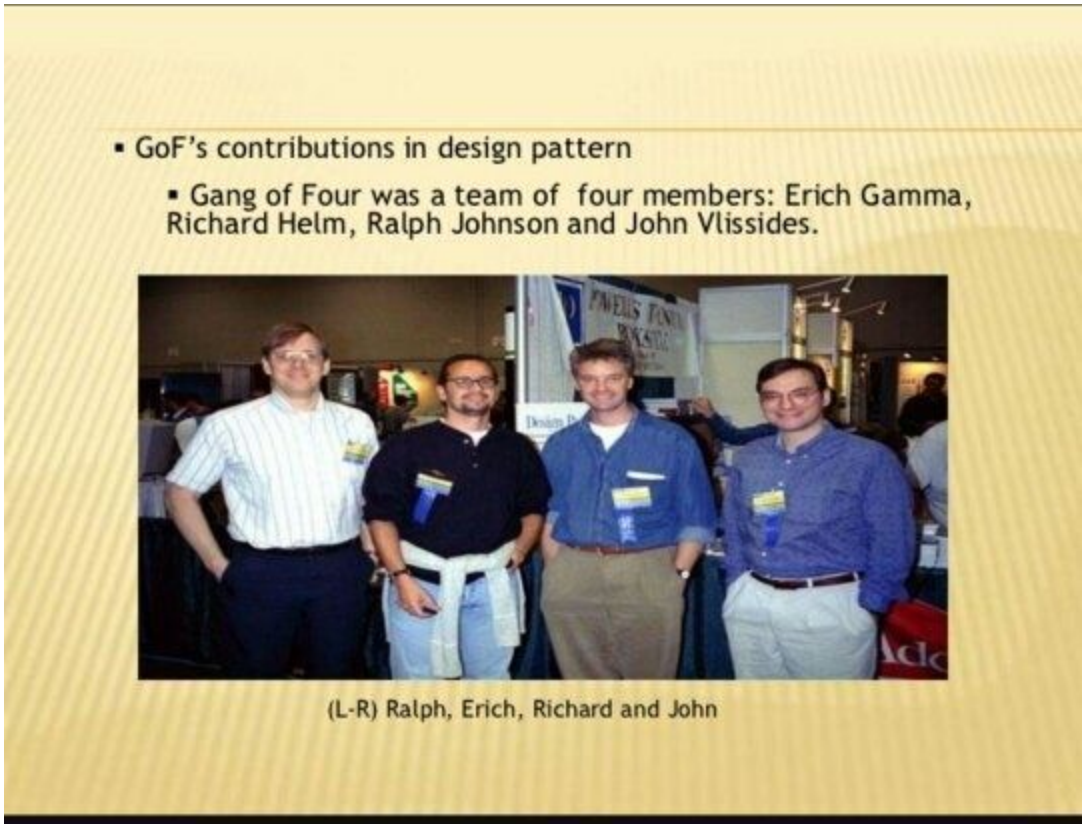
Reactive Programming with the Interpreter pattern, automated parsers.

Dr Bheemaiah, Anil Kumar, A.B Seattle W.A 98125
miyawaki@yopmail.com

Abstract:
Reactive programming was popularized by Netflix, with a need for quick stream based processing, for a unified framework of event programming, with JavaX and the introduction of Beans for events and streams, a broad spectrum of applications for data driven, enterprise architectures, in the context of SaaS for a framework for many applications from ubiquitous computing to IIoT are described.
This paper is on the formal automation of parsing in the three world scenario of CAS, OOPS and NLP, with a broad definition of algebras, programming languages and natural languages, for stream based event parsing.
Keywords: CAS, OOPS, NLP, IIOT, ubiquitous computing, JavaX, Netflix, reactive Computing

What:
The interpreter design pattern is embedded in RxJava and RxJS to enable the parsing of arbitrary text files. We illustrate this with a SaaS based CDSS and Expert system, with the interpreter for parsing and automated persistence of Case Studies.
How:
We use an integrated SaaS based expert system using AWS for persistence and text mining. Pre processing is done by a RxJS based client web worker for pre-processing and parsing of case studies and persistence on AWS with Taskoids for rule creation and persistence.
Why:
The MetaECHO, of the University Of New Mexico is a knowledge democratization movement, to make available specialist practice based expert knowledge through online knowledge sharing and training. In an attempt to make specialist practice available through expert system based CDSS systems, Vayu Vaidya, a research organization, based in Seattle, WA, USA, and HIHT Dehradun, India, is in the process of creating AWS based expert systems to persist all the case studies, presented through the TeleECHO and MetaECHO program.

