for the crushing loss in the Ashes
series, but Australia's resurgent one-day form has
suggested a fourth consecutive World Cup title isn't out of
reach, especially if key players return from injury. Even
taking into account England's own injury problems and
declining form, the home side's performances have boded
well in the absence of Ricky Ponting, Mike Hussey and
Nathan Hauritz - all key figures in the one-day side.
```

Figure 3.2: Human authored bulletin

If we analyze the news bulletin above, it becomes obvious why automatically generating such output is difficult without background knowledge that humans have access to. For example, the winning margin of Australia, as mentioned in the first sentence is indeed available in the input data. However, it is not clear how the fact mentioned in the second sentence that the match was not a high-quality one except for the performances of Adam Voges and David Hussey could be deduced since there was one other player (Michael Yardy) who scored as many runs as Hussey, and two bowlers who took 3 wickets each, all of which are considerable performances. In the same way, it is not possible to deduce from the input that the English top order was mentally finished before being blown away. The same observation holds for the second paragraph of the

human written bulletin. Based on this argument, it can be stated that such text cannot realistically be generated by a system. So instead, we focus on utilizing the available data in the input to extract the key semantic concepts and realize them into proper NL sentences, as will be seen later in the chapter.

We followed the standard pipelined architecture for designing the generation system due to its advantages such as clear separation between different phases where a single component has a well defined task of transforming the data in a specific manner, low dependency of the components where the task of one component does not rely on that of others to a significant extent, and extensibility which allows adding new modules to the system as necessary without requiring changes to the existing components. However, a potential drawback of the architecture is the one way flow of data. For example, if after aggregation it is found that some of the selected information should have been excluded or some other information might have been included by the content selector, there is no easy way in the standard pipeline to accomplish this. However, since generation is usually done in a single pass, this kind of requirements do not usually show up.

Figure 3.3 provides a high level diagram of the system that displays the different phases of generation and their connectivity in the pipeline.

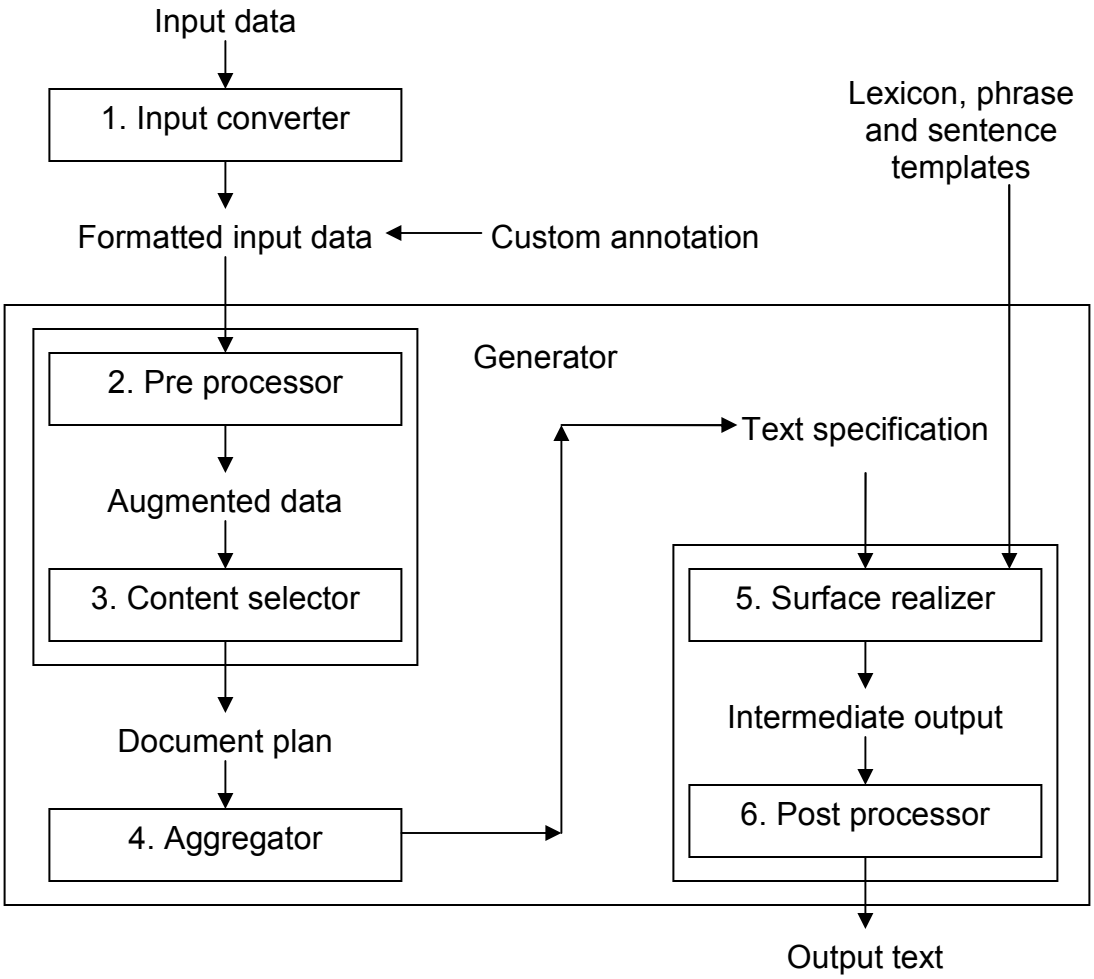

Figure 3.3: System architecture

The input to the system is a Cricket match scorecard in plain text in English. This is converted to a collection of key-value pairs, comprising a JSON [39] formatted tree structure by the input converter, which can be considered as the first phase of the pipeline. The reason behind choosing a tree structure is that it allows defining a logical hierarchy as well as ordering of information in the input, which is apparent by its widespread use in other NLG systems such as those described in [7, 17, 21, 22]. JSON is chosen instead of XML because it is

easier to manipulate in Java and does not require templates such as XSLT for transformation. Furthermore, it is more readable than XML and hence manually annotating or editing the data is easier.

The formatted version of input data acts as the actual input to the core generation system, which is provided to the document planning stage, consisting of the pre-processor and the content selection components. During these stages, the player portion of the input data is first formatted to create a list of news items and augmented with previous performances of the players for further processing by the pre-processor and passed to the content selection module. The content selector applies a set of content selection rules to determine which items of the input data should be selected for surface realization. It also specifies the realization order of the selected items and thus produces a document plan. Next, the document plan is provided to the aggregation module, which is the sole component of the micro-planning stage since currently the system does not perform referring expression generation. The aggregator determines which items of the document plan could be aggregated together to realize them as part of the same sentence in the output. After completing this stage, the modified document plan becomes the final text specification plan and is provided to the surface realizer. The surface realizer utilizes a set of sentence and phrase templates, along with a lexicon to create natural language sentences from the news items in the text specification tree as the intermediate output of the system. This intermediate output is forwarded to the post processor, which applies

capitalization or other rules on the output as specified in the language specific lexicon and produces the final output text in natural language.

Having discussed an overview of the system architecture, we now look at the transformation phases in the pipeline and the related data in more detail. In doing so, it is useful to first have output examples generated by the system. Game summaries of two different matches [38, 40] in both Bangla and English are included below. The outputs in both languages as displayed in Table 3.1 and 3.2 are exact generated texts by the system. Output text 1 is produced using the input data described in Figure 3.1.

Output Text (Game 1)

Bangla	English
অস্ট্রেলিয়া বনাম ইংল্যান্ড	Australia versus England
ওয়ান ডে ইন্টারন্যাশনাল ম্যাচ, ভেন্যু: ওয়েস্টার্ন অস্ট্রেলিয়া ক্রিকেট এসোসিয়েশন গ্রাউন্ড, পার্থ	One Day International at Western Australia Cricket Association Ground, Perth
অস্ট্রেলিয়া ৫৭ রানে জয়লাভ করে ।	Australia won by 57 runs.
অস্ট্রেলিয়া এর পক্ষে এডাম ভোগস ম্যান অফ দ্য ম্যাচ হন ।	Adam Voges of Australia was the player of the match.
টসে জিতে আগে ব্যাট করতে নেমে অস্ট্রেলিয়া ৫০.০ ওভারে ৭ উইকেটে ২৭৯ রান করে ।	After winning the toss & batting first Australia scored a total of 279 runs in 50.0 overs for 7 wickets.
অস্ট্রেলিয়া এর পক্ষে এডাম ভোগস ৭২ বল খেলে ৪টি চার সহযোগে অপরাজিত ৮০ রান করেন ।	For Australia, Adam Voges scored a half century of 80 runs unbeaten off 72 balls with 4 fours.
ডেভিড হাসি ৬০ বল খেলে ৫টি বাউন্ডারি এবং ১টি ওভার বাউন্ডারি সমেত ৬০ স্কোর করেন যদিও ইনিংসের প্রথম দিকে তাকে আত্মবিশ্বাসী দেখাচ্ছিল না ।	David Hussey scored a half century of 60 runs in 60 balls with 5 fours and 1 six despite having a shaky start to his innings.
ব্র্যাড হ্যাডিন ১টি ওভার বাউন্ডারি সমেত ২৭ স্কোর করেন ।	Brad Haddin scored 27 runs with 1 six.
মিচেল জনসন ২৫ বল খেলে ২৬ রান করেন ।	Mitchell Johnson scored 26 runs off 25 balls.
ক্যামেরন ওয়াইট ২৪ স্কোর করেন ।	Cameron White scored 24 runs.
ইংল্যান্ড এর পক্ষে জেমস অ্যান্ডারসন ১০.০ ওভার বল করে ৪৮ রান দিয়ে ৩ উইকেট লাভ করেন ।	For England, James Anderson took 3 wickets for 48 runs in 10.0 overs.
লিয়াম প্লাঙ্কেট ১০.০ ওভারে ৪৯ রান দিয়ে ২ উইকেট নেন ।	Liam Plunkett took 2 wickets and conceded 49 runs bowling 10.0 overs.
মাইকেল ইয়ার্ডি ১০.০ ওভারে ২ উইকেট নেন ।	Michael Yardy took 2 wickets bowling 10.0 overs.
জবাবে ব্যাট করতে নেমে ইংল্যান্ড ৪৪.০ ওভারে ২২২ রান করে অলআউট হয় ।	Batting second England scored a total of 222 runs in 44.0 overs & were allout.
মাইকেল ইয়ার্ডি ৭৬ বল খেলে ৩টি চার এবং ১টি ওভার বাউন্ডারি সমেত অপরাজিত ৬০ রান করেন ।	Michael Yardy scored a half century of 60 runs unbeaten from 76 balls with 3 boundaries & 1 six.
ম্যাট প্রাইওর ৩টি চার সমেত ৩৯ রান করেন ।	Matt Prior scored 39 runs with 3 boundaries.
লিউক রাইট ১৯ বল খেলে ৩টি বাউন্ডারি সমেত ২৪ রান করেন এবং তার ইনিংসটি দ্রুত শেষ হয়ে যায় ।	Luke Wright scored 24 runs in 19 balls with 3 fours and was back to the pavilion before long.
কেভিন পিটারসেন ৩১ বল খেলে ৩টি চার সহযোগে ২৬ রান করেন ।	Kevin Pietersen scored 26 runs in 31 balls with 3 fours.
লিয়াম প্লাঙ্কেট ১টি ছয় সমেত ২০ স্কোর করেন ।	Liam Plunkett scored 20 runs with 1 six.
এন্ড্রু স্ট্রাউস ১ স্কোর করেন এবং একটি অপ্রয়োজনীয় শট খেলে আউট হন ।	Andrew Strauss scored 1 run and got out playing a poor shot.
অস্ট্রেলিয়া এর পক্ষে মিচেল জনসন ৭.০ ওভার বল করে ১৮ রান দিয়ে ৩	For Australia, Mitchell Johnson took 3

উইকেট নেন ।	wickets for 18 runs in 7.0 overs.
শন টেইট ৮.০ ওভার বল করে ৩ উইকেট নেন ।	Shawn Tait took 3 wickets bowling 8.0 overs.
জেসন ক্রেজ্বা ৯.০ ওভারে ২ উইকেট নেন ।	Jason Krejza took 2 wickets bowling 9.0 overs.
ব্র্যাড হ্যাডিন ও ডেভিড হাসি ৩ ও ২টি ক্যাচ নেন ।	Brad Haddin and David Hussey took 3 and 2 catches.

Table 3.1: System output 1

Output Text (Game 2)

Bangla	English
দক্ষিণ আফ্রিকা বনাম ভারত	South Africa versus India
ওয়ান ডে ইন্টারন্যাশনাল ম্যাচ, ভ্যেনু: সুপার স্পোর্টস পার্ক, সেঞ্চুরিয়ন	One Day International at SuperSport Park, Centurion
দক্ষিণ আফ্রিকা ৩৩ রানে জয়লাভ করে ।	South Africa won by 33 runs.
দক্ষিণ আফ্রিকা এর পক্ষে হাশিম আমলা ম্যান অফ দ্য ম্যাচ হন ।	Hashim Amla of South Africa was the player of the match.
টসে হেরে আগে ব্যাট করতে নেমে দক্ষিণ আফ্রিকা ৪৬.০ ওভারে ৯ উইকেটে ২৫০ রান করে ।	After losing the toss and batting first South Africa scored a total of 250 runs in 46.0 overs for 9 wickets.
দক্ষিণ আফ্রিকা এর পক্ষে হাশিম আমলা ১৩২ বল খেলে ৯টি বাউন্ডারি সমেত অপরাজিত ১১৬ রান করেন ।	For South Africa, Hashim Amla scored a century of 116 runs unbeaten from 132 balls with 9 fours.
এটি গত ৫ ম্যাচে তার ৩য় অর্ধ-শতাধিক রান ।	In the last 5 matches, it was his 3rd score of 50 or more runs.
মরনে ভ্যান উইক ৬৩ বল খেলে ৮টি চার সহযোগে ৫৬ স্কোর করেন ।	Morne Van Wyk scored a half century of 56 runs from 63 balls with 8 boundaries.
জন-পল দুমিনি ১টি বাউন্ডারি সহযোগে ৩৫ রান করেন ।	Jean-Paul Duminy scored 35 runs with 1 four.
ভারত এর পক্ষে মুনাফ প্যাটেল ৮.০ ওভার বল করে ৩ উইকেট লাভ করেন ।	For India, Munaf Patel took 3 wickets in 8.0 overs.
যুবরাজ সিং ৮.০ ওভার বল করে ২ উইকেট লাভ করেন ।	Yuvraj Singh took 2 wickets bowling 8.0 overs.
জহির খান ৯.০ ওভারে ২ উইকেট নেন ।	Zaheer Khan took 2 wickets bowling 9.0 overs.
পরে ব্যাট করতে নেমে ভারত ৪০.২ ওভারে ২৩৪ রান করে অলআউট হয় ।	Batting second India scored a total of 234 runs in 40.2 overs & were allout.
ইউসুফ পাঠান ৭০ বল খেলে ৮টি চার আর ৮টি ছয় সহযোগে ১০৫ রান করেন ।	Yusuf Pathan scored a century of 105 runs off 70 balls with 8 boundaries and 8 sixes.
পার্থিব প্যাটেল ৩৪ বল খেলে ৬টি বাউন্ডারি সমেত ৩৮ রান করেন ।	Parthiv Patel scored 38 runs from 34 balls with 6 boundaries.
জহির খান ৩টি চার সমেত ২৪ স্কোর করেন ।	Zaheer Khan scored 24 runs with 3 boundaries.
দক্ষিণ আফ্রিকা এর পক্ষে মরনে মরকেল ৮.০ ওভার বল করে ৪ উইকেট নেন ।	For South Africa, Morne Morkel took 4 wickets in 8.0 overs.
ডেল স্টেইন ৯.০ ওভার বল করে ৩২ রান দিয়ে ২ উইকেট লাভ করেন ।	Dale Steyn took 2 wickets for 32 runs in 9.0 overs.
লনওয়াবো সতসোবে ৭.২ ওভারে ২ উইকেট নেন ।	Lonwabo Tsotsobe took 2 wickets bowling 7.2 overs.
মরনে মরকেল, ফাফ দ্যু প্লেসিস ও জন-পল দুমিনি ২টি ক্যাচ নেন ।	Morne Morkel, Faf Du Plessis and Jean-Paul Duminy took 2 catches.

Table 3.2: System output 2

It is worth mentioning that custom tags, which allow annotating the input with background information not available in the actual data, were utilized to improve the output quality. The details of the custom annotation tags are discussed in the next section. Also, use of the player performance history, aggregation, variation in the output sentences created using the same template, and other rules can be noticed in the output examples listed above. For example, balls faced and over-boundary, boundary counts are mentioned alongside the respective scores for not all of the batsmen. Similarly, runs conceded is only mentioned for some of the bowlers. These are accomplished by applying the content selection rules that determine which features should be chosen for surface realization. Details of these will be presented later in the chapter when the respective generation phases are discussed.

3.2 Input Conversion

The plain text data discussed earlier acts as an intermediate form of input that is transformed to create a tree structure during the first phase of the pipeline, i.e. input conversion. The formatted input data is produced by the input converter and acts as the main input to the generation system. It is essentially a JSON [39] formatted tree structure which is required to segment the input data in different sections such as game overview, game result, overall performance summaries of the two teams and a list of players for each team. In the plain text, performance data of a specific player might be scattered over multiple places, e.g. batting, bowling and catching. The input converter aggregates these under a single player node in the tree. Each player node consists of the batting and bowling

order, scores and other details related to the performance in the game of the corresponding player.

The formatted input supports adding custom annotation tags to each player item as displayed in Figure 3.4 and 3.5, which is another reason why the tree structured format is necessary. The custom tags allow users to optionally annotate each player's data with background information that are otherwise not available in the input. For example, it is not possible for the system to deduce from the input data that a player got out playing a poor shot, did not have a good start to his innings despite scoring runs later, or played a short lived innings. So, in order to improve the output text quality, a user having access to this kind of background knowledge can annotate the player data with such tags, which allows the system to include the corresponding phrase from the phrase template while realizing that player's data in the output (Figure 3.5). Furthermore, any news item that has a custom tag is always selected for realization by the content selection module, which allows users to override the content selection rules for particular news items, if necessary.

Currently, the system supports a total of six custom tags, three of which are used to annotate batting items and describe batting conditions such as *a shaky start to an innings,* a batsman getting out *playing a poor shot* and a *short lived innings.* The other three tags are bowling related, i.e. a player who *bowled with consistent line and length,* a bowler *giving away comparatively more runs later during the batting innings* or *being expensive during his earlier spells.* These tags/playing conditions were chosen after studying several human written

bulletins for frequently occurring situations that cannot be inferred from the input data. This short list of tags can be extended easily as required.

For example, if there is a news item annotated with the *short innings* custom tag (Figure 3.4), then it might be realized as displayed in Figure 3.5.

