Framework For Linguistic Applications Development (FLAD)

C.M. Liyanage, P.A.D. Chandana, D.A.B.P. Dodangoda, G.D.D. Kanchana, K. P. Hewagamage, A. R. Weerasinghe, Viraj Welgama

***Abstract*—Natural Language Processing (NLP) is a main area which emerged with the Artificial Intelligence. Text to Speech (TTS), Speech to Text (STT), Optical Character Recognition (OCR) and Language Translation are four main dimensions of Natural Language Processing software applications which are also known as Linguistic components. There are many service providers currently available providing the services of the previously mentioned four linguistic components. Each linguistic component also has different vendors providing the service. These service providers can be online or offline service providers. These service providers maintain different interfaces to expose their services which makes a burden to the linguistic application developers. Developers who want to use different linguistic components in one application will have to face a lot of difficulties in many ways such as installing services, configuring services, etc. Developers who want to add more advanced functionalities to their applications by combining two or more linguistic components will have to face many major problems in integrating these components in the same application as these components are not in the same platform to be integrated. Considering all these problems we propose a framework which brings all the above mentioned linguistic components to a common platform and exposes the services as web services through REST APIs. Further, this framework addresses the issue of combining the services of linguistic components by exposing REST APIs for the complex services which are made up of services of two or more linguistic components. Therefore developers only have to send requests to the REST APIs and handle the JSON responses sent by the system. Developers can create projects to access the services provided by the framework through the FLAD Console of the system. Developers can select the linguistic components and vendors of the selected linguistic components they need in their applications. When considering this framework from the technical aspect, expandable nature of this framework is an important achievement. This framework is designed in a such a way that a new linguistic component or a new instance of a linguistic component can be integrated to the framework with less overhead. When considering the overall nature of this framework it is clear that this will fill the gap between linguistic applications development and the linguistic application components which addresses the goal of this project that is to reduce the application development overhead through implementing a framework for linguistic application development. Additionally, the framework is not limited to the linguistic domain and can be used to integrate any pluggable component and expose the service through a common REST API.**

***Index Terms*—NLP, TTS, STT, OCR, Translation, Linguistic, REST API, framework, pluggable**

I. Introduction

The application development around linguistic components has become a necessity in current application development processes. The gap between the application development and integration, configuration of linguistic components has been a setback in the development of linguistic applications around the innovative ideas. The unavailability of a standard method to utilize the services of linguistic components leads application developers to rewrite the code for integrating and configuring the linguistic components. This violates basic software engineering principles like code reuse. Further, the integration patterns used may not be optimal which leads to poor performance of applications. Lack of an architecture that provides the modularity, flexibility, and efficiency for linguistic application development is a dearth in the software engineering domain.A framework which comprises an architecture that integrates all the linguistic components is a major requirement in this context. A framework enables developers to devote their time to meet software requirements rather than dealing with the more standard low-level details of providing a working system, thereby reducing overall development time.

The solution framework,

- Enables rapid application development as the framework handles the low-level details of components.
- Increases the application performance due to code reusability and internal architecture of the framework.
- Reduces the complexity of application code because of application programming interfaces and low dependency between the application code and the code of the linguistic component.
- Decreases the application size as the third party linguistic components libraries are implemented on framework side.

The design of a framework for linguistic application development supports the software engineering domain by addressing a commonly occurring issue in application development and support linguistic application developers all around the world

A. Background

The key idea about the project is to integrate all the linguistic components Text To Speech (TTS), Optical Character Recognition (OCR), Speech To Text (STT) and language translation in a framework and expose their functionalities as a RESTful web service. Figure 1 shows the abstract view of the system. The framework facilitates to utilize the services simply using the REST API for linguistic application developers, without considering about the configuration, integration and other low-level details of linguistic components.