So What:
The expert system, with a conversational UI interface to a CDSS system, would make specialist knowledge available 24/7 to the medical community, coupled with the preventive medicine based risk evaluation and medical infotainment information available through Amazon Alexa and IFDB.



An answer to a question on Quora, to Calvin, from Calvin and Hobbes, on why architecture is so important. For more answers consider stack overflow or Quora.("Where Did Design Patterns Come From?" n.d.)

Summary:

Main Points:
The creation of OOPS, CAS and NLP parsers with the interpreter on character streams , parsing of event streams with integration of the Interpreter pattern in Rx Java.(Maglie 2016; Davis 2019b, [a] 2019)
Illustration with examples, document as text stream, for semantic segmentation and automatic markup.
An example of MetaEcho text streams.

Applications:
Security, SQL attack detection, data mining and semantic segmentation, event programming.

Introduction.

The gang of four(Hunt 2013) defines the use of generalized design patterns in OOPS architectures, we include the Interpreter pattern(Badenhorst 2017; Sarcar 2018; Joshi 2016) in streams and event streams in reactive programming.

Problem Definition.

Reactive programming was defined as the addition of the iterator and observer patterns to stream data structures.
We add the Interpreter for parsing, arbitrary inputs on stream beans and event beans.(Bheemaiah, n.d.)

Background.

<original-contribution>

Formal Definitions:

Interpreter is defined as:

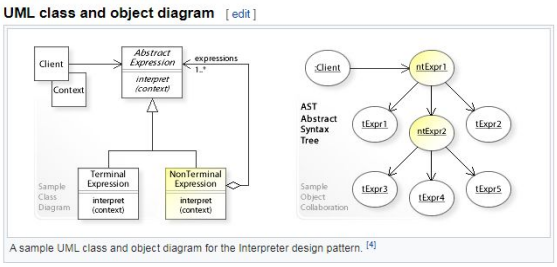


Fig: From Wikipedia.(Contributors to Wikimedia projects 2003)

In Java,

```
public class Interpreter {
  @FunctionalInterface
  public interface Expr {
    int interpret(Map<String,
Integer> context);

    static Expr number(int number)
{
      return context -> number;
    }

    static Expr plus(Expr left,
Expr right) {
      return context ->
left.interpret(context) +
right.interpret(context);
    }

    static Expr minus(Expr left,
Expr right) {
      return context ->
left.interpret(context) -
right.interpret(context);
    }

    static Expr variable(String
name) {
      return context ->
context.getOrDefault(name, 0);
    }
  }
...
```

Here we consider a parser for an expression, generalizable in an NLP-OOPS-CAS equivalence. An NLP module can be added with a parse tree as part of the Interpreter pattern, as in processor JS.

Here is an example of a simple arithmetic expression from wikipedia.(Contributors to Wikimedia projects 2003)

```
private static Expr
parseToken(String token,
ArrayDeque<Expr> stack) {
    Expr left, right;
    switch(token) {
    case "+":
      // it's necessary to remove
first the right operand from the
stack
      right = stack.pop();
      // ..and then the left one
      left = stack.pop();
      return Expr.plus(left,
right);
    case "-":
```

```
      right = stack.pop();
      left = stack.pop();
      return Expr.minus(left,
right);
    default:
      return Expr.variable(token);
    }
  }
  public static Expr parse(String
expression) {
    ArrayDeque<Expr> stack = new
ArrayDeque<Expr>();
    for (String token :
expression.split(" ")) {
      stack.push(parseToken(token,
stack));
    }
    return stack.pop();
  }
…
```

And the main()…..

```
...
  public static void main(final
String[] args) {
    Expr expr = parse("w x z -
+");
    Map<String, Integer> context =
Map.of("w", 5, "x", 10, "z", 42);
    int result =
expr.interpret(context);
    System.out.println(result);
// -27
  }
}
```

Similar parsers can be meta-defined with code-generator modules, with even template definitions.

