20CYS312 - Principles of Programming Languages
Exploring Programming Paradigms

Assignment-01

Presented by G H Hem Sagar
CB.EN.U4CYS21016
TIFAC-CORE in Cyber Security
Amrita Vishwa Vidyapeetham, Coimbatore Campus

Feb 2024

AMRITA
VISHWA VIDYAPEETHAM

Outline

Intro to Reactive Paradigm

Reactive programming is a programming paradigm that focuses on constructing responsive and robust software applications that can handle *asynchronous* data streams and *change propagation*, allowing developers to create scalable and more easily maintainable applications that can adapt to a dynamic environment.

- In a reactive system, data flows through streams, which can be thought of as sequences of events over time.
- A reactive system is basically made up of **observables**, **observers** and **schedulers**.

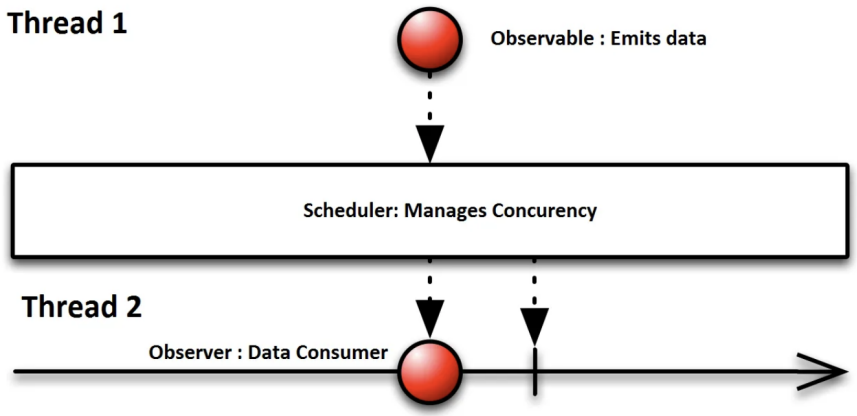


Figure: Reactive streams

Principles of Reactive Paradigm

1. **Data Streams** :
   - They represent sequences of events that occur over time, such as user inputs, API calls, or sensor readings.
2. **Observables** :
   - Observables are objects that represent data streams.
   - They provide a way to create, manipulate, and subscribe to data streams in a reactive system.
3. **Observers** :
   - They define the actions to be taken when new data arrives, or an error occurs.
   - Observers can be considered the consumers of the data provided by observables.
4. **Operators / Schedulers** :
   - Operators are functions that allow developers to manipulate and transform data streams.
   - They can be used to filter, merge, or modify data streams, enabling the creation of new streams that cater to specific needs.

Benefits and Characteristics of Reactive Programming

Benefits of Reactive Programming:

- They are inherently more **scalable** than traditional imperative systems, which rely on blocking calls and threads to manage concurrency.
- Reactive programming promotes a declarative style, this approach leads to cleaner and more **maintainable** code, making it easier to understand, modify, and troubleshoot.
- Reactive systems are built to handle asynchronous events and can respond to changes more quickly making them more **responsive** and adaptive to dynamic environments.
- Reactive programming enables more robust **error handling**, as errors are propagated through the data stream and can be handled centrally.

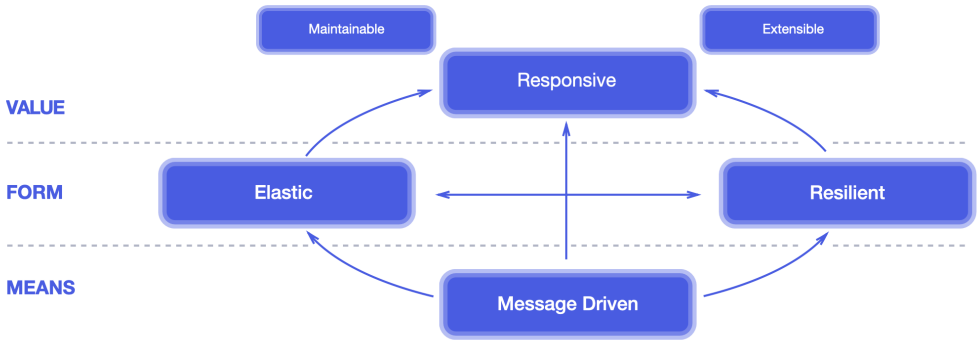
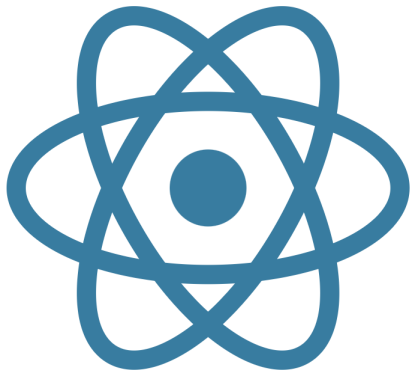


Figure: Characteristics of Reactive System

ReactJS - An Intro

ReactJS, commonly known as *React*, is an open-source JavaScript library developed and maintained by Facebook(META). It is specifically designed for building user interfaces (UIs) for single-page applications where user interactions are dynamic and require efficient updates to the UI.
Key features of ReactJS:

- Component Based Architecture - it is built around the concept of components, which are reusable and self-contained units of UI.
- Virtual DOM - used to efficiently update the browser DOM.
- Declarative Syntax - makes the code predictable and debug friendly.
- React Hooks - enables the use of state and other react functions in functional components.