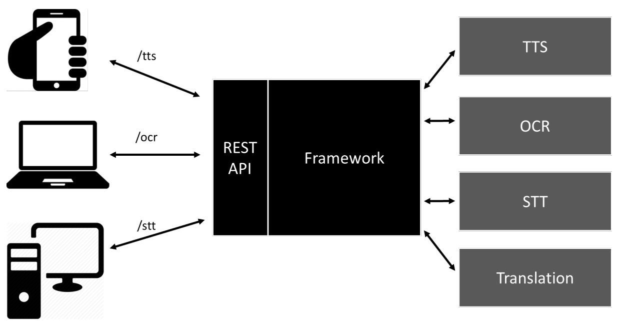

Fig. 1. Abstract view of the system

The core of the framework is implemented using Java programming language and Jersey which is a RESTful web services framework used to implement the RESTful web services. The back end consists all the third party libraries and servers related to linguistic components which are configured and integrated into the core of the framework. The architecture of the framework is designed in a way to support the plug and play architectural model. The plug and play model enables the addition of new linguistic components to the framework easily without affecting initial architecture. The framework architecture was the main research area of the project as there were no prior linguistic application frameworks or similar systems that motivate the design of the architecture.The plug and play model solves the issue of integrating heterogeneous technology into the framework. Available linguistic components are developed by many vendors using different programming languages and different technologies. For an example, the Google Speech API is a RESTful web service which exposes the STT service while the CMU Sphinx is an STT service which provides a standalone library. The framework should be able to integrate both the services irrespective of the vendor or technology used. Further, the framework should support the functionality of plugging a new STT component with minimal effort. The framework is designed following software engineering best practices and different combinations of design patterns to achieve the ideal architecture for the framework.

II. Related Work

A. *WSO2 ESB*

The component architecture of the WSO2 ESB [1] is built on the Apache Synapse project, which is built using the Apache Axis2 project. The initial foundation for the framework was inspired by the WS02 ESB architecture. ESB supports web services and support for XML. Message transformation feature leads to the introduction of Message Builder and Formatter component of FLAD. Logging and monitoring features in the ESB and integration patterns in WSO2 ESB leads to the foundation of the architecture in the FLAD framework.

B. *Google Cloud Services*

Google Cloud Services provides Google Cloud Speech API [2], Google Cloud Vision API [3], and Google Translation API [4] as RESTful web services. The architecture and design decisions of Google Cloud Services were considered in the design and development phrases.

III. Approach

A. *System design and architecture of the FLAD back end core framework*

The entire system is mainly divided into two subsystems as front end FLAD Console and back end FLAD core. The main architecture of FLAD core is divided into three sub architectures considering different functionalities of the system. Different architectures are required to cater different core functionalities supplied by the back end and thus divided into three sub architectures as,

1) Communication architecture
2) Structure architecture
3) Deployment architecture

Figure 2 shows the main architecture segregation and the specific architectural styles used in each category. The main research area of the project was to introduce best architectural styles into the framework. The background study guided into selecting the Service Oriented Architecture (SOA) as the communication architecture, Component Architecture for the structure architecture and Client-Server Architecture as the deployment architecture .

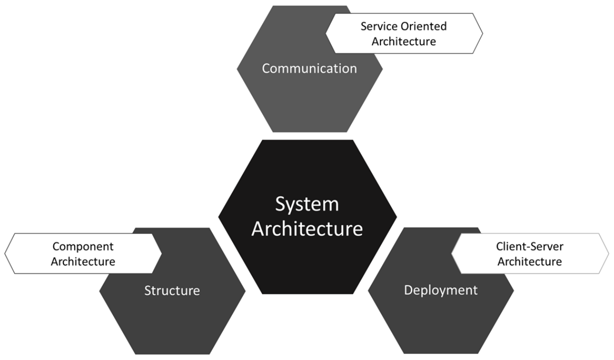

Fig. 2. System architecture of FLAD

1) Communication architecture: The communication architecture is implemented using Service Oriented Architecture (SOA). SOA enables the communication of services of different application components through a communication protocol. According to SOA definition, a exposed service should contain four properties [6]. These properties are followed in the implementation of FLAD.

1) The service should logically represents a business activity with a specified outcome
   FLAD endpoints serve their specific services and it directly represents the business value of the service. For

an example /tts endpoint directly represents the TTS service entity and results the intended outcome.