```
{
"playerName": "jj krejza",
"team": "australia",
"runsScored": 6,
"ballsFaced": 4,
"customTag": "shortInnings",
},
```

Figure 3.4: News item with a custom tag

```
Jason Krejza scored 6 runs off 4 balls and
was back to the pavilion before long.
```

Figure 3.5: The corresponding output text

Since not having access to background information is a major reason behind output produced by NLG systems being considered inferior to that written by humans [1], the use of custom tags is one possible way to solve this problem. Also, the idea of using custom tags might be utilized in generation tasks in other domains as well since it only requires carefully defining the phrases corresponding to the tags once, that can later be plugged into the existing sentences as many times as necessary without disrupting the flow.

The formatted version in shortened form, of the input data in Figure 3.1 is listed in Figure 3.6.

```
{ "gameId": 5003098,
  "gameOverview": {
    "team1": "australia",
    "team2": "england",
    "gameType": "ODI",
    "venue": "western australia cricket association ground, perth",
    "infoType": "gameOverview", },
  "gameResult": {
    "winnerTeam": "australia",
    "winBy": "57",
    "winByType": "runs",
    "pom": "ac voges",
    "pomTeam": "australia",
    "infoType": "gameResult", },
  "team1Summary": {
    "toss": "WIN",
    "batOrder": "first",
    "team": {
      "teamName": "australia",
      "over": 50.0,
      "wicket": 7,
      "run": 279,
      "extra": 30,
      "allOut": false },
    "infoType": "teamSummary", },
  "innings1Players": [{
      "playerName": "dj hussey",
       "keeper": false,
      "captain": false,
      "battingOrder": 5,
      "runsScored": 60,
      "ballsFaced": 60,
      "boundaryCount": 5,
      "overBoundaryCount": 1,
      "outType": "CATCH",
      "wicketTakerBowler": "plunkett",
      "wicketTakerAssist": "bell",
      "bowlingOrder": 6,
      "oversBowled": 4.0,
      "runsConceded": 16,
      "wicketsTaken": 0,
      "catchesTaken": 2, },],}
```


Figure 3.6: Formatted input

3.3 Pre-Processing

Pre-processing of the input is the second phase of data transformation in the pipeline. The first task of the pre-processor is to load previous performances of each player item in the formatted input. The player information and previous performances are stored by the pre-processor in a SQLite [41] database. It loads the required information from the database and augments the player objects with their previous performances, which are later used by the content selector to provide additional information to the realization module. After loading previous performances and updating the database with the player data of the current game if necessary, the pre-processor creates four lists of *play info* type objects, one batting and one bowling for each team from the list of player objects. Finally, the pre-processor carries out the first stage of document planning by specifying the realization order of the news items in the output text. At this stage, a fixed structure is followed for the output document where the game overview and result are followed by the summary of the team batting first. Then batting information of that team is followed by the bowling information of the opposing team in chronological order. After that, the summary of the team batting second is included, followed by the player batting information of that team and the bowling information of the opposing team.

3.4 Content Selection

The output of the pre-processor is forwarded for content selection, which is the third phase in the pipeline. The content selector is responsible for choosing the items and their features such as balls faced, overs bowled and catches

taken, from the news item list that should be realized in the output document. It also carries out the second stage of document planning by ordering the selected items according to a weight metric. In order to determine the news items and their features that should be included in the output text, the content selector runs a set of selection rules on each item, the details of which will be discussed later in the chapter.

It should be noted that, instead of the rule based approach similar to [20, 29] that we followed for content selection, a learning based approach could also be applied as discussed by Barzilay and Lapata in [42] where algorithms are applied to generate content selection rules for American football games from a collection of parallel corpus documents and database where the entries that should be included in the output are already specified. Since this approach would require preparing a significant amount of input data along with the corresponding documents, it was not followed for implementing our system. However, since the inner workings of the separate modules of the generation pipeline are encapsulated from each other, it could be implemented in future and compared to the current rule based method.

As mentioned in the previous section, there can be four types of news items, namely, game overview, game result, team summary and player performance items. Of these, the game overview and result items are always realized, have a fixed position in the output and do not contain optional features for realization. So these are immediately selected for realization without further processing. The two team summary items each precede the player performances

of the corresponding team in the output. So when the content selector finds one of these two items, it first checks the current team in the discourse to determine the correct position of the item in the output. Then it checks whether the optional *toss* feature has already been selected for realization. If not, then the feature gets selected for the team summary being processed. Finally, the *number of wickets* lost by the team being processed is tested to determine whether the feature should be realized as the numerical value or the *all out* (used to indicate that the team lost all 10 wickets) phrase and the appropriate value is added as a feature for realization.

Next, the four lists of player info items (batting and bowling lists for each of the two teams) are processed for content selection. The items are already provided in chronological order in the input and are separated into four lists based on their types (batting and bowling for each team) by the pre-processor. The content selector iterates through the lists and adds the items to a processing queue. The content selector also keeps track of the item type, i.e. batting or bowling in the discourse, to determine when the content selection rules should be run on the processing queue.

When the current item type changes in the discourse, the content selector stops adding new items and instead processes the queue. During this stage, the content selection rules are run on each item in the processing queue to determine whether the item should be included in the output and more importantly, the features of the item that should be realized. The items selected for realization are added to a temporary output queue.

Currently, the rules are embedded within the content selection module since the list of rules and their respective weights are adjusted and finalized during experiments on the development data set. The rules are not decoupled from the content selector since they do not require frequent adjustments. So, even though updating the existing rules or adding new ones is a straightforward process, it does require modifying the content selector code. It would be possible to represent the rules as displayed in Tables 3.3-3.5 using a high level language such as first order logic and read them from an external source, which would allow users to modify the rules. However, this was not implemented since it would increase overall complexity by requiring a separate parser for processing the rules and would not contribute to improve the output quality of the system. Furthermore, it is not entirely clear how useful such a strategy of decoupling the core algorithm from the system would prove to be since this approach was not followed in any of the previous systems discussed in chapter 2.

3.4.1 Selection Rules for Batting Items

For a batting item, if the *number of runs scored* is over the threshold amount set during the experiments on the development data set or if the item has a custom tag set then it is selected for realization. For selecting the features to include in the output text, the rules listed in Table 3.3 are applied on the item. The weight metric is used to determine the realization order of the item in the output. Initial weight is set as the amount of runs scored since it is the primary performance indicator of a batsman. The *added weight* column in Table 3.3 lists the amount by which the weight is increased when each rule is satisfied. The

values are determined by starting from a base value of 1 for all rules that are subsequently adjusted for each rule during the development stage to match the contents of the human written news bulletins.

Rule	Feature to realize	Added weight
Runs scored is in the top nth (top 3)	Runs scored	5
Runs scored ≥ 30 and player is team captain	Captain	3
Runs scored ≥ 100	Century	10
Runs scored ≥ 50 and < 100	Half Century (Selected with 20% probability)	10
Runs scored ≥ 50 and 2 or more scores of 50 or above in the history (last 10 games)	Batting history	7
Strike rate (runs scored/balls faced) is in the top nth (top 3) or > 85	Balls faced	3
Boundaries scored is in the top nth (top 3)	Boundary count	3
Over-boundaries scored is in the top nth (top 3)	Over-boundary count	3
Player did not get out	Not out	3

Table 3.3: Selection rules for batting items

3.4.2 Selection Rules for Bowling Items

For a bowling item, if the total calculated weight exceeds the threshold amount or if the item has a custom tag set and the player has taken one or more wickets then it is selected for realization. The threshold weight is determined by experimenting on the development dataset to maximize content matching with the human written output. Initial weight is set as the number of wickets taken since it is the primary performance indicator of a bowler. Then the rules listed in Table 3.4 are applied to calculate the total weight. As before, the *added weight* column lists the amount by which the weight is increased when each rule is satisfied. The values are determined by starting from a base value of 1 for all

rules that are subsequently adjusted for each rule during the development stage to match the contents of the human written news bulletins.

Rule	Feature to realize	Added weight
Wickets taken is in the top nth (top 3)	Wickets taken	5
Wickets taken is in the top nth (top 3) and player is team captain	Captain	3
Runs conceded is in bottom nth (bottom 3) or < 4	Runs conceded	3
Wickets taken ≥ 5	Wickets taken	5
Wickets taken ≥ 5 and 2 or more scores of 5 or above wickets in the history (last 10 games)	Bowling history	7
If multiple players take the same amount of wickets, the player bowling less overs should get higher weight	Wickets taken	10 / overs bowled
If multiple players take the same amount of wickets, the player with less RPO (runs conceded per over) should have higher weight, provided RPO < 4.5	Wickets taken	15 / RPO + wickets taken

Table 3.4: Selection rules for bowling items

3.4.3 Selection Rules for Catching Items

Information on catches taken is included in the bowling section of the output text. So when the bowling items are processed, they are also checked as catching items and added to a catching queue if selected for realization by the following rules listed in Table 3.5.

Rule	Feature to realize	Added weight
Catches taken is > 1 and is in the top nth (top 3)	Catches taken	Number of catches taken
Player is a wicket keeper	Wicket keeper	1

Table 3.5: Selection rules for catching items

After each item in the processing queue is checked, the output queue is sorted according to the calculated item weights in descending order and added to

the content selector's output list. If the output queue is for bowling items, additionally, the catching queue is also sorted and added to the output.

After content selection is completed for a batting or bowling list as described above, the process queue is emptied and the next list of items is processed in the same way. Finally, after every item in the input is processed and the realization order is determined, the content selector iterates over the output list of items. This time it keeps track of the current team in the discourse and if the *team* feature of the item being processed is found to be different than that of the discourse, then the feature is added for realization for the item and the discourse is updated accordingly. This rule allows disambiguation of the player team when the context moves from one team to the other, while at the same time avoids repetition of mentioning the team name for members of the same team. With this step, content selection, which is the second stage of document planning, is completed and the output is forwarded to the aggregation component.

3.5 Aggregation

The aggregation phase follows content selection in the pipeline and is the fourth stage of data transformation. The aggregator takes the output list of items produced by the content selector and determines which of these items can be aggregated together. The items in the list can be of different types such as overview, result, team summaries and batting, bowling or catching info items. For aggregation, we followed an idea similar to that presented in [23, 28] where items are semantically aggregated based on their domain and property similarity. In our

approach, we extend this idea by aggregating items based on their order, type (domain) and attribute. For example, in our system the aggregator may only aggregate items in consecutive order as specified during the previous phases of generation. For consecutive items that are potential candidates for aggregation, the algorithm first checks whether the items are of the same type, e.g. batting, bowling or catching. If the types match for consecutive items, then their attributes are checked for similarity. Our approach is again different here than the previously mentioned method since after matching order, type and attribute, we also apply a list of rules, as listed below, on the items to determine whether they can be effectively aggregated to form a single sentence. In order to be aggregated with its previous or next item in the list, an item must satisfy all of the rules. The items that conform to the rules are set to form chains where at most 3 items of the same type are aggregated to create a single output sentence during the surface realization stage. The maximum chain length of 3 is determined by experimenting with different lengths where it is found that for values greater than 3, the resulting sentences do not sound natural and become ambiguous, which violates the Gricean maxim of manner as described in chapter 2.

1. Items must be in consecutive order.
2. Items must be of the same type.
3. At most 3 items can be chained together.
4. A batting item with *did not bat* status should not be aggregated. This is because a *did not bat* status is unlikely to be present in the output since it is not considered to be a useful piece of information

and cannot effectively be aggregated with a sentence of the form "*X, Y and Z scored A, B and C runs*".

5. Batting items must have a positive value for the *runs scored* feature. This and the following 2 rules are used to filter out items with erroneous values in the input when the content selector might be disabled.
6. Bowling items must have a positive value for the *wickets taken* feature.
7. Catching items must have a positive value for the *catches taken* feature.
8. An item can have at most one optional feature for realization, i.e. either the *team* or the *wicket keeper* feature. Otherwise the resulting sentence becomes complicated and contradicts the Gricean maxim of manner, as introduced in chapter 2.

The resulting list where eligible items are marked for aggregation is treated as the text specification plan and is forwarded to the surface realization component for producing the output text.

3.6 Surface Realization

Surface realization is the fifth stage in the generation pipeline where output text in natural language is produced from the internal representation format. The surface realizer uses language specific lexicons and templates in order to generate natural language text from the structured data. Below we first

introduce the lexicon and templates and then discuss how the surface realizer utilizes those to create the output text.

3.6.1 Lexicon

A lexicon is required for each of the target languages by the surface realizer to provide generation support in multiple languages. The lexicon maps specific portions of the input data to their appropriate forms in the selected output language. The underlying idea of lexicons in our system is similar to those discussed in [14, 19, 23, 28] where lexicons are used to provide multilingual support and contain direct mappings of words or linguistic information such as word classes. However, in our approach, the lexicon contains mostly mapping of word forms and little linguistic information since the realizer does not utilize grammars.

The lexicon is formatted as a key, value pair on each line and contains mapping of player names and other language specific parameters such as the sentence end marker, whether the language needs sentence capitalization used to realize the input data, mappings of words such as *and, or, with, by* in the specified language. For example, a lexicon for Bangla might contain the following entries, among other things as shown in (3.1).