The application to streambeans and eventbeans is immediate with the direct interpretation of streambean data, for event parsing and the parsing of arbitrary streams, with or without lazy evaluation.

DOSe from Text Segmentation(Contributors to Wikimedia projects 2006)

DOSe or Document Object Segmentation models are defined as text streams with semantic markups, models and semantic categories can be made with parsers from text streams, with object categories as markup or tags. This is useful in text mining.
Consider a text document of a best practice case from MetaEcho, for a text document M, we can define object categories [prognosis, case ID, heuristic dump]

We consider the example code in Java for this on a medical text stream from a MetaEcho presentation.

```
 public static Expr
parse(StreamBean TextIO) {
    ArrayDeque<Expr> stack = new
ArrayDeque<Expr>();
    for (String token :
TextIO.split(ParseGenerator(flex_r
ules)) {
      stack.push(parseToken(token,
stack));
    }
    return stack.pop();
  }

public static
ParseGenerator(String[] flex_rules
){}
```

...

</original-contribution>

Discussion.

Parsers, enable a wide interpretation of languages, from languages with grammars and an alphabet to continuous languages, much like ambient and background light. The design of a common framework and a programming paradigm for a OOPS, NLP and CAS type reactive framework, is with the Interpreter pattern, as shown in the example from wikipedia.
The use of text mining and parsing of semantic patterns is illustrated for markup in text dumps, with an example from a MetaECHO presentation. This work is particularly relevant in countries in Europe and Asia, where expert systems are relevant.

Future Work.

We discuss the use of Reactive Parsing in data mining in the case of waveform discretization, on EEG data and audio streams, available as block data in JSON.

References.

Badenhorst, Wessel. 2017. “Interpreter Pattern.” *Practical Python Design Patterns*. https://doi.org/10.1007/978-1-4842-2680-3_12.

Bheemaiah, Anil Kumar. n.d. “Enterprise Java Beans with ReactiveX.” https://doi.org/10.31224/osf.io/a8d3x.

Contributors to Wikimedia projects. 2003. “Interpreter Pattern - Wikipedia.” Wikimedia Foundation, Inc. January 3, 2003. https://en.wikipedia.org/wiki/Interpreter_pattern.

———. 2006. “Text Segmentation - Wikipedia.” Wikimedia Foundation, Inc. March 4, 2006. https://en.wikipedia.org/wiki/Text_segmentation.

Davis, Adam L. 2019a. “Reactive Streams in Java.” https://doi.org/10.1007/978-1-4842-4176-9.

———. 2019b. “RxJava.” *Reactive Streams in Java*. https://doi.org/10.1007/978-1-4842-4176-9_4.

Hunt, John. 2013. “Gang of Four Design Patterns.” *Scala Design Patterns*. https://doi.org/10.1007/978-3-319-02192-8_16.

Joshi, Bipin. 2016. “Behavioral Patterns: Chain of Responsibility, Command, Interpreter, and Iterator.” *Beginning SOLID Principles and Design Patterns for ASP.NET Developers*. https://doi.org/10.1007/978-1-4842-1848-8_7.

Maglie, Andrea. 2016. “ReactiveX and RxJava.” *Reactive Java Programming*. https://doi.org/10.1007/978-1-4842-1428-2_1.

Sarcar, Vaskaran. 2018. “Interpreter Pattern.” *Design Patterns in C#.* https://doi.org/10.1007/978-1-4842-3640-6_23.

“Where Did Design Patterns Come From?” n.d. Quora. Accessed October 11, 2019. https://www.quora.com/Where-did-design-patterns-come-from.