2) The service is self-contained
FLAD provides simple services which are self-contained and independent. Everything that is required by the service is separated from other services and exposed via the REST API.
3) Service is a black box for its customers
The service provided to the customers just gives the intended response to the customer without exposing the interior logic. The logic is handled inside the framework and the user simply gets the service according to the preferences selected in the creation of service API project using FLAD Console.
4) It may consist of other underlying services
The FLAD complex services which are sequences, validations, message builder and formatter use other underlying services internally when executing. FLAD implements services in a loosely coupled manner and modular approach which lean towards the SOA architectural pattern. REST architectural style reflects the SOA in the framework.This enables exposing functional building blocks accessible over HTTP providing the solution to represent new components or standalone systems. SOA based systems allow the utilization of services independently of development technologies and platforms such as Java, C++ and etc. Also by embracing SOA style motivates the use of well-defined interfaces and supports the component-based architecture which is the selected structural architecture of FLAD.

2) Component architecture: The main structure of the project is laid on top of the component architecture which is highly compatible with SOA. The definition of the different components in the system and binding of them was part of the major research area of the project. The component-based architecture should be able to support the plug and play model of the system where any linguistic component can be added or removed from the system with minimal effect to the back end framework and this should support the extendability of the framework.Figure 3 shows the components defined in our framework which provide the fundamental functionalities by communicating via well-defined interfaces.

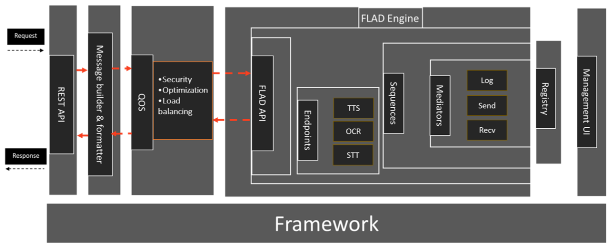

Fig. 3. Component architecture of FLAD

The component architecture of FLAD is described using an example, Assume an application developer wants to use the STT service from the framework.

- First, the user makes a request to /stt endpoint in the REST API with the specified JSON payload.
- Then the users request has to be parsed with a JSON parser and convert to the internal data representation using the Message builder and formatter.
- After parsing the payload, it is sent to the QOS components for validation and optimizations before sending to the real endpoint service. In an STT request, the mono stream of the audio file is checked and if the audio is on stereo it is converted to a mono stream and then encoded using Base64 encoding scheme.
- After preprocessing the request payload, it is sent to the internal FLAD API written in Java. The STT request is validated first and the STT instance is invoked dynamically with the help of factory and abstract factory design patterns. If the user has selected Google STT service in project creation phase then it has to invoke the Google STT instance and if the user has selected any other STT service it is dynamically detected with the help of data layer and then the initiation takes place.
- The FLAD internal API decides which endpoint to invoke and endpoints shown in Figure 4.2 are instances of STT, TTS, Trans or OCR. For example, an endpoint in the framework related to this example is Google speech API.
- Another important component is the Sequence component which is a combination of different send and receive calls of endpoints. For example seq2 in our framework invokes the sequence of OCR $\rightarrow$ Translation $\rightarrow$ TTS. This sequence is capable of reading the content of an image and output the speech in a specified translation. The content of a German signboard can be heard in English using the above sequence.
- Mediators are the basic functional units in the framework. Currently, there are three mediators in the framework as Send, Receive and Logging. Send mediator is responsible for sending messages between different components of the framework and the Receive mediators are responsible for dealing with the return values of the components. The Log mediator logs every action in the framework to support monitoring.

REST API
Rest API exposes the resources and functionalities of the framework and enables basic HTTP protocols to utilize the framework services.

Message builders and formatters
Message builder is used to process incoming and outgoing payload data to XML or JSON. There are two basic functional units as message parsers and formatters. The message formatter is used to build the outgoing response from the message back into its original format. Message parsers and builders convert the payload data to required data formats of endpoints.

QoS component
The Quality of Service (QoS) component implements security, load balancing, and optimization. Basic message preprocessing is done with this component.