```
language:Bangla
start digit:০
sentence end marker:।
end marker prefix space:true
capitalize sentence:false
place 1:ম
place 2:য়
place 3:য়
place 4:র্থ
place default:ম
mn van wyk:মরনে ভ্যান উইক
..
```
(3.1)

3.6.2 Templates

In chapter 2, we discussed how templates are utilized in previous works on NLG. It can be noticed that the usage of templates varies from defining a parameterized description of the whole output document [13], specific portions of it [20, 21, 22] or even to describe a structure for the internal representation formats [7, 15, 16, 17]. In our system, we use a combination and extension of these ideas by introducing two levels of templates, i.e. the sentence templates and the phrase templates that together allow substantial flexibility and also provide variation in the output.

The sentence templates provide a means to specify basic structures of the different types of sentences for the surface realization component. In Cricket, the key semantic concepts of a game can be considered as the overview of the game, the result, score summary of each team and actions of individual players. The player actions can be further categorized as batting actions, consisting of runs scored, balls faced, boundaries/over-boundaries scored and losing wicket, bowling actions, which are wickets taken, overs bowled and runs conceded and

catching action, i.e. catches taken. Figure 3.7 provides a diagram of these key semantic concepts of Cricket.

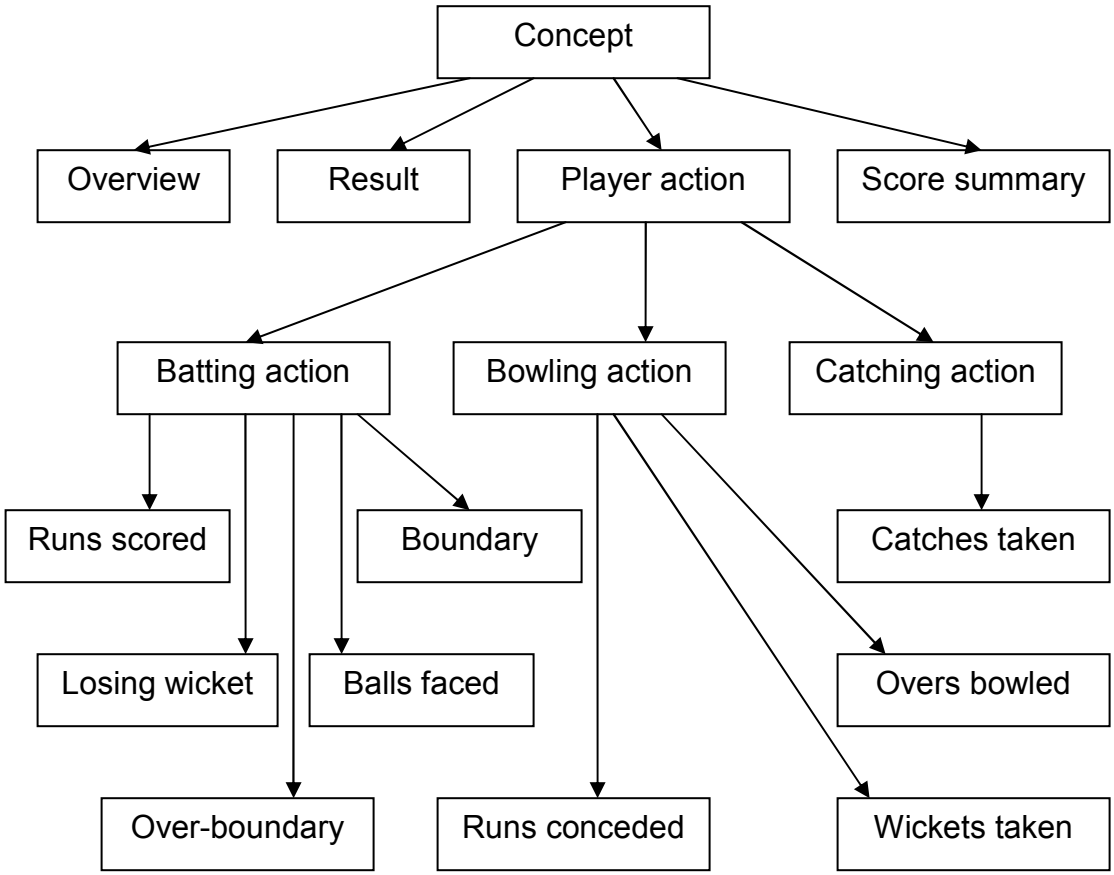

Figure 3.7: Key semantic concepts

In our approach, we define templates for each of these actions so that all the key semantic concepts can be realized in the output text. Of course, there could be other kinds of actions in a Cricket game such as an attempt by a fielder to stop a ball, captain choosing a specific bowler to bowl or a batsman attempting to play a particular shot. However, these were not included in our set of selected actions for these cannot be inferred from the input data. Also, the losing wicket action of a batsman is represented implicitly in the system output by only

specifically mentioning when a player remains not out at the end of the innings. The wicket taking bowler name and the wicket losing method were initially included in the realized output but were removed later to keep the sentence from getting too long.

Each sentence template in the system is a collection of phrase templates and static text values. The phrases can be specified as optional, in which case, they are realized only if the content determination module has specified the phrase as a feature for realization. The *key* or *id* of the template is matched with the type of the news item describing one or more semantic concepts by the surface realization module to determine which template is appropriate for realizing a specific news item. If multiple templates are provided for the same sentence, the system selects one template at random. Two example sentence templates, for Bangla and English, specifying the structure of the sentence to realize the *team summary* concept, along with their realized forms in the two languages are listed in Table 3.6. Notice how some of the optional phrases, i.e. those surrounded by (), are not present in the realized output. The other sentence templates used by the system are provided in Appendix 3.

Sentence template	**Realized output**
team summary: ([toss]) [batOrder] [bat] [teamName] [over] ([wicket]) [totalRun] ([allout]) ।	টসে জিতে আগে ব্যাট করতে নেমে অস্ট্রেলিয়া ৫০.০ ওভারে ৭ উইকেটে ২৭৯ রান করে ।
team summary: ([toss]) (and) [bat] [batOrder] [teamName] [totalRun] [over] (and) ([allout]) ([wicket]).	Batting second England scored a total of 222 runs in 44.0 overs & were allout.

Table 3.6: Sentence templates with realized outputs

While the sentence templates are used to define structures for the output sentences, the phrase templates describe mappings of key phrases used by the realizer to produce the surface forms of the corresponding phrases in the specific language. The phrase templates allow the system to provide variety in the output without requiring linguistic information to be specified in the input. They also define the player action related semantic concepts such as *runs scored*, *balls faced* and *wickets taken.* If there is more than one template specified for a phrase, the system selects one of the templates at random. Three possible phrase templates for Bangla as used by the system are displayed below in (3.2). A list of all the phrase templates is provided in Appendix 3.

```
runs scored:[x] রান করেন
bat history:এটি গত [x] ম্যাচে তার [x][th] অর্ধ-শতাধিক রান ।
wicketkeeper:উইকেটরক্ষক হিসাবে
```
(3.2)

3.6.3 Realization

Given a list of information items, along with their features to realize as determined during the content selection stage, the surface realizer starts the realization process by iterating over the items and checking whether each item is a member of an aggregation chain. If the item is not specified for aggregation, the surface realizer selects an appropriate sentence template for the item, based on the item-type. On the other hand, if the item is found to be a part of an aggregation chain, then the surface realizer stores the item in a temporary list and continues processing the following items in the same way until the final item of the chain is found. When all items of the chain are found, their item types are updated for aggregation and an appropriate sentence template is selected for

realizing the items as part of a single sentence. Key portions of the realization algorithm are provided in pseudocode in Table 3.7.

Each sentence template object of the surface realizer is capable of realizing a single type of sentence, in standard or aggregation mode. The correct sentence template and the realization mode for an item are selected using the item type, e.g. *team summary, batting, battingAggregate,* which can be considered as the condition for applying a template. Once the template and realization mode are selected, the module creates a list containing phrases or phrase templates in the order defined by the sentence template and iterates over each phrase in the list. If the phrase is specified as optional for realization, it is checked whether the phrase is specified by the content selector for surface realization in the list of features of the input item. If the phrase is found in the list, an appropriate template is selected and the data from the input item are used to realize the phrase template; otherwise the phrase is not included in the output.

On the other hand, if the phrase is not marked as optional, it is directly passed to the appropriate template for realization. The "*and"* phrase is realized after completing realization of all phrases in the list. Each "*and"* phrase is realized only when both of its left and right neighbour phrases are realized. The realization process is mostly similar for realizing a sentence in aggregation mode with the difference being instead of a single input item, there are multiple of them present. So for creating the output surface form of each phrase in the template, a list of phrases is prepared using one phrase from each input item and these are realized using the phrase template object.

Now, in order to create the surface form of each phrase, first the realizer tries to find an appropriate template for each. If a template is found, the matching property of the input item is extracted and the output text is created using the template and the property value. For example, if the *runsScored* phrase template is being processed, the property returned by the input item when queried with the *runsScored* phrase is retrieved and used in the slot of the template to create the output text. If the phrase template contains multiple slots, then the list of values are used in their retrieval order. The phrase template object also forces number, e.g. singular and plural, and position, e.g. 1^{st}, 2^{nd}, 3^{rd}, 4^{th} agreement as specified for the input language in the lexicon.

If a suitable template is not found, the realizer checks whether the lexicon contains a mapping for the phrase and uses it to create the output text. If a mapping is unavailable in the lexicon, the realizer considers the phrase as a property name of the input item and extracts the value of the property. It then tries the previous two steps, i.e. using the retrieved value as a template or finding a mapping of the value in the lexicon, in order to generate the surface form. If neither of these steps, nor treating the phrase as a property name works, then the only possible step remaining is to treat the phrase or its extracted value from the input item as a number and try to generate the output in the specified language. If this step also fails, the realizer generates an error message, realizes the phrase as-is and continues processing the next phrase in the list. In this manner, after processing is completed for all the phrases, the intermediate output text is forwarded to the post-processor.

Realization algorithm

```
for each news item
   add item to queue
   if marked as aggregate candidate
      change item type to aggregation
   else
      if item type has changed
         append newline to output
      update context
      if queue size = 1
         get matching sentence template for item
         sentenceTemplate.realize(item)
      else
         get matching aggregated sentence template for queue
         sentenceTemplate.realize(queue)
```

```
sentenceTemplate.realize(item)
   for each phrase in template
      if optional phrase
         if chosen by content selector for realization
            realizePhrase(phrase, item)
      else
         realizePhrase(phrase, item)
   process special phrases, i.e. and, with, sentence end markers and
   newlines
```