Endpoints

An endpoint defines an external destination for a message. An endpoint can connect to any external service after configuring it with any attributes or semantics needed for communicating with that service. TTS endpoint, OCR endpoint, STT endpoint and Translation endpoint are the basic endpoint types of the framework.

Mediators

Mediators are individual processing units that perform a specific function, such as sending, transforming, logging or filtering messages.

Sequences

A sequence is a set of mediators organized into a logical flow. The concept of sequence is an optimization technique used to reduce the successive incoming and outgoing calls to the framework and improve performance.

Registry

A registry is a content store and metadata repository. The framework cache required data in primary memory for fast retrieval. For example, the database connections are pooled after the first request so any request made after gets the advantage of fast retrieval of data and factory objects are cached in the Registry component after first initialization using the Service Locator design pattern.

Management UI and FLAD Console

The Management console provides a Graphical User Interface (GUI) that allows the framework administrator to monitor and manage the framework functionalities. The FLAD Console is the management UI which is implemented in React and deployed in a NodeJS server. FLAD Console enables users to specify required services through a project creation

3) Deployment architecture : Distributed application structure, support the future scalability and load balancing in presence of high request growth. The frameworks functionality exposed in a way to be utilized by any client globally and server can be replicated with the future expansion according to this model. The use of distributed client-server architecture enables four services of the framework to be distributed. The front end FLAD console service, back end framework core, MongoDB database and each endpoint service can be distributed using this model with the help of well-defined interfaces.The front end FLAD console is deployed in a separate Node JS server and can be deployed in a different server. It communicates with back end server through REST API calls in a loosely coupled manner. The back end is deployed in a java based Jetty server and is independent to the front end server. Both the front end server and back end server communicate over well-defined interfaces. The database server can be deployed on several servers and currently, the MongoDB server is on the same server machine where the back end framework server is deployed. The deployment of each linguistic components in separate servers is possible according to this model. For example, MaryTTS server is running on a different server and can be scaled up with the huge growth of requests.

4) The multi-layered architecture of FLAD: The layered architecture enables to understand the system in a concise way where all the core functional layers of the system are identified. Figure 4 shows the multi-layered architecture of the FLAD system. This representation allows easy understanding of the idea behind the scalability and extensibility of the system.

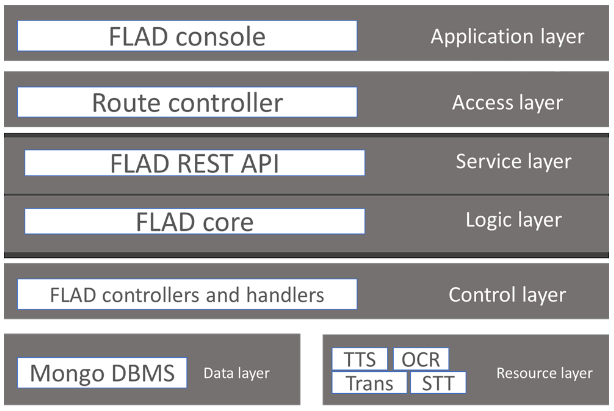

Fig. 4. The multi-layered architecture of FLAD

5) Core Component architecture of FLAD : This section describes the design patterns used in the core component architecture and the decision points which lead to use them. The design patterns are applied in both the front end FLAD console and the back end framework. Applied design patterns are integrated in order to overcome the commonly occurring issues in the system.

Figure 5 shows the class diagram architecture of the system. The structure of FLAD system design is based on Component Architecture.

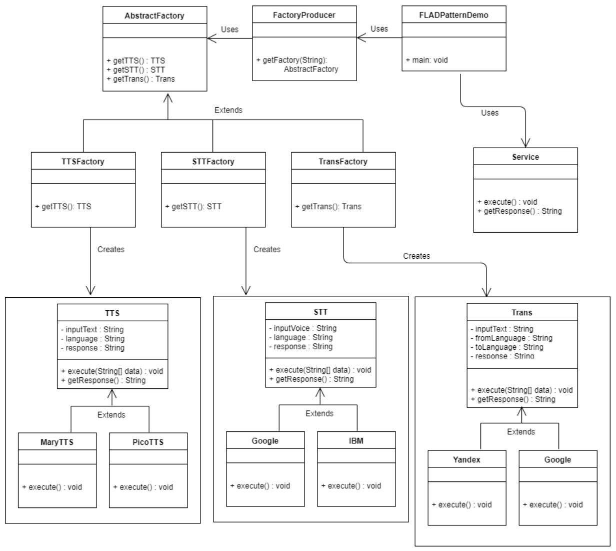

Fig. 5. Class diagram of FLAD core

This diagram consists of service components and factories that generate above components and relation of their particular instances. Abstract Factory Pattern [7], Builder Pattern [8], and