```
realizePhrase(phrase, item)
   retrieve values of phrase from item
   if matching phrase template is found
      phraseTemplate.realize(values)
   else if lexicon has matching entry for phrase
      return matched entry
   else if retrieved value from item is found
      if the value is a number
         generate language specific output
      else if matching phrase template is found (for value)
         phraseTemplate.realize(value)
      else if lexicon has matching entry for value
         return matched entry
      else
         use value as is
   else if the phrase is a number
      generate language specific output
   else
      print error message and use phrase as is
```

Table 3.7: Realization algorithm in pseudocode

3.7 Post-Processing

The post processing phase, being the final stage of the pipeline, is intended for applying orthographic rules on the output text that may not be suitable for application in the earlier stages of the pipeline. At present, the system applies a sentence capitalization rule during the post processing stage, if specified in the corresponding lexicon. The rule is intended for output language using the English alphabets and as such is not applicable for Bangla, which does not use capitalization. Another rule was tested where the system would insert appropriate adjectives or phrases before words in the output text, based on their numeric values. However, this rule resulted in unnatural sentences in the output when the aggregator was activated and hence was not enabled while generating the output text examples discussed here.

3.8 Implementation Details

The generation system was written in the Java programming language [43]. It uses the Google Gson library [44] for processing the JSON [39] formatted input and output files. SQLite [41] is used as the database system for storing player performance statistics. The system requires 1.65 and 0.57 seconds on average (over 5 runs each) to generate output with and without database access on a machine running Pentium 4 (Northwood) 2.60 GHz processor with 1 GB DDR RAM (PC 3200). The documented source code is available at the SFU Natural Language Laboratory website (http://natlang.cs.sfu.ca/) for viewing and/or experimentation purposes.

3.9 Application of the Gricean Maxims

We have already introduced the Gricean Maxims in chapter 2 and briefly stated how one of the rules is applied in the aggregation phase. Now we discuss how the maxims are used as a general guideline in different stages of generation in the system. The maxims are applied at all times, regardless of the specified output language of the system.

The maxim of quality is applied in the content determination, surface realization and post-processing stages of the system where we decided to generate text from game information available in the input data and not based on background information that may or may not be true (*lacks adequate evidence*). For example, a high individual score does not necessarily imply that the batsman is on a good batting form. So, we decided to only state the fact in the output, i.e. the number of runs scored such as "*David Hussey scored a half century of 60*" and not "*David Hussey continued his consistent batting form to score a half century of 60*".

The maxim of quantity is applied also in the content determination and surface realization stages where we decide to include only the most important information from the game data (*contribution should only be as informative as is required*), so we generate text for only the significant player performances. For example, even for the highest score of the game, which might be considered one of the most important information, we do not generate text for all of its features such as *balls played* or *boundaries scored* unless those are among the top values as well. So for the highest scorer, the system might generate "*For*

Australia, Adam Voges scored a half century of 80 runs" instead of "*For Australia, Adam Voges scored a half century of 80 runs off 72 balls with 1 over-boundary and 2 boundaries*".

The maxim of relevance is applied during content determination by not mentioning the same information multiple times, which would make the information irrelevant and redundant. For example, if the outcome of the toss is mentioned in one team's summary, it is skipped while generating text for the other team. So the system might generate the following sentences as the surface forms of the team summary items, "*After winning the toss & batting first Australia scored a total of 279 runs in 50.0 overs for 7 wickets*", and "*Batting second England scored a total of 222 runs in 44.0 overs & were allout*". In the same manner, the team is only mentioned once while the context remains the same, when generating text for the player performance items.

Finally, the maxim of manner is applied during the content determination, aggregation and surface realization stages. As mentioned previously, the content determination module only selects the most important facts for realization and thus satisfies the maxim of being brief. It also maintains the chronological order of batting and bowling of the teams and applies sorting by weight for the player performance items, thus applying the maxim that encourages orderliness of the items.

The aggregation component selects items for aggregation only when the item types are identical and there is a single realization feature. This conforms to the maxims *avoid obscurity of expression* and *avoid ambiguity* by generating

simple aggregated sentences such as "*Brad Haddin and David Hussey took 3 and 2 catches*" instead of "*Shawn Tait, Jason Krejza and Brad Haddin took 3 and 2 wickets, and 3 catches*".

The surface realization component applies the above mentioned two maxims by avoiding the use of more than necessary phrases, which might also contradict the maxim of quality as described above. So, for example, it generates sentences such as "*James Anderson took 3 wickets for 20 runs in 10.0 overs*" instead of "*James Anderson took 3 wickets for a mere 20 runs bowling all of his allotted 10.0 overs for a match winning performance*".

3.10 Chapter Summary

We have looked at input and output examples of our generation system and explained why the automatically generated output would differ from a human authored bulletin. We discussed the generation phases in detail including content selection and aggregation rules, and utility of lexicon and templates in surface realization. We also saw application of the Gricean maxims during different stages of generation in our system.

CHAPTER 4: PERFORMANCE DEMONSTRATION

In this chapter, after discussing the general issues related to the evaluation of NLG techniques, we present the methodology followed to compare the output of our system to that of human authors and discuss the corresponding experimental results.

4.1 Methodology

In general, evaluation of NLG systems is not a straightforward task due to the lack of standardized metrics and also the fact that not much is still known as to how an NLG system can be effectively evaluated [1]. This is also apparent in the discussion of NLG systems in chapter 2 where it can be noticed that an actual evaluation method is presented for few of the systems and even when an evaluation method is discussed it vastly differs from those mentioned in other papers and is mainly dependent upon the generation task.

Also, as stated by Reiter and Dale in [1], it is not known how meaningful the results of a quality evaluation are for predicting the success or failure of a particular NLG system. According to the authors, due to these reasons, NLG systems are usually evaluated based on user acceptance level. But then, this kind of evaluation is problematic due to the length of the required time span, and also for the fact that user acceptance might be influenced by factors which do not properly reflect the NLG technology used to develop the system. Therefore, as

the authors mention, other kinds of evaluation methods, such as testing success rate of human users in performing a task using the NLG system, or asking a group of expert users to judge the output of the system are used. These methods are known as black box evaluations since they test the performance of the system as a whole without knowledge of the internal workings of the system. On the other hand, the glass box evaluation methods measure the performance of each component of the system separately.

For our generation system, we performed glass box evaluation of the different components as unit tests, i.e. whether the components produce the expected results. For example, it was verified that for a given input case, whether the content selection module was choosing the appropriate content for realization and whether the order of the items to realize was according to expectation. In the same way, for the aggregation component, it was checked whether the module was selecting the proper items to be realized together and finally whether all specified realization-features of the selected items were present in the generated output.

However, black box evaluation, i.e. evaluating the performance of the system as a whole proved to be rather difficult. This is due to the following reasons. After analyzing several human authored reports of games, we found that in all the cases, humans use background knowledge not available in the input data to present information in the report. A few examples of this from actual game bulletins extracted from the Cricinfo website [37] are given in (4.1) – (4.4) below.

```
Shane Watson produced one of Australia's finest one-day
hundreds to carry them to a record-breaking six-wicket
win at the MCG with the highest successful chase on the
ground. [45]
```
(4.1)

```
The other opener Parthiv Patel, who has had only one net
session to adapt to South African conditions after flying
in as a replacement for Sachin Tendulkar, was in far
better touch but in the 10th over he was lbw missing a
full delivery from Tsotsobe. [46]
```
(4.2)

```
Watson (16) cut to point and Brad Haddin (37) walked
across his stumps to give Finn his first ODI wicket. [47]
```
(4.3)

```
Zaheer shortened his length and dismissed Miller with an
offcutter that the batsman failed to pull and gloved to
short fine leg. [48]
```
(4.4)

In the above cases, we can see that in (4.1) and (4.2), the authors used background knowledge such as "*record breaking 6 wicket win at the MCG with the highest successful chase on the ground*", and "*who (Parthiv Patel) had only one net session to adapt to South African conditions after flying in as a replacement for Sachin Tendulkar*" not available to the system from the input data. Again, another important point to be noted in the examples is the subjective manner in which background knowledge is used. For example, it is not clear how to define a "*one of finest one-day hundreds*" and this can vary between different human authors.

A different use of background information can be seen in (4.3) and (4.4). Here, the authors used information taken from the game that is not available in the input data. For example, it is not possible to interpret from the input that "*Brad Haddin walked across his stumps*" or "*Zaheer shortened his length and*

dismissed Miller with an offcutter". The only information available is that "*Brad Haddin scored 37*" and "*Zaheer took Miller's wicket*".

Since such information comprises a significant portion of human authored game bulletins, it is subsequently not possible to do a word-by-word, phrase-by-phrase or even presented-information wise comparison between the automatically generated and human authored reports. Hence, evaluation metrics such as BLEU [49] or ROUGE [50] would not be appropriate to evaluate the performance of the generation system.

Based on these observations, it may seem that a reasonable technique to compare the system and human authored outputs would then be to create a list of players mentioned in the human authored report and find how many of them are mentioned in the automatically generated report for the same game. However, this approach also has the following weaknesses. The first problem is that the length of human authored bulletins are significant as can be seen in [45, 46, 47 and 48] with lengths (in words) being 896, 845, 761 and 873 whereas the objective of the generation system is to produce concise summaries consisting of only the most important facts from the game (lengths in words of 261, 210, 253 and 240 for English; 277, 207, 260 and 249 for Bangla). Due to the extended length, it was found that most of the players (82% to 95% of all players in the games of the test data sets) were mentioned in the human authored reports either due to their performances in the game, or by using background information as discussed previously. On the other hand, mentioning all players is infeasible in the automatically generated report since the focus is on extracting the most

important facts from the game, i.e. including only the players whose performances contribute significantly to the outcome of the game.

A second observation that we made while analyzing human authored reports is that a player is mentioned there mainly for two reasons as described above, i.e. for his own performance or for other reasons such as to include facts not directly related to the outcome of the game, or for assisting some other player's performance. For the latter case, use of background information not available in the input data is necessary. We also noticed that when a player is mentioned for poor performance, on most of the cases information not available in the input data is used significantly alongside it (examples above), which is why we decided only to include positive performances in the system generated summary, unless specifically instructed by the user through custom tags.

Examples of mentions due to own performances of players could be as displayed in (4.5) and (4.6).

```
Trott's perfectly timed 102 off 126 balls [51]                    (4.5)

Chris Woakes, who took 6 for 45 [47]                              (4.6)
```

Whereas, examples of the second type of mention are as provided in (4.7) and (4.8) below.

```
Shaun Marsh (16) lazily flicked to midwicket [47]                 (4.7)

best partnership was 50 for the sixth wicket between              (4.8)
Kevin Pietersen, who top-scored with 78, and Michael
Yardy [45] (Here Michael Yardy who scored only 9 runs was
mentioned for assisting Kevin Pietersen)
```

Now, it is apparent from the above examples that since the system does not have access to background information not available in the input data, it will not be able to generate text comparable to humans who utilize such information. Therefore, a direct player list comparison between the human authored report and the system-generated summary may not be a very accurate measure to judge the overall performance of the system. However, since a better approach was not available, for demonstrating the output differences between the system and human authors we decided to extract the list of players from the human authored report and compare how many of them were included in the automatically generated report for the same game. To this end, we calculate the precision, recall and F-score measures [52] using the following formulae where the players in the human authored report are considered the *relevant* values and the players in the automatically generated report are considered the *retrieved* values. It should be noted that English bulletins extracted from the Cricinfo website [37] were focussed on for demonstrating the output differences since it was found that Bangla bulletins in general contain more background information than their English counterparts and thus would be less suitable to compare the system generated output to that of humans.

$$precision = \frac{players\ in\ auto\ report \cap players\ in\ manual\ report}{players\ in\ auto\ report}$$

$$recall = \frac{players\ in\ auto\ report \cap players\ in\ manual\ report}{players\ in\ manual\ report}$$

$$F - score = 2 * \frac{precision * recall}{precision + recall}$$

We used a set of seven games [38, 45, 47, 51, 53, 54, 55] as the development data set, which were used to generate preliminary summaries. These summaries were then compared to the human authored reports of the same games, and the system parameters were adjusted accordingly so that the generated reports covered most of the players mentioned in the human authored news bulletins.

4.2 Results

In the next phase, we used a set of five games [40, 44, 48, 56, 57], none of which were used previously for adjusting the parameters of the system, as the test data set. The teams of the test data set were different from the development data set and thus there was no overlap of players as well. We extracted the mentioned player names from the bulletins of each of these five games and compared those to the system-generated game summaries using the formulae described above. The results are displayed in Table 4.1 below.

Case	Players mentioned in human text (A)	Players mentioned in auto-text	Players in A mentioned in auto-text	Precision	Recall	F-score
1	18	16	13	0.81	0.72	0.76
2	19	15	14	0.93	0.74	0.82
3	14	15	11	0.73	0.79	0.76
4	19	9	8	0.89	0.42	0.57
5	18	12	11	0.92	0.61	0.73

Table 4.1: Precision, recall and F-score results

Case	Length of human written summary (in English words)	Length of auto-generated summary (in English words)	Length of auto-generated summary (in Bangla words)
1	913	260	267
2	873	240	249
3	909	252	255
4	845	210	207
5	904	224	226

Table 4.2: Lengths of human authored and system generated reports

From the above tables, it can be stated that the performance of the system was reasonable in general. Average precision of the 5 input cases was 0.86 with lowest being 0.73 for input case 3 where the system included a player in the output for taking 3 catches behind the wicket that it determined to be important information. However, this did not match with the human authored report and hence contributed to a significant decline of precision compared to the other input cases. For input case 1, there were 3 system selected players that were not chosen by human authors despite the fact that they took the highest number of wickets, catches or scored runs with good strikes rates. This resulted in 0.81 precision. In input case 2, 14 of the 15 system selected players were also included in the human report, thus the precision was highest at 0.93. In input case 4, precision was 0.89 since 8 of the 9 players that the system selected were also mentioned in the human report whereas for input case 5, it was at 0.92. Here, an interesting observation was that a player had the second best bowling figure for his team but still was not included in the human report. However, the system selected the player for realization and this was the only case where the system-selected information did not match that of the human.

For recall, the average was 0.66, which was lower than that of precision since the human authored reports usually mention a lot more players than the automatically generated ones due to their increased length as displayed in Table 4.2. The lowest recall was 0.42 for input case 4 where the system selected only 9 players for realization (compared to 12-16 in the other input cases), 8 of which matched the human authored report. However, the human authored report also mentioned other players (19 players in total, which is the highest of the input cases) using background information such as "*rarely found the middle of the bat or his timing", "adjudged caught-behind though it was unclear whether he edged the ball"* or "*perished to some senseless running"*, which contributed significantly to the decline in recall. For input case 1, the 5 players from the human report that the system did not select for realization were all found to be mentioned using background information such as "*moved across to drag a short-of-length delivery",* or "*played inside the line to lose his off stump".* In input case 2, recall was the second highest at 0.72 where 14 of the 15 system selected players matched with the human selection, whereas in input case 3, it increased to 0.79 since a comparatively lower number of players, 14 as opposed to 19 in case 2, were mentioned by the human author in the report. In input case 5, recall dropped to 0.61 even though the matched number of players remained the same, i.e. 11 as in case 3, because the number of human selected players increased to 18 from 14.

Instead of player names, if we consider the batting, bowling and catching actions, e.g. runs scored, balls faced, overs bowled, wickets taken, catches taken, then we can find some interesting patterns as displayed in Table 4.3.

Case	Player actions mentioned in human text (A)	Actions mentioned in auto-text	Actions in A mentioned in auto-text	Precision	Recall	F-score
1	15	16	11	0.69	0.73	0.71
2	8	15	6	0.40	0.75	0.52
3	5	15	5	0.33	1.00	0.50
4	8	9	7	0.78	0.88	0.82
5	9	12	8	0.67	0.89	0.76

Table 4.3: Precision, recall and F-score considering player actions

The first observation that we can make by comparing the number of players mentioned in the human reports from Table 4.1 to the number of actions above, is that a significant portion of players (3, 11, 9, 11 and 9) are included in the human reports not using actions but through other means such as background information. However, the system successfully includes most of the actions that are mentioned by humans in its output, which is apparent by the high recall values.

Moving onto precision, we find that the values are considerably lower than Table 4.1 because again human reports tend to include a small number of actions directly. Instead, they rely more on background information to describe a situation. For example, instead of specifying the number of *overs bowled* and *wickets taken* by *Zaheer Khan*, the report in [40] would say, "*troubles (of the batsmen) against Zaheer Khan continued*". Whereas, everything in the system-

generated report is based on player actions since it cannot utilize background information like humans. Thus precision gets lower even though recall is higher.

Table 4.4 presents the system generated outputs for input data set 5. Outputs for the rest of the data sets are available in appendix 1. Table 4.5 then provides the list of players and whether they were included in the human written English and Bangla bulletins [40, 58] and the system outputs.

Bangla	English
দক্ষিণ আফ্রিকা বনাম ভারত	South Africa versus India
ওয়ান ডে ইন্টারন্যাশনাল ম্যাচ, ভেন্যু: সুপার স্পোর্টস পার্ক, সেঞ্চুরিয়ন	One Day International at SuperSport Park, Centurion
দক্ষিণ আফ্রিকা ৩৩ রানে জয়লাভ করে ।	South Africa won by 33 runs.
দক্ষিণ আফ্রিকা এর পক্ষে হাশিম আমলা ম্যান অফ দ্য ম্যাচ হন ।	Hashim Amla of South Africa was the player of the match.
টসে হেরে আগে ব্যাট করতে নেমে দক্ষিণ আফ্রিকা ৪৬.০ ওভারে ৯ উইকেটে ২৫০ রান করে ।	After losing the toss & batting first South Africa scored a total of 250 runs in 46.0 overs for 9 wickets.
দক্ষিণ আফ্রিকা এর পক্ষে হাশিম আমলা ১৩২ বল খেলে ৯টি চার সহযোগে অপরাজিত ১১৬ রান করেন ।	For South Africa, Hashim Amla scored a century of 116 runs unbeaten in 132 balls with 9 fours.
এটি গত ৫ ম্যাচে তার ৩য় অর্ধ-শতাধিক রান ।	In the last 5 matches, it was his 3rd score of 50 or more runs.
মরনে ভ্যান উইক ৬৩ বল খেলে ৮টি চার সমেত ৫৬ রান করেন ।	Morne Van Wyk scored a half century of 56 runs in 63 balls with 8 boundaries.
জন-পল দুমিনি ১টি বাউন্ডারি সমেত ৩৫ স্কোর করেন ।	Jean-Paul Duminy scored 35 runs with 1 boundary.
ভারত এর পক্ষে মুনাফ প্যাটেল ৮.০ ওভার বল করে ৩ উইকেট নেন ।	For India, Munaf Patel took 3 wickets bowling 8.0 overs.
যুবরাজ সিং ৮.০ ওভারে ২ উইকেট নেন ।	Yuvraj Singh took 2 wickets bowling 8.0 overs.
জহির খান ৯.০ ওভারে ২ উইকেট লাভ করেন ।	Zaheer Khan took 2 wickets in 9.0 overs.
জবাবে ব্যাট করতে নেমে ভারত ৪০.২ ওভারে ২৩৪ রান করে অলআউট হয় ।	Batting second India scored a total of 234 runs in 40.2 overs & were allout.
ইউসুফ পাঠান ৭০ বল খেলে ৮টি চার আর ৮টি ওভার বাউন্ডারি সহযোগে ১০৫ রান করেন ।	Yusuf Pathan scored a century of 105 runs off 70 balls with 8 boundaries & 8 over-boundaries.
পার্থিব প্যাটেল ৩৪ বল খেলে ৬টি চার সহযোগে ৩৮ রান করেন ।	Parthiv Patel scored 38 runs from 34 balls with 6 fours.
জহির খান ৩টি চার সহযোগে ২৪ রান করেন ।	Zaheer Khan scored 24 runs with 3 boundaries.
দক্ষিণ আফ্রিকা এর পক্ষে মরনে মরকেল ৮.০ ওভারে ৪ উইকেট নেন ।	For South Africa, Morne Morkel took 4 wickets bowling 8.0 overs.
ডেল স্টেইন ৯.০ ওভারে ৩২ রান দিয়ে ২ উইকেট নেন ।	Dale Steyn took 2 wickets conceding 32 runs bowling 9.0 overs.
লনওয়াবো সতসোবে ৭.২ ওভারে ২ উইকেট নেন ।	Lonwabo Tsotsobe took 2 wickets in 7.2 overs.
মরনে মরকেল, ফাফ দ্যু প্লেসিস ও জন-পল দুমিনি ২টি ক্যাচ নেন ।	Morne Morkel, Faf Du Plessis & Jean-Paul Duminy took 2 catches.

Table 4.4: System output for input case 5

Player name	Mention type (human text in English)	Mention type (human text in Bangla)	Included in auto-text
Hashim Amla	Action	Action	Yes
Morne van Wyk	Action	None	Yes
Morne Morkel	Action	Action	Yes
Lonwabo Tsotsobe	Action	None	Yes
Jean-Paul Duminy	Other	Action	Yes
Faf du Plessis	Action	Action	Yes
Robin Peterson	Other	None	No
Graeme Smith	Other	Other	No
AB de Villiers	Other	None	No
Dale Steyn	None	None	Yes
Munaf Patel	Action	None	Yes
Yusuf Pathan	Action	Action	Yes
Mahendra Singh Dhoni	Other	None	No
Virat Kohli	Other	None	No
Rohit Sharma	Other	None	No
Yuvraj Singh	Action	None	Yes
Suresh Raina	Action	None	No
Parthiv Patel	Other	None	Yes
Zaheer Khan	Other	Action	Yes

Table 4.5: Player inclusion type details for input case 5

4.3 Discussion

In Table 4.5, we can see that 9 of 18 players that the human report in English includes are mentioned using player action events. The other 9 players are included using background information that may or may not be available in the input data. Of the 18 players and 9 actions mentioned in the human text, 12 and 8, respectively are present in the output generated by the system. One player (*Dale Steyn*) is included in the system generated report for taking 2 wickets and not conceding many runs, but is not mentioned in the human report.

On the other hand, the human report includes the explosive but brief innings of *Suresh Raina*, which is not selected by the system for not having any eventual impact on the game result. Again, comparing the system output to the Bangla bulletin, we find that the bulletin (consisting of 361 words) mentions a total of 7 players, 6 of which match with the system output. Thus precision, recall and F-score values are at 0.50, 0.86 and 0.63. Precision is comparatively lower for the Bangla report since the system output mentions a total of 12 players, 6 of which match the human authored output that mentions a total of only 7 players.

However, what can be considered a *good* or *poor* performance (and hence included or excluded in a report) is a subjective matter and might depend upon several factors. Thus, a comparison based on such measures might suffer from being subjective where the result might not indicate the true accuracy of the output. Therefore, for the test cases, ranking results of the system-generated outputs may differ when compared to other reports written by different human judges, none of which may in turn, match the ranking presented above. Again, the discussed method mainly focuses on content determination and does not cover testing the actual quality of the output text. However, this could be compensated by doing a user evaluation study where domain experts would assign a score to the output generated by the system.

Finally, the results may not be statistically significant due to the small size of the test data set and thus should mainly be considered suggestive. A comprehensive study consisting of multiple input cases covering each possible game scenario, e.g. low/high scores for one or both teams should be useful for

thoroughly evaluating the system by comparing its performance to that of a baseline. The baseline generator could be created by disabling (one or more at a time) some of the system components such as the content selector, aggregator and post-processor that are intended for improving the overall output quality.