Command Pattern [9] integrated in order to achieve the plug and play architecture.

Abstract Factory Pattern involves installing and instantiating each component in the environment. Factory Producer provides interface for the Abstract Factory to communicate with front view, In this context, it will be the REST API which accesses by users.

Builder Pattern uses when the components or subcomponents in the architecture dependant on each other. When adding a new service such as MaryTTS, Google or IBM, system will create an object of the given service and inject it to their parent objects. As shown in Figure 5, the parent objects are the service initiators named TTS, STT, OCR, and Trans. Each of this service objects depend on these service initiators.

Command Pattern gives components a consistent entry point, allowing them to be readily swapped in and out. Each of service initiators feeds service objects assigned to them through execute method. This also follows dependency injection when passing of a dependency to a dependent service object that would use it. All the service objects located at the bottom of the diagram designed as a dependant to the service initiator objects. These plugged service objects will execute as single services on their own, but on the inside of the structure, they are coupled with service objects which are generated using the particular factory. When new service introduces, it is required to link that service to the relevant service initiators.

The Factory and Abstract Factory design patterns are combined in the core of the system to achieve the plug and play model. This abstract factory design pattern further facilitates the dynamic initiation of services at runtime in a flexible manner. The Abstract factory class delegates the responsibility of object instantiation to TTSFactory, STTFactory, OCRFactory, and TransFactory. Those Factories use inheritance for the object instantiation of real service dynamically. These factories are the key components of achieving the plug and play model for the framework.

Figure 5 shows how inheritance and encapsulation are used in the class diagram. The component class is the base class of each integrated service component. It defines all the common features and behaviors that expected to retrieve from particular instances which are yet to plug into the architecture. The execute method which is defined as public describes the internal logic of each instance. These component instances are the actual integrated service classes which interact directly with parent service components in order to provide their services to the upper level. Figure 5 contains MaryTTS and PicoTTS, as TTS subclasses. These instances may have their own additional features. Basically, each instance supposed to provide minimum required features defined in parent components. All the internal logic of each instance describes in the execute method which is designed to invoke dynamically.

These steps should be followed to add a new component to the system,

- Define a class with the name of the new component.
- Inherit the superclass which the new component belongs to.
- Override the execute method and write the interior logic specific to the component.
- Add the entry name in the related Factory class.

This approach leads the minimal change to the framework functionality and this can be further simplified using XML to define the name of the new component at runtime using Management UI. However, writing of the implementation logic in the execute method of the component is unavoidable as different linguistic services provide different APIs.

6) Service Locator Pattern: Service locator pattern caches the objects in creation in order to achieve performance enhancement [5]. The use of this pattern enabled to cache the object references after its first creation into separate HashMaps in Java. The caching is done at the Cache class as shown in Figure 6. The service locator first looks at the cache and if there is no reference in the hash maps then creates the objects and cache it before sending. This improves the performance of the object creation time via caching the references in the Registry component of the framework by avoiding object creation at each request.