CHAPTER 5: CONCLUSION

NLG is a subfield of NLP that is concerned with automatically generating human readable text such as summaries or descriptions of objects and scenarios from input data in non-linguistic format such as time series data, medical records, game score cards and ontology concepts. A widely used architecture for building NLG systems is based on a pipeline where the operations of each component of the system are well defined and separated from each other. The components process and transform input data in a specific order to produce the final output text of the system. NLG systems can be template based, i.e. utilize shallow processing where base structures of the output sentences are defined as templates that are transformed and values from the input data are filled in to generate the final output. The other approach is knowledge based and requires detailed grammatical and linguistic information to be embedded within the system. For languages such as Bangla, where much linguistic resources for NLP are not available yet, a template based approach might be suitable for generation tasks. Below, we summarize the work presented in the previous chapters on our NLG system and conclude with some future research directions.

5.1 Summary

In this thesis, we investigated the suitability of a template based approach for generating short summaries of Cricket games in a cross-lingual manner from structured input data, i.e. game score cards in a standard format. Based on our

findings, we presented an implementation of such a system that utilizes the pipelined architecture where the generation task is divided in subtasks performed by different components such as conversion of input data structure, content selection, aggregation and surface realization. The pre-processing stage transforms the input data from raw text to a structured JSON [39] based format that is used and augmented with information as necessary in later stages of the pipeline. The content selection module applies selection rules on the input data to determine which portions of it should be chosen for surface realization and also defines the output document structure. The aggregation component specifies which news items should be aggregated together in the output. Finally, the surface realizer produces the natural language sentences in the specified language from the structured data.

The presented system is based on two levels of templates, i.e. sentence and phrase templates that are easily extensible for creating significant variation in the output. Also, by utilizing domain knowledge in different stages of generation, and by avoiding the use of language specific resources such as grammars, it has been made possible for end users without expert linguistic knowledge to be able to create new templates or modify existing ones easily. The portability of the system to a new language is verified by first developing it with a focus on Bangla and later updating it to generate outputs in English simply by defining new sentence and phrase templates without requiring major modification to the core system. Finally, as discussed in the previous chapter, evaluation of NLG systems is difficult and non-trivial due to the lack of standardized methods

and the application oriented nature of the task. As such, demonstration of the system performance is carried out by comparing its output to contents taken from human authored news bulletins, which confirms that the system is able to extract the key semantic ideas from the input that match with the human authored text, and output those as sentences in multiple languages with reasonable accuracy with an average precision of 0.86 and average recall of 0.66 over 5 test cases.

5.2 Future Works

Having summarized the work on our NLG system, below we discuss some future research directions that can be followed to further improve its output quality.

Referring expression generation is one of the NLG tasks where the system applies rules on the input data to determine appropriate places where phrases such as *he, she, they, it* can be used to replace the corresponding proper nouns. It is a part of the micro-planning stage and can be included in the generation pipeline as phase 5, as displayed in Figure 5.1.

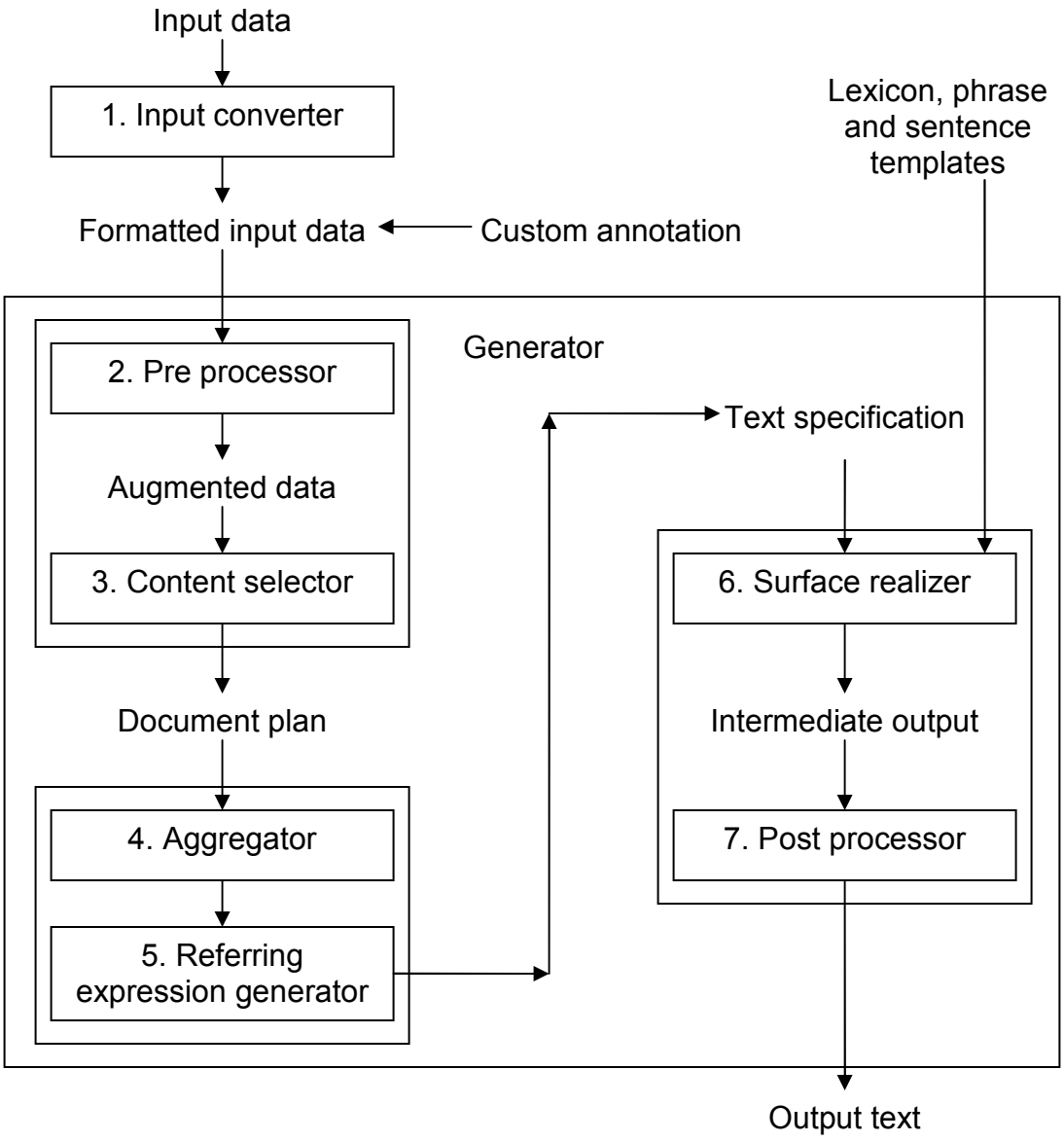

Figure 5.1: Revised system architecture

For example, after applying referring expression generation rules on the input, the system might determine that the consecutive sentences "*Mitchell Johnson scored 26 runs*" and "*Mitchell Johnson took 3 wickets*" should be realized as "*Mitchell Johnson scored 26 runs*" and "*He also took 3 wickets*". Our

system does not apply referring expression generation at present since a similar effect is achieved in the output using the templates. Furthermore, it is apparent by the automatically generated texts presented in chapter 3 and 4 that the scope of using referring expressions is very limited in the current system. Nevertheless, such a component could still be implemented, which would remove the present dependency on the templates but might introduce language specific rules within the system.

The content determination module could be improved further by analyzing more game data, which would allow implementing additional content selection rules or to adjust the existing ones in order to improve output accuracy. For example, currently the system does not include a bowler's performance information in the output if he has not taken any wickets. However, in a real life scenario, sometimes a bowler can influence the outcome without taking wickets. And although this is a possibility, it cannot always be interpreted from the input data and hence is not included in the content determination rules at present. But by analyzing more data, it might be possible to design new rules that would effectively address this case.

At present, the aggregation module of the system carries out aggregation only when the target sentences are simple, i.e. have a single realization feature. The reasoning behind this is explained in chapter 3, where the details of the component are discussed. However, output quality could be improved even more by adding appropriate phrases such as *respectively (e.g. scored 26, 36, and 41 runs, respectively)* or *each (e.g. took 3 catches each)* in the output based on the

aggregation parameters. These were not included in the current stage since the implementation is not straightforward within the template based framework and hence would require significant effort resulting in minor improvement of the output quality.

Variation in the output could be increased further by adding language specific grammatical knowledge to the system and by allowing templates to specify in which tense a parameter should be realized. For example, this sort of knowledge would allow realizing the *toss* phrase in the following two ways, *[toss:present_continuous]: "After winning the toss",* and *[toss:past_participle]:* "*Having lost the toss"*, where the system should have the knowledge that the *present continuous* and *past participle* forms of *win* and *loss* are *winning* and *lost,* respectively.

Since the system is dependent upon domain knowledge in order to generate output in multiple languages, it is not possible to directly use it to generate reports of a different sport. However, it would be interesting to see what kind of modification is required to the system to produce summary of an entirely different game such as Golf which, like Cricket, also uses a very structured format of scorecards. And this might also help redesigning some of the components to reduce domain dependency and decrease the time required to port the system to a different domain.

APPENDICES

APPENDIX 1: DEMONSTRATION DETAILS

Input Case 1

System output

Bangla	English
দক্ষিণ আফ্রিকা বনাম ভারত	South Africa versus India
ওয়ান ডে ইন্টারন্যাশনাল ম্যাচ, ভ্যেনু: কিংসমিড, ডারবান	One Day International at The Kingsmead, Durban
দক্ষিণ আফ্রিকা ১৩৫ রানে জয়লাভ করে ।	South Africa won by 135 runs.
দক্ষিণ আফ্রিকা এর পক্ষে লনওয়াবো সতসোবে ম্যান অফ দ্য ম্যাচ হন ।	Lonwabo Tsotsobe of South Africa was the player of the match.
টসে জিতে আগে ব্যাট করতে নেমে দক্ষিণ আফ্রিকা ৫০.০ ওভারে ৯ উইকেটে ২৮৯ রান করে ।	After winning the toss and batting first South Africa scored a total of 289 runs in 50.0 overs for 9 wickets.
দক্ষিণ আফ্রিকা এর পক্ষে এবি ডি ভিলিয়ার্স ৬৯ বল খেলে ৭টি চার ও ১টি ওভার বাউন্ডারি সহযোগে ৭৬ রান করেন ।	For South Africa, AB De Villers scored a half century of 76 runs off 69 balls with 7 fours and 1 six.
জন-পল দুমিনি ১টি ওভার বাউন্ডারি সহযোগে ৭৩ রান করেন ।	Jean-Paul Duminy scored 73 runs with 1 over-boundary.
হাশিম আমলা ৩৬ বল খেলে ৮টি চার সমেত ৫০ স্কোর করেন ।	Hashim Amla scored a half century of 50 runs from 36 balls with 8 boundaries.
ওয়েন পারনেল ১৯ বল খেলে ১টি ওভার বাউন্ডারি সহযোগে অপরাজিত ২১ স্কোর করেন ।	Wayne Parnell scored 21 runs unbeaten off 19 balls with 1 over-boundary.
ইযোহান বোথা ২৭ বল খেলে ২৩ রান করেন ।	Johan Botha scored 23 runs in 27 balls.
ভারত এর পক্ষে রোহিত শর্মা ৭.০ ওভারে ৩০ রান দিয়ে ২ উইকেট নেন ।	For India, Rohit Sharma took 2 wickets and conceded 30 runs bowling 7.0 overs.
জহির খান ১০.০ ওভার বল করে ৪৪ রান দিয়ে ২ উইকেট নেন ।	Zaheer Khan took 2 wickets and conceded 44 runs bowling 10.0 overs.
মুনাফ প্যাটেল ৭.০ ওভারে ৩৬ রান দিয়ে ২ উইকেট নেন ।	Munaf Patel took 2 wickets and gave 36 runs in 7.0 overs.
হরভজন সিং ২টি ক্যাচ নেন ।	Harbhajan Singh took 2 catches.
পরে ব্যাট করতে নেমে ভারত ৩৫.৪ ওভারে ১৫৪ রান করে অলআউট হয় ।	Batting second India scored a total of 154 runs in 35.4 overs and were allout.
ভিরাত কোহলি ৭০ বল খেলে ২টি চার এবং ১টি ছয় সমেত ৫৪ রান করেন ।	Virat Kohli scored 54 runs from 70 balls with 2 fours & 1 over-boundary.
সুরেশ রায়না ৩৬ বল খেলে ২টি বাউন্ডারি এবং ১টি ছয় সহযোগে ৩২ রান করেন ।	Suresh Raina scored 32 runs from 36 balls with 2 fours and 1 six.
মহেন্দ্র সিং ধোনি ১টি চার সমেত ২৫ রান করেন ।	Mahendra Singh Dhoni scored 25 runs with 1 boundary.
দক্ষিণ আফ্রিকা এর পক্ষে লনওয়াবো সতসোবে ৮.৪ ওভারে ৩১ রান দিয়ে ৪	For South Africa, Lonwabo Tsotsobe took 4 wickets conceding 31 runs in 8.4

উইকেট লাভ করেন ।	overs.
মরনে মরকেল ৫.০ ওভার বল করে ১২ রান দিয়ে ২ উইকেট লাভ করেন ।	Morne Morkel took 2 wickets and gave 12 runs in 5.0 overs.
ডেল স্টেইন ৬.০ ওভার বল করে ২ উইকেট লাভ করেন ।	Dale Steyn took 2 wickets bowling 6.0 overs.
ওয়েন পারনেল ৭.০ ওভারে ২৫ রান দিয়ে ১ উইকেট নেন ।	Wayne Parnell took 1 wicket and conceded 25 runs in 7.0 overs.
গ্রায়েম স্মিথ ২টি ক্যাচ নেন ।	Graeme Smith took 2 catches.

Table 5.1: System output for case 1

Comparison with human authored report

Player name	Mention type (human text)	Included in auto-text
AB De Villers	Action	Yes
Hashim Amla	Action	Yes
Jean-Paul Duminy	Action	Yes
Lonwabo Tsotsobe	Action	Yes
Morne Morkel	Action	Yes
Dale Steyn	Action	Yes
Wayne Parnell	Other	Yes
David Miller	Action	No
Johan Botha	None	Yes
Graeme Smith	None	Yes
Virat Kohli	Action	Yes
Munaf Patel	Action	Yes
Yuvraj Singh	Action	No
Murali Vijay	Action	No
Sachin Tendulkar	Action	No
Rohit Sharma	Action	Yes
Mahendra Singh Dhoni	Action	Yes
Harbhajan Singh	Other	Yes
Ashish Nehra	Other	No
Suresh Raina	Action	Yes
Zaheer Khan	None	Yes

Table 5.2: Player inclusion type details for case 1

Input Case 2

System output

Bangla	English
দক্ষিণ আফ্রিকা বনাম ভারত	South Africa versus India
ওয়ান ডে ইন্টারন্যাশনাল ম্যাচ, ভেন্যু: নিউ ওয়ানডার্াস স্টেডিয়াম, জোহানেসর্াগ	One Day International at New Wanderers Stadium, Johannesburg
ভারত ১ রানে জয়লাভ করে ।	India won by 1 run.
ভারত এর পক্ষে মুনাফ প্যাটেল ম্যান অফ দ্য ম্যাচ হন ।	Munaf Patel of India was the player of the match.
টসে জিতে আগে ব্যাট করতে নেমে ভারত ৪৭.২ ওভারে ১৯০ রান করে অলআউট হয় ।	After winning the toss and batting first India scored a total of 190 runs in 47.2 overs & were allout.
ভারত এর পক্ষে যুবরাজ সিং ৬৮ বল খেলে ৪টি বাউন্ডারি সমেত ৫৩ রান করেন ।	For India, Yuvraj Singh scored 53 runs from 68 balls with 4 boundaries.
মহেন্দ্র সিং ধোনি অধিনায়কোচিত ৩৮ স্কোর করেন ।	Mahendra Singh Dhoni as the team leader scored 38 runs.
শচীন টেন্ডুলকার ২টি চার সমেত ২৪ রান করেন ।	Sachin Tendulkar scored 24 runs with 2 fours.
ভিরাত কোহলি ২২ স্কোর করেন ।	Virat Kohli scored 22 runs.
দক্ষিণ আফ্রিকা এর পক্ষে লনওয়াবো সতসোবে ১০.০ ওভারে ২২ রান দিয়ে ৪ উইকেট লাভ করেন ।	For South Africa, Lonwabo Tsotsobe took 4 wickets and gave 22 runs in 10.0 overs.
ডেল স্টেইন ৯.২ ওভারে ৩৫ রান দিয়ে ২ উইকেট লাভ করেন ।	Dale Steyn took 2 wickets for 35 runs in 9.2 overs.
মরনে মরকেল ৮.০ ওভারে ২ উইকেট নেন ।	Morne Morkel took 2 wickets in 8.0 overs.
ইযোহান বোথা ১০.০ ওভার বল করে ৩৫ রান দিয়ে ১ উইকেট নেন ।	Johan Botha took 1 wicket for 35 runs in 10.0 overs.
পরে ব্যাট করতে নেমে দক্ষিণ আফ্রিকা ৪৩.০ ওভারে ১৮৯ রান করে অলআউট হয় ।	Batting second South Africa scored a total of 189 runs in 43.0 overs and were allout.
গ্রায়েম স্মিথ ৯৮ বল খেলে ৮টি চার সহযোগে অধিনায়কসুলভ ৭৭ স্কোর করেন ।	Graeme Smith as the captain scored a half century of 77 runs off 98 balls with 8 boundaries.
ডেভিড মিলার ২৮ বল খেলে ৩টি বাউন্ডারি এবং ১টি ছয় সহযোগে ২৭ স্কোর করেন ।	David Miller scored 27 runs in 28 balls with 3 fours and 1 over-boundary.
কলিন ইন্গ্রাম ৩০ বল খেলে ১টি বাউন্ডারি এবং ১টি ওভার বাউন্ডারি সহযোগে ২৫ স্কোর করেন ।	Colin Ingram scored 25 runs in 30 balls with 1 four and 1 six.
ভারত এর পক্ষে মুনাফ প্যাটেল ৮.০ ওভারে ২৯ রান দিয়ে ৪ উইকেট নেন ।	For India, Munaf Patel took 4 wickets conceding 29 runs in 8.0 overs.
জহির খান ৯.০ ওভার বল করে ২ উইকেট লাভ করেন ।	Zaheer Khan took 2 wickets in 9.0 overs.
হরভজন সিং ১০.০ ওভার বল করে ৩২ রান দিয়ে ১ উইকেট লাভ করেন ।	Harbhajan Singh took 1 wicket for 32 runs bowling 10.0 overs.
রোহিত শর্মা ২.০ ওভার বল করে ১ উইকেট লাভ করেন ।	Rohit Sharma took 1 wicket in 2.0

	overs.
উইকেটরক্ষক হিসাবে মহেন্দ্র সিং ধোনি ২টি ক্যাচ লেন ।	Mahendra Singh Dhoni took 2 catches as the wicket keeper.

Table 5.3: System output for case 2

Comparison with human authored report