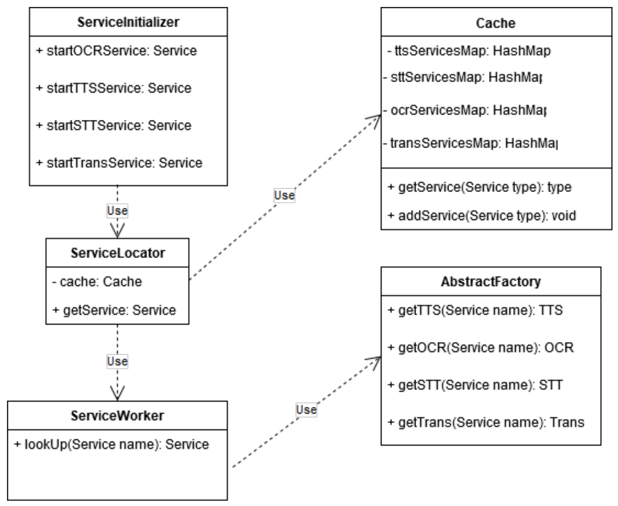
ServiceInitializer
+ startOCRService: Service
+ startTTSService: Service
+ startSTTService: Service
+ startTransService: Service
Use
ServiceLocator
- cache: Cache
+ getService: Service
Use
ServiceWorker
+ lookUp(Service name): Service
Use
Cache
- ttsServicesMap: HashMap
- sttServicesMap: HashMap
- ocrServicesMap: HashMap
- transServicesMap: HashMap
+ getService(Service type): type
+ addService(Service type): void
Use
AbstractFactory
+ getTTS(Service name): TTS
+ getOCR(Service name): OCR
+ getSTT(Service name): STT
+ getTrans(Service name): Trans

Fig. 6. Service locator pattern of FLAD core

IV. IMPLEMENTATION

A. The back end development

The back end framework is implemented in Java EE and Maven as the software project management and comprehension tool. In the development phase, the requirement of a full-featured lightweight server that can be embedded and handles many concurrent requests wasa key concern. Jetty server consumes raw resources from the server machine and can be embedded in Maven Project Object Model (POM). Jetty is more developer friendly and supports many developer options and commands. It can be deployed, launched and restarted quickly than other servers like Apache Tomcat or Wildfly server.

Jersey framework which follows JAX-RS specifications has been used to implement the REST API in our framework. Maven project management tool was used mainly for the dependency management and as the build tool of the project. Maven helps the inclusion of necessary libraries needed for the framework and building and compilation of the project.

1) Message builder and formatter implementation: Message builder and formatter is one of the main components in the component architecture of the framework as shown in Figure 7. The message builder and formatter is a message parser to and from the REST API of the framework. The framework supports endpoint for the services developed by many vendors which accept different inputs to process and return different outputs. In this example Message Builder and Formatter enables a common input format for all the OCR services and defines a common format for all the responses output by the OCR components. This heterogeneity of inputs and outputs of all the linguistic components are handled by the message builder and formatter component.

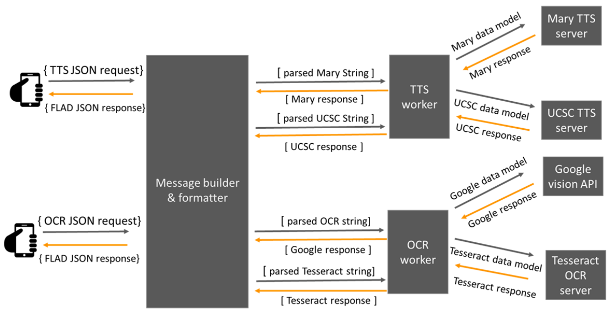

Fig. 7. Function of Message Builder and Formatter

V. Prepare Your Paper Before Styling

Before you begin to format your paper, first write and save the content as a separate text file. Complete all content and organizational editing before formatting. Please note sections V-A–V-E below for more information on proofreading, spelling and grammar.

Keep your text and graphic files separate until after the text has been formatted and styled. Do not number text heads—LaTeX will do that for you.

A. Abbreviations and Acronyms

Define abbreviations and acronyms the first time they are used in the text, even after they have been defined in the abstract. Abbreviations such as IEEE, SI, MKS, CGS, ac, dc, and rms do not have to be defined. Do not use abbreviations in the title or heads unless they are unavoidable.

B. Units

- Use either SI (MKS) or CGS as primary units. (SI units are encouraged.) English units may be used as secondary units (in parentheses). An exception would be the use of English units as identifiers in trade, such as "3.5-inch disk drive".
- Avoid combining SI and CGS units, such as current in amperes and magnetic field in oersteds. This often leads to confusion because equations do not balance dimensionally. If you must use mixed units, clearly state the units for each quantity that you use in an equation.
- Do not mix complete spellings and abbreviations of units: "Wb/m^2" or "webers per square meter", not "webers/m^2". Spell out units when they appear in text: ". . . a few henries", not ". . . a few H".
- Use a zero before decimal points: "0.25", not ".25". Use "cm^3", not "cc".)

C. Equations

Number equations consecutively. To make your equations more compact, you may use the solidus (/), the exp function, or appropriate exponents. Italicize Roman symbols for quantities and variables, but not Greek symbols. Use a long dash rather than a hyphen for a minus sign. Punctuate equations with commas or periods when they are part of a sentence, as in:

$$a + b = \gamma \tag{1}$$

Be sure that the symbols in your equation have been defined before or immediately following the equation. Use "(1)", not "Eq. (1)" or "equation (1)", except at the beginning of a sentence: "Equation (1) is . . ."

D. LaTeX-Specific Advice

Please use "soft" (e.g., `\eqref{Eq}`) cross references instead of "hard" references (e.g., `(1)`). That will make it possible to combine sections, add equations, or change the order of figures or citations without having to go through the file line by line.

Please don't use the `{eqnarray}` equation environment. Use `{align}` or `{IEEEeqnarray}` instead. The `{eqnarray}` environment leaves unsightly spaces around relation symbols.

Please note that the `{subequations}` environment in LaTeX will increment the main equation counter even when there are no equation numbers displayed. If you forget that, you might write an article in which the equation numbers skip from (17) to (20), causing the copy editors to wonder if you've discovered a new method of counting.

BIBTeX does not work by magic. It doesn't get the bibliographic data from thin air but from .bib files. If you use BIBTeX to produce a bibliography you must send the .bib files.

LaTeX can't read your mind. If you assign the same label to a subsubsection and a table, you might find that Table I has been cross referenced as Table IV-B3.

LaTeX does not have precognitive abilities. If you put a `\label` command before the command that updates the counter it's supposed to be using, the label will pick up the last counter to be cross referenced instead. In particular, a `\label` command should not go before the caption of a figure or a table.

Do not use `\nonumber` inside the `{array}` environment. It will not stop equation numbers inside `{array}` (there won't be any anyway) and it might stop a wanted equation number in the surrounding equation.

E. Some Common Mistakes

- The word “data” is plural, not singular.
- The subscript for the permeability of vacuum μ_0, and other common scientific constants, is zero with subscript formatting, not a lowercase letter “o”.
- In American English, commas, semicolons, periods, question and exclamation marks are located within quotation marks only when a complete thought or name is cited, such as a title or full quotation. When quotation marks are used, instead of a bold or italic typeface, to highlight a word or phrase, punctuation should appear outside of the quotation marks. A parenthetical phrase or statement at the end of a sentence is punctuated outside of the closing parenthesis (like this). (A parenthetical sentence is punctuated within the parentheses.)
- A graph within a graph is an “inset”, not an “insert”. The word alternatively is preferred to the word “alternately” (unless you really mean something that alternates).
- Do not use the word “essentially” to mean “approximately” or “effectively”.
- In your paper title, if the words “that uses” can accurately replace the word “using”, capitalize the “u”; if not, keep using lower-cased.
- Be aware of the different meanings of the homophones “affect” and “effect”, “complement” and “compliment”, “discreet” and “discrete”, “principal” and “principle”.
- Do not confuse “imply” and “infer”.
- The prefix “non” is not a word; it should be joined to the word it modifies, usually without a hyphen.
- There is no period after the “et” in the Latin abbreviation “et al.”.
- The abbreviation “i.e.” means “that is”, and the abbreviation “e.g.” means “for example”.

An excellent style manual for science writers is [12].

F. Authors and Affiliations

The class file is designed for, but not limited to, six authors. A minimum of one author is required for all conference articles. Author names should be listed starting from left to right and then moving down to the next line. This is the author sequence that will be used in future citations and by indexing services. Names should not be listed in columns nor group by affiliation. Please keep your affiliations as succinct as possible (for example, do not differentiate among departments of the same organization).