Player name	Mention type (human text)	Included in auto-text
Yuvraj Singh	Action	Yes
Mahendra Singh Dhoni	Action	Yes
Munaf Patel	Action	Yes
Yusuf Pathan	Action	No
Zaheer Khan	Action	Yes
Harbhajan Singh	Other	Yes
Suresh Raina	Other	No
Sachin Tendulkar	Other	Yes
Virat Kohli	Other	Yes
Rohit Sharma	None	Yes
Lonwabo Tsotsobe	Action	Yes
Graeme Smith	Action	Yes
Morne Morkel	Other	Yes
Wayne Parnell	Action	No
Colin Ingram	Other	Yes
Jean-Paul Duminy	Other	No
David Miller	Other	Yes
Hashim Amla	Other	No
Johan Botha	Other	Yes
Dale Steyn	Other	Yes

Table 5.4: Player inclusion type details for case 2

Input Case 3

System output

Bangla	English
দক্ষিণ আফ্রিকা বনাম ভারত	South Africa versus India
ওয়ান ডে ইন্টারন্যাশনাল ম্যাচ, ভেন্যু: নিউল্যান্ডস, কেপটাউন	One Day International at Newlands, Cape Town
ভারত ২ উইকেটে জয়লাভ করে ।	India won by 2 wickets.
ভারত এর পক্ষে ইউসুফ পাঠান ম্যান অফ দ্য ম্যাচ হন ।	Yusuf Pathan of India was the player of the match.
টসে জিতে আগে ব্যাট করতে নেমে দক্ষিণ আফ্রিকা ৪৯.২ ওভারে ২২০ রান করে অলআউট হয় ।	After winning the toss & batting first South Africa scored a total of 220 runs in 49.2 overs and were allout.
দক্ষিণ আফ্রিকা এর পক্ষে ফাফ দ্যু প্লেসিস ২টি চার সহযোগে ৬০ রান করেন ।	For South Africa, Faf Du Plessis scored a half century of 60 runs with 2 fours.
জন-পল দুমিনি ৫৯ বল খেলে ২টি চার সহযোগে ৫২ স্কোর করেন ।	Jean-Paul Duminy scored a half century of 52 runs from 59 balls with 2 fours.
গ্রায়েম স্মিথ ৩টি বাউন্ডারি সহযোগে অধিনায়কসুলভ ৪৩ স্কোর করেন ।	Graeme Smith as the captain scored 43 runs with 3 fours.
ভারত এর পক্ষে জহির খান ৯.২ ওভার বল করে ৩ উইকেট নেন ।	For India, Zaheer Khan took 3 wickets in 9.2 overs.
হরভজন সিং ৯.০ ওভার বল করে ২৩ রান দিয়ে ২ উইকেট নেন ।	Harbhajan Singh took 2 wickets and gave 23 runs bowling 9.0 overs.
মুনাফ প্যাটেল ১০.০ ওভারে ৪২ রান দিয়ে ২ উইকেট লাভ করেন ।	Munaf Patel took 2 wickets for 42 runs in 10.0 overs.
ভিরাত কোহলি ৩টি ক্যাচ নেন ।	Virat Kohli took 3 catches.
জবাবে ব্যাট করতে নেমে ভারত ৪৮.২ ওভারে ৮ উইকেটে ২২৩ রান করে ।	Batting second India scored a total of 223 runs in 48.2 overs for 8 wickets.
ইউসুফ পাঠান ৫০ বল খেলে ৬টি চার এবং ৩টি ছয় সহযোগে ৫৯ রান করেন ।	Yusuf Pathan scored a half century of 59 runs off 50 balls with 6 boundaries and 3 over-boundaries.
সুরেশ রায়না ৪৭ বল খেলে ৪টি চার সমেত ৩৭ স্কোর করেন ।	Suresh Raina scored 37 runs off 47 balls with 4 fours.
ভিরাত কোহলি ৫টি চার সমেত ২৮ স্কোর করেন ।	Virat Kohli scored 28 runs with 5 fours.
হরভজন সিং ২৫ বল খেলে ২টি ছয় সমেত অপরাজিত ২৩ স্কোর করেন ।	Harbhajan Singh scored 23 runs unbeaten from 25 balls with 2 over-boundaries.
রোহিত শর্মা ২৩ রান করেন ।	Rohit Sharma scored 23 runs.
দক্ষিণ আফ্রিকা এর পক্ষে মরনে মরকেল ১০.০ ওভারে ২৮ রান দিয়ে ৩ উইকেট লাভ করেন ।	For South Africa, Morne Morkel took 3 wickets and conceded 28 runs bowling 10.0 overs.
ডেল স্টেইন ১০.০ ওভার বল করে ৩১ রান দিয়ে ২ উইকেট নেন ।	Dale Steyn took 2 wickets conceding 31 runs bowling 10.0 overs.
লনওয়াবো সতসোবে ১০.০ ওভার বল করে ৪১ রান দিয়ে ১ উইকেট লাভ করেন ।	Lonwabo Tsotsobe took 1 wicket and conceded 41 runs in 10.0 overs.

জন-পল দুমিনি ২.০ ওভার বল করে ১ উইকেট লাভ করেন ।	Jean-Paul Duminy took 1 wicket bowling 2.0 overs.
উইকেটরক্ষক হিসাবে এবি ডি ভিলিয়ার্স ৩টি ক্যাচ নেন ।	AB De Villers took 3 catches as the wicket keeper.

Table 5.5: System output for case 3

Comparison with human authored report

Player name	Mention type (human text)	Included in auto-text
Yusuf Pathan	Action	Yes
Harbhajan Singh	Action	Yes
Suresh Raina	Other	Yes
Zaheer Khan	Other	Yes
Ashish Nehra	Other	No
Virat Kohli	Other	Yes
Munaf Patel	Other	Yes
Rohit Sharma	None	Yes
Morne Morkel	Action	Yes
Faf Du Plessis	Action	Yes
Jean-Paul Duminy	Action	Yes
Johan Botha	Other	No
Wayne Parnell	Other	No
Graeme Smith	Other	Yes
Jean-Paul Duminy	Other	Yes
Dale Steyn	None	Yes
Lonwabo Tsotsobe	None	Yes
AB De Villers	None	Yes

Table 5.6: Player inclusion type details for case 3

Input Case 4

System output

Bangla	English
দক্ষিণ আফ্রিকা বনাম ভারত	South Africa versus India
ওয়ান ডে ইন্টারন্যাশনাল ম্যাচ, ভেন্যু: সেইন্ট জর্জেস পার্ক, পোর্ট এলিজাবেথ	One Day International at St. George's Park, Port Elizabeth
দক্ষিণ আফ্রিকা ৪৮ রানে জয়লাভ করে ।	South Africa won by 48 runs.
দক্ষিণ আফ্রিকা এর পক্ষে জন-পল দুমিনি ম্যান অফ দ্য ম্যাচ হন ।	Jean-Paul Duminy of South Africa was the player of the match.
টসে জিতে আগে ব্যাট করতে নেমে দক্ষিণ আফ্রিকা ৫০.০ ওভারে ৭ উইকেটে ২৬৫ রান করে ।	After winning the toss & batting first South Africa scored a total of 265 runs in 50.0 overs for 7 wickets.
দক্ষিণ আফ্রিকা এর পক্ষে জন-পল দুমিনি ৭২ বল খেলে ১টি ছয় সমেত অপরাজিত ৭১ স্কোর করেন ।	For South Africa, Jean-Paul Duminy scored a half century of 71 runs unbeaten off 72 balls with 1 over-boundary.
এটি গত ৪ ম্যাচে তার ৩য় অর্ধ-শতাধিক রান ।	In the last 4 matches, it was his 3rd score of 50 or more runs.
হাশিম আমলা ৬৯ বল খেলে ৮টি বাউন্ডারি সহযোগে ৬৪ রান করেন ।	Hashim Amla scored 64 runs off 69 balls with 8 boundaries.
ইযোহান বোথা ৩টি চার সহযোগে ৪৪ রান করেন ।	Johan Botha scored 44 runs with 3 boundaries.
রবিন পিটারসন ৩৫ বল খেলে ৩টি চার সমেত ৩১ রান করেন ।	Robin Peterson scored 31 runs from 35 balls with 3 fours.
ভারত এর পক্ষে যুবরাজ সিং ৮.০ ওভার বল করে ৩৪ রান দিয়ে ৩ উইকেট লাভ করেন ।	For India, Yuvraj Singh took 3 wickets for 34 runs bowling 8.0 overs.
পরে ব্যাট করতে নেমে ভারত ৩২.৫ ওভারে ৬ উইকেটে ১৪২ রান করে ।	Batting second India scored a total of 142 runs in 32.5 overs for 6 wickets.
ভিরাত কোহলি ৯২ বল খেলে ৭টি চার এবং ২টি ওভার বাউন্ডারি সহযোগে অপরাজিত ৮৭ রান করেন ।	Virat Kohli scored a half century of 87 runs unbeaten in 92 balls with 7 boundaries and 2 sixes.
সুরেশ রায়না ২০ স্কোর করেন ।	Suresh Raina scored 20 runs.
দক্ষিণ আফ্রিকা এর পক্ষে লনওয়াবো সতসোবে ৬.০ ওভারে ২ উইকেট লাভ করেন ।	For South Africa, Lonwabo Tsotsobe took 2 wickets bowling 6.0 overs.
মরনে মরকেল ৬.০ ওভারে ১৩ রান দিয়ে ১ উইকেট নেন ।	Morne Morkel took 1 wicket and gave 13 runs in 6.0 overs.
রবিন পিটারসন ৮.০ ওভার বল করে ২ উইকেট লাভ করেন ।	Robin Peterson took 2 wickets in 8.0 overs.
ইযোহান বোথা ৬.৫ ওভারে ২৭ রান দিয়ে ১ উইকেট নেন ।	Johan Botha took 1 wicket and gave 27 runs bowling 6.5 overs.

Table 5.7: System output for case 4

Comparison with human authored report

Player name	Mention type (human text)	Included in auto-text
Jean-Paul Duminy	Action	Yes
Hashim Amla	Action	Yes
Lonwabo Tsotsobe	Action	Yes
Graeme Smith	Action	No
Morne van Wyk	Other	No
AB de Villiers	Other	No
Faf Du Plessis	Other	No
Johan Botha	Action	Yes
Robin Peterson	Action	Yes
Morne Morkel	None	Yes
Yuvraj Singh	Action	Yes
Virat Kohli	Action	Yes
Zaheer Khan	Other	No
Munaf Patel	Other	No
Ashish Nehra	Other	No
Rohit Sharma	Other	No
Parthiv Patel	Other	No
Suresh Raina	Other	Yes
Mahendra Singh Dhoni	Other	No
Yusuf Pathan	Other	No

Table 5.8: Player inclusion type details for case 4

APPENDIX 2: HUMAN AUTHORED GAME BULLETIN

Voges helps Australia take series 6-1

Australia 7 for 279 (Voges 80*, Hussey 60, Anderson 3-48) beat England 222 (Yardy 60, Johnson 3-18, Tait 3-48) by 57 runs

Australia rounded off their international summer in style with a commanding 57-run victory in Perth. It wasn't a high-quality match, with the exception of the batting from Adam Voges and David Hussey, as a long season drew to a close with two patched-up sides on show. However, Australia's depth came to the fore again as Voges hit a career-best 80 before England's mentally-finished top order was blown away to end hopes of a face-saving win.

Nothing will compensate for the crushing loss in the Ashes series, but Australia's resurgent one-day form has suggested a fourth consecutive World Cup title isn't out of reach, especially if key players return from injury. Even taking into account England's own injury problems and declining form, the home side's performances have boded well in the absence of Ricky Ponting, Mike Hussey and Nathan Hauritz - all key figures in the one-day side.

During the Test matches, Australia's reserves did not appear up to international standard, but the team has not retained its No.1 one-day ranking by luck. Their pace attack is rapid, if wayward at times - they matched England's wide tally of 19 - while the lack of a matchwinning spinner isn't so harshly felt. Meanwhile, the batting is packed with stroke-makers.

Two were on show here after the top order wobbled to 4 for 102 before Hussey and Voges added 95 in 13 overs. This could be a one-off opportunity for Voges but if an injury replacement is needed for the World Cup, and Shaun Marsh doesn't recover, he might have put his name ahead of Callum Ferguson, who edged James Anderson to slip for 15.

Once Australia had posted a competitive total it was always unlikely that the visitors would be able to dig deep enough to make it a contest with the prospect of their flight home tomorrow evening. Mentally, a number of the players have long since been in those aircraft seats.

Andrew Strauss has plenty of reasons to be feeling weary after arriving in Australia on October 29 and it was a tired shot that ended his series when he was very late against Shaun Tait. The bat had barely come down when the ball knocked back the off stump. Steve Davies, back opening after the reshuffle caused by Eoin Morgan's injury, was unconvincing in his short stay until flapping at Doug Bollinger to complete an unhappy few weeks.

Jonathan Trott and Kevin Pietersen briefly consolidated but there was never any great sense of permanency even from the in-form Trott. He was drawn into a flat-footed drive against Johnson which sent a thick edge to first slip, then Pietersen's uncertain stay ended with a drive to backward point. Even taking into account the looseness of England's batting this was the good Mitchell Johnson and he added a third when Ian Bell carved down to third man.

At 5 for 64 the game was over. Matt Prior played some handsome drives before giving Jason Krejza his maiden one-day wicket to end another unfulfilled innings and Michael Yardy battled hard to reduce the margin of defeat with his highest ODI score. But it had ceased to matter in the wider context.

England's makeshift bowling attack had done a decent job through the first half of the innings as the quicks started well and Yardy picked up two, but in a familiar pattern the work of the front line bowlers was squandered. Hussey and Voges took advantage with some positive strokeplay as they dispatched the loose deliveries on offer. Hussey had been given a life on 4 when Luke Wright dropped a return chance that should have been held and reached his fifty from 44 balls, which included a pulled six off Yardy.

With his boundary-clearing ability and a Powerplay to come he could have cut loose during the final 10 overs, but was squared up by Liam Plunkett and got a leading edge to backward point. Plunkett ended with 2 for 49, which was an impressive effort considering that he only arrived in the country three days ago following a 40-hour journey from the Caribbean.

Voges, though, remained to reach fifty off 45 deliveries, regularly showing his strength square of the wicket and rapid running. Although the boundaries dried up he placed the ball well to ensure 34 came off the last three overs, but Australia were helped by England's waywardness. That was symptomatic of a team not fully focussed and the batting effort was further evidence that minds were elsewhere. If they want to perform at the World Cup there isn't much time to refocus, but Australia can leave for the subcontinent this week in good spirits.

APPENDIX 3: TEMPLATES

Syntax of Templates

Both the sentence and phrase templates follow a *key:value* structure where the key of the template is followed by a colon ":" which precedes the value as displayed in Tables 5.9 – 5.12. The sentence templates are structured as a collection of phrase templates (surrouded by braces "[]") or static text. The phrase templates within a sentence template are specified as optional phrases by surrounding them with braces "()". For the aggregation templates, value of the phrases that should be aggregated are specified by immediately following each phrase with "…" as displayed in Tables 5.9 and 5.10.

Sentence Templates

English Templates
language:english
game overview: [team1] versus [team2] [gameType] at [venue]
game result: ([resultTeam]) [result] ([margin]) ([marginType]). [pom] [pomTeam] was the player of the match.
team summary: ([toss]) (and) [bat] [batOrder] [teamName] [totalRun] [over] (and) ([allout]) ([wicket]).
batting: ([team]) [playerName] ([captainLike]) scored ([century]) ([halfCentury]) [runsScored] ([notOut]) ([ballsFaced]) (with) ([boundaryCount]) (and) ([overBoundaryCount]) ([customTag]). ([batHistory])
battingAggregate: ([team]) [playerName...] scored [runsScored...].
bowling: ([team]) [playerName] [wicketsTaken] ([runsConceded]) [oversBowled] ([asCaptain]) ([customTag]). ([bowlHistory])
bowlingAggregate: ([team]) [playerName...] [wicketsTaken...].
catching: ([team]) [playerName] [catchesTaken] ([wicketKeeper]).
catchingAggregate: ([team]) [playerName...] [catchesTaken...].

Table 5.9: Sentence templates (English)

Bangla Templates
language:bangla
game overview: [team1] বনাম [team2] [gameType], ভেন্যু [venue]
game result: ([resultTeam]) ([margin]) ([marginType]) [result] । [pomTeam] [pom] ম্যান অফ দ্য ম্যাচ হন।
team summary: ([toss]) [batOrder] [bat] [teamName] [over] ([wicket]) [totalRun] ([allout]) ।
batting: ([team]) [playerName] ([ballsFaced]) ([boundaryCount]) (and) ([overBoundaryCount]) (with) ([captainLike]) (and) ([notOut]) [runsScored] ([customTag]) । ([batHistory])
battingAggregate: ([team]) [playerName...] [runsScored...] ।
bowling: ([team]) ([asCaptain]) [playerName] [oversBowled] ([runsConceded]) [wicketsTaken] ([customTag]) । ([bowlHistory])
bowlingAggregate: ([team]) [playerName...] [wicketsTaken...] ।
catching: ([team]) ([wicketkeeper]) [playerName] [catchesTaken] ।
catchingAggregate: ([team]) [playerName...] [catchesTaken...] ।

Table 5.10: Sentence templates (Bangla)

Phrase Templates

English Templates