G. Identify the Headings

Headings, or heads, are organizational devices that guide the reader through your paper. There are two types: component heads and text heads.

Component heads identify the different components of your paper and are not topically subordinate to each other. Examples include Acknowledgments and References and, for these, the correct style to use is “Heading 5”. Use “figure caption” for your Figure captions, and “table head” for your table title. Run-in heads, such as “Abstract”, will require you to apply a style (in this case, italic) in addition to the style provided by the drop down menu to differentiate the head from the text.

Text heads organize the topics on a relational, hierarchical basis. For example, the paper title is the primary text head because all subsequent material relates and elaborates on this one topic. If there are two or more sub-topics, the next level head (uppercase Roman numerals) should be used and, conversely, if there are not at least two sub-topics, then no subheads should be introduced.

H. Figures and Tables

a) Positioning Figures and Tables: Place figures and tables at the top and bottom of columns. Avoid placing them in the middle of columns. Large figures and tables may span across both columns. Figure captions should be below the figures; table heads should appear above the tables. Insert figures and tables after they are cited in the text. Use the abbreviation “Fig. 8”, even at the beginning of a sentence.

TABLE I
TABLE TYPE STYLES

Table Head	**Table Column Head**		
	Table column subhead	***Subhead***	***Subhead***
copy	More table copy[a]		

[a]Sample of a Table footnote.

Fig. 8. Example of a figure caption.

Figure Labels: Use 8 point Times New Roman for Figure labels. Use words rather than symbols or abbreviations when writing Figure axis labels to avoid confusing the reader. As an example, write the quantity “Magnetization”, or “Magnetization, M”, not just “M”. If including units in the label, present them within parentheses. Do not label axes only with units. In the example, write “Magnetization (A/m)” or “Magnetization {A[m(1)]}”, not just “A/m”. Do not label axes with a ratio of quantities and units. For example, write “Temperature (K)”, not “Temperature/K”.

ACKNOWLEDGMENT

The preferred spelling of the word “acknowledgment” in America is without an “e” after the “g”. Avoid the stilted expression “one of us (R. B. G.) thanks ...”. Instead, try “R. B. G. thanks...”. Put sponsor acknowledgments in the unnumbered footnote on the first page.

References

Please number citations consecutively within brackets [6]. The sentence punctuation follows the bracket [7]. Refer simply to the reference number, as in [8]—do not use "Ref. [8]" or "reference [8]" except at the beginning of a sentence: "Reference [8] was the first . . ."

Number footnotes separately in superscripts. Place the actual footnote at the bottom of the column in which it was cited. Do not put footnotes in the abstract or reference list. Use letters for table footnotes.

Unless there are six authors or more give all authors' names; do not use "et al.". Papers that have not been published, even if they have been submitted for publication, should be cited as "unpublished" [9]. Papers that have been accepted for publication should be cited as "in press" [10]. Capitalize only the first word in a paper title, except for proper nouns and element symbols.

For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation [11].

References

[1] "Wso2 documentation." [Online]. Available: https://docs.wso2.com/display/ESB490/Architecture

[2] "Speech api - speech recognition google cloud platform." [Online]. Available: https://cloud.google.com/speech/

[3] "Vision api - image content analysis google cloud platform." [Online]. Available: https://cloud.google.com/vision/

[4] "google cloud translation api documentation translation api google cloud platform." [Online]. Available: https://cloud.google.com/translate/docs/

[5] "the service locator pattern." [Online]. Available: https://msdn.microsoft.com/en-us/library/ff648968.aspx

[6] "Service-oriented architecture standards", [Online]. Available: http://www.opengroup.org/standards/soa

[7] "Abstract Factory Pattern",[Online]. Available:http://www.oodesign.com/abstract-factory-pattern.html

[8] "Builder Pattern",[Online]. Available: http://www.oodesign.com/builder-pattern.html

[9] "Command Pattern", [Online]. Available: http://www.oodesign.com/command-pattern.html

[10] R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.

[11] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[12] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.