```
language:english
number agreement:true
runs scored: [x] runs
balls faced: off [x] balls
balls faced: from [x] balls
balls faced: in [x] balls
boundary count: [x] boundaries
boundary count: [x] fours
overBoundary count: [x] over-boundaries
overBoundary count: [x] sixes
wickets taken: took [x] wickets
runs conceded for wicket: for [x] runs
runs conceded: for [x] runs
runs conceded: and gave [x] runs
runs conceded: and conceded [x] runs
runs conceded: conceding [x] runs
bat history: In the last [x] matches, it was his [x][th] score of 50 or
more runs.
bowl history: In the last [x] matches, it was the [x][th] time he took 5
or more wickets.
catches taken: took [x] catches
team:For [x],
pomTeam:of [x]
over: in [x] overs
overs bowled: in [x] overs
overs bowled: bowling [x] overs
wicket: for [x] wickets
total run: scored a total of [x] runs
bat:batting
as captain: as the captain
as captain: as the team leader
as captain: as the leader
captain like: as the captain
captain like: as the team leader
captain like: as the leader
toss: After [x] the toss
all out: were allout
not out: unbeaten
respectively:respectively
wicketKeeper:as the wicket keeper
century:a century of
half century:a half century of
```

```
shaky start: despite having a shaky start to his innings
poor shot: and got out playing a poor shot
short innings: and was back to the pavilion before long
won:won by
tied:match tied
consistent bowling: with consistent line and length
late run givaway: being expensive later in the innings
early run givaway: being expensive early on in the innings
```

Table 5.11: Phrase templates (English)

Bangla Templates
language:bangla
runs scored:[x] রান করেন
runs scored:[x] স্কোর করেন
balls faced: [x] বল খেলে
boundary count:[x]টি চার
boundary count:[x]টি বাউন্ডারি
overBoundary count:[x]টি ছয়
overBoundary count:[x]টি ওভার বাউন্ডারি
wickets taken:[x] উইকেট নেন
wickets taken:[x] উইকেট লাভ করেন
runs conceded for wicket:[x] রানের বিনিময়ে
runs conceded:[x] রান দিয়ে
bat history:এটি গত [x] ম্যাচে তার [x][th] অর্ধ-শতাধিক রান।
bowl history:এটি গত [x] ম্যাচে তার [x][th] ৫ বা ততোধিক উইকেট।
zero score:০ রানে আউট হন।
out score:[x] রানে আউট হন
catches taken:[x]টি ক্যাচ নেন
team:[x] এর পক্ষে
pomTeam:[x] এর পক্ষে
over:[x] ওভারে
overs bowled:[x] ওভার বল করে
overs bowled:[x] ওভারে
wicket:[x] উইকেটে
total run:[x] রান করে
bat:ব্যাট করতে নেমে
captain like:অধিনায়কোচিত
captain like:অধিনায়কসুলভ
as captain:দলনেতা হিসাবে
as captain:অধিনায়ক হিসাবে
toss:টসে [x]
all out:অলআউট হয়
not out:অপরাজিত
respectively:যথাক্রমে
wicketkeeper:উইকেটরক্ষক হিসাবে
shaky start:যদিও ইনিংসের প্রথম দিকে তাকে আত্মবিশ্বাসী দেখাচ্ছিল না
poor shot:এবং একটি অপ্রয়োজনীয় শট খেলে আউট হন
short innings:এবং তার ইনিংসটি দ্রুত শেষ হয়ে যায়
won: জয়লাভ করে
tied:ফলাফল অমিমাংসিত
consistent bowling: এবং ধারাবাহিকভাবে কার্যকর বোলিং করেন
late run givaway: যদিও ইনিংসের শুরুতে তিনি তুলনামূলকভাবে কম রান দেন
early run givaway: যদিও ইনিংসের শুরুতে তিনি তুলনামূলকভাবে বেশি রান দেন

Table 5.12: Phrase templates (Bangla)

REFERENCE LIST

1. E. Reiter and R. Dale. (2000). Building Natural Language Generation Systems. *Cambridge University Press, Cambridge, UK.*

2. K. van Deemter, E. Krahmer and M. Theune. (2005). Real Versus Template-Based Natural Language Generation: A False Opposition? *Computational Linguistics 31(1):15–24.*

3. M. P. Lewis. (2009). Ethnologue: Languages of the World, Sixteenth edition. *Dallas, Tex: SIL International. Online version: http://www.ethnologue.com/.*

4. S. Bhattacharya, M. Choudhury, S. Sarkar and A. Basu. (2005). Inflectional Morphology Synthesis for Bengali Noun, Pronoun and Verb Systems, *In Proceedings of the National Conference on Computer Processing of Bangla (CCPB 05), pp. 34 - 43, Dhaka, Bangladesh.*

5. A. Das and S. Bandyopadhyay. (2011). Syntactic Sentence Fusion Techniques for Bengali. *In International Journal of Computer Science and Information Technologies (IJCSIT), Vol. 2 (1), 494-503.*

6. S. Das, A. Basu and S. Sarkar. (2010). Discourse Marker Generation and Syntactic Aggregation in Bengali Text Generation. *In Proceedings of the IEEE Students' Technology Symposium, 3-4 April 2010, IIT Kharagpur, India.*

7. G. Wilcock. (2005). An Overview of Shallow XML-Based Natural Language Generation. *In Proceedings of the Second Baltic Conference on Human Language Technologies, pp 67-78*.

8. M. Theune, E. Klabbers, J. Odijk, J.R. de Pijper, and E. Krahmer. (2001). From Data to Speech: A General Approach. *Natural Lang. Eng. 7(1): 47–86.*

9. G. Wilcock. (2003). Talking OWLs: Towards an Ontology Verbalizer. *In Proceedings of the Workshop on Human Language Technology for the Semantic Web, 2nd International Semantic Web Conference, pages 109–112, Sanibel Island, fl.*

10. Prothom Alo (2010, September 2). Retrieved from http://www.prothom-alo.com/detail/ref/nf/date/2010-09-02/news/91262

11. S. Busemann and H. Horacek. (1998). A Flexible Shallow Approach to Text Generation. *In Proceedings of the International Natural Language Generation Workshop. Niagara-on-the-Lake, Canada.*

12. E. Reiter, S. Sripada, and R. Robertson. (2003). Acquiring Correct Knowledge for Natural Language Generation. *Journal of Artificial Intelligence Research, 18: 491-516.*

13. M. Cristiá, B. and Plüss. (2010). Generating Natural Language Descriptions of Z Test Cases. *In Proceedings of the 6th International Natural Language Generation Conference, Dublin, Ireland.*

14. E. Reiter, C. Mellish and J. Levine. (1995). Automatic Generation of Technical Documentation. *Applied Articial Intelligence 9: 259-287.*

15. G. Wilcock. (2003). Integrating Natural Language Generation with XML Web Technology. *In Proceedings of the Demo Sessions of EACL-2003, pages 247–250, Budapest*.

16. G. Wilcock. (2001). Pipelines, Templates and Transformations: XML for Natural Language Generation. *In Proceedings of the 1st NLP and XML Workshop, Tokyo. 1–8.*

17. G. Wilcock. (2002). XML-Based Natural Language Generation. *In Towards the Semantic Web and Web Services: XML Finland 2002 – Slide Presentations, Helsinki. 40–63.*

18. OWL Web Ontology Language Overview. (2011, June 6). Retrieved from http://www.w3.org/TR/owl-features/

19. D. Galanis and I. Androutsopoulos. (2007). Generating Multilingual Personalized Descriptions from OWL Ontologies on the Semantic Web: the NaturalOWL System.

20. M. White and T. Caldwell. (1998). EXEMPLARS: A Practical Extensible Framework for Dynamic Text Generation. *In Proceedings of the Ninth International Workshop on Natural Language Generation (INLG-1998), Niagara-on-the-Lake, ON, pp. 266–275.*

21. H. Stenzhorn. (2002). A Natural Language Generation System using XML and Java Technologies. *In Proceedings of the 2nd Workshop on NLP and XML, Taipei, Taiwan.*

22. S. W. McRoy, S. Channarukul and S. Ali. (2003). An Augmented Template-Based Approach to Text Realization. *Natural Language Engineering, 9(4):381–420.*

23. K. Bontcheva and Y. Wilks. (2004). Automatic Report Generation from Ontologies: The MIAKT Approach. *In Proceedings of the 9th International Conference on Applications of Natural Language to Information Systems (NLDB'2004), Manchester, UK.*

24. O. Lassila and R. Swick. (1999). Resource Description Framework (RDF) Model and Syntax Specification. *Technical Report 19990222, W3C Consortium, http://www.w3.org/TR/REC-rdf-syntax/*

25. X. Sun and C. Mellish. (2006). Domain Independent Sentence Generation from RDF Representations for the Semantic Web. *In Combined Workshop on Language Enabled Educational Technology and Development and Evaluation of Robust Spoken Dialogue Systems, European Conference on AI, Riva del Garda, Italy.*

26. B. Lavoie and O. Rambow. (1997). A Fast and Portable Realizer for Text Generation Systems. *In Proceedings of the Fifth Conference on Applied Natural Language Processing (ANLP 97), pp. 265-268, Washington, D.C.*

27. D. Hewlett, A. Kalyanpur, V. Kolovski, and C. Halaschek-Wiener. (2005). Effective NL Paraphrasing of Ontologies on the Semantic Web. *In Proceedings of the Workshop on End-User Semantic Web Interaction, 4th International Semantic Web Conference, Galway, Ireland.*

28. K. Bontcheva. (2005). Generating Tailored Textual Summaries from Ontologies. *In 2nd European Semantic Web Conference, ESWC 2005, volume 3532 of LNCS, pages 241–256, Heraklion, Crete, Greece, Springer.*

29. S. Demir, S. Carberry, and K. F. McCoy. (2008). Generating Textual Summaries of Bar Charts. *In Proceedings of The International Natural Language Generation Conference (INLG)*.

30. M. Elhadad and J. Robin. (1996). An Overview of SURGE: A Re-Usable Comprehensive Syntactic Realization Component. *In Proceedings of The International Natural Language Generation Conference (INLG)*.

31. S. Sripada, E. Reiter, J. Hunter and J. Yu. (2001). A Two-Stage Model for Content Determination. *In Proceedings of ENLG-2001, 2001, pp. 3–10.*

32. S. Sripada, E. Reiter and I. Davy. (2003). SumTime-Mousam: Configurable Marine Weather Forecast Generator, *Expert Update 6 (3), pp. 4–10.*

33. J. Yu, E. Reiter, J. Hunter and S. Sripada. (2003). SumTime-Turbine: A Knowledge-Based System to Communicate Gas Turbine Time-Series Data, *In Proceedings of the 16th international conference on Developments in applied artificial intelligence, p.379-384, June 23-26, 2003, Laughborough, UK.*

34. A. Belz. (2007). Probabilistic Generation of Weather Forecast Texts. *In Proceedings of NAACL-HLT.*

35. S. Sripada, E. Reiter, J. Hunter and J. Yu. (2003). Generating English Summaries of Time Series Data using the Gricean Maxims. *In Proceedings of KDD-2003, pp 187 –196.*

36. H. P. Grice. (1975). Logic and Conversation. *Speech Acts 3:41–58.*

37. ESPN cricinfo. (2011). Retrieved from http://www.cricinfo.com/

38. Australia v England, 7th ODI, Perth. (February 06, 2011). Retrieved from http://www.espncricinfo.com/the-ashes-2010-11/content/story/499682.html

39. JSON. (2011, June 6). Retrieved from http://json.org/

40. South Africa v India, 5th ODI, Centurion. (January 23, 2011). Retrieved from http://www.espncricinfo.com/south-africa-v-india-2010/content/story/497947.html

41. SQLite. (2011, June 1). Retrieved from http://www.sqlite.org/index.html

42. R. Barzilay and M. Lapata. (2005). Collective Content Selection for Concept-to-Text Generation. *In Proceedings of the Human Language Technology Conference and the Conference on Empirical Methods in Natural Language Processing, 331–338, Vancouver.*

43. Java (programming language). (2011, June 6). Retrieved from http://www.oracle.com/technetwork/java

44. Google Gson. (2011, June 1). Retrieved from http://code.google.com/p/google-gson/

45. Australia v England, 1st ODI, Melbourne. (January 16, 2011). Retrieved from http://www.espncricinfo.com/the-ashes-2010-11/content/story/496936.html

46. South Africa v India, 4th ODI, Port Elizabeth. (January 21, 2011). Retrieved from http://www.espncricinfo.com/south-africa-v-india-2010/content/story/497680.html

47. Australia v England, 5th ODI, Brisbane. (January 30, 2011). Retrieved from http://www.espncricinfo.com/the-ashes-2010-11/content/story/498740.html

48. South Africa v India, 2nd ODI, Johannesburg. (January 15, 2011). Retrieved from http://www.espncricinfo.com/south-africa-v-india-2010/content/story/496853.html

49. K. Papineni, S. Roukos, T. Ward and W.-J. Zhu. (2001). BLEU: A Method for Automatic Evaluation of Machine Translation. *Technical Report RC22176(W0109-022), IBM Research Report.*

50. C. Y. Lin. (2004). ROUGE: A Package for Automatic Evaluation of Summaries. *In Proceedings of the Workshop on Text Summarization Branches Out. Post-Conference Workshop of ACL 2004, Barcelona, Spain.*

51. Australia v England, 4th ODI, Adelaide. (January 26, 2011). Retrieved from http://www.espncricinfo.com/the-ashes-2010-11/content/story/498309.html

52. P. Koehn. (2010). Statistical Machine Translation. *Cambridge University Press, Cambridge, UK.*

53. Australia v England, 2nd ODI, Hobart. (January 21, 2011). Retrieved from http://www.espncricinfo.com/the-ashes-2010-11/content/story/497617.html

54. Australia v England, 3rd ODI, Sydney. (January 23, 2011). Retrieved from http://www.espncricinfo.com/the-ashes-2010-11/content/story/497909.html

55. Australia v England, 6th ODI, Sydney. (February 02, 2011). Retrieved from http://www.espncricinfo.com/the-ashes-2010-11/content/story/499132.html

56. South Africa v India, 1st ODI, Durban. (January 12, 2011). Retrieved from http://www.espncricinfo.com/south-africa-v-india-2010/content/story/496451.html

57. South Africa v India, 3rd ODI, Cape Town. (January 18, 2011). Retrieved from http://www.espncricinfo.com/south-africa-v-india-2010/content/story/497291.html

58. South Africa v India, 5th ODI, Centurion. (January 23, 2011). Retrieved from http://www.prothom-alo.com/detail/date/2011-01-24/news/125953