Reactive Paradigm in ReactJS

- **ReactJS** follows the principles of **reactive** programming by using a virtual DOM. When the state of a component changes, React efficiently updates the virtual DOM and then selectively updates the actual DOM to reflect those changes.
- React's virtual DOM is a lightweight in-memory representation of the actual DOM. It allows React to calculate the most efficient way to update the UI, resulting in improved performance by minimizing direct manipulations to the browser's DOM.

In ReactJS, reactive programming is expressed through the use of *components*, *state*, and the *virtual DOM*. The following example illustrates a simple counter component:

counter.jsx > ...

```jsx
class Counter extends React.Component {
  constructor(props) {
    super(props);
    this.state = { count: 0 };
  }

  render() {
    return (
      <div>
        <p>{this.state.count}</p>
        <button onClick={() => this.setState({ count: this.state.count + 1 })}>
          Increment
        </button>
      </div>
    );
  }
}
```

Figure: A simple counter component

Case Study: Instagram

Instagram, one of the world's largest social media platforms, utilizes **ReactJS** for building its web application. React's component-based architecture and efficient UI updates make it well-suited for the dynamic and interactive nature of social media platforms.

How ReactJS is Used:

- *Component-Based UI*: Instagram's web application is structured using React components. Each component encapsulates a specific part of the UI, such as posts, comments, and user profiles.
- *Dynamic Feed Updates*: React's virtual DOM and efficient diffing algorithm are crucial for handling the real-time updates in users' feeds.
- *Interactive Features*: React allows the integration of interactive features, such as real-time notifications, likes, and comments.
- *Single-Page Application (SPA)*: Navigation between different sections of the app occurs without full page reloads, providing a smoother user experience and everything occurs with-in a single page.

Intro to Logical Paradigm

Logical programming is a computer programming paradigm that has its foundations in mathematical logic in which program statements express facts and rules about problems within a system.
The basic constructs of logical programming, terms, and statements, are inherited from logic. There are three basic statements:

- **Facts** are fundamental assertions about the problem domain (e.g. "Socrates is a man")
- **Rules** are inferences about facts in the domain (e.g. "All men are mortal.")
- **Queries** are questions about that domain (e.g. "Is Socrates mortal?")

Features of Logical Programming

In logic programming we have a *knowledge base* which we know before and along with the *question* and *knowledge base* which is given to machine, it produces *result*.
Logic programming is naturally designed to answer queries. It can determine whether a query is true or false, or provide a list of choices that satisfies the query
Features of Logical Programming:

- It is very useful for representing knowledge. Logical relationships can easily be transferred into facts and rules for use in a logic program.
- Logic programming syntax is straightforward and users have to understand only the logical domain and how to add predicates.
- It can be used to represent very complicated ideas and rapidly refine an existing data model.
- It is efficient in terms of memory management and data storage and allows data to be presented in several different ways.

Use-cases : Artificial Intelligence, Database Management, Predictive Analysis, NLP

DataLog - An Intro

Datalog is a declarative logic programming language. While it is syntactically a subset of Prolog, Datalog generally uses a bottom-up rather than top-down evaluation model. This difference yields significantly different behavior and properties from Prolog.

- A Datalog program consists of facts, which are statements that are held to be true, and rules, which say how to deduce new facts from known facts.

For example, here are two facts that mean *xerces* is a **parent** of *brooke* and *brooke* is a **parent** of *damocles*:

- Parent(xerces,brooke)
- Parent(brooke,damocles)
- :- is read as IF and ,(comma) is read as AND
- ancestor(X, Y) :- parent(X, Y). -> X is ancestor of Y if X is parent of Y.

Logical Paradigm in DataLog

Datalog supports recursion, enabling the definition of rules in terms of other rules. This feature is crucial for expressing complex relationships and recursive queries.
Logical Programming in Datalog:

- **Knowledge Representation**: Facts and rules model domain knowledge.
- **Inference Engine**: Derives new facts from existing knowledge, answering queries.
- **Recursion**: Supports recursive definitions, enabling complex relationships.
- **Declarative Semantics**: Focus on what to compute, not how.

```datalog
parent(john, jim).
parent(john, ann).
parent(ann, bob).
parent(ann, mary).
parent(bob, sara).

% Rule 1: X is a parent of Y if there is a 'parent' relationship.
parent(X, Y) :- parent(X, Y).

% Rule 2: X is an ancestor of Y if X is a parent of Y.
ancestor(X, Y) :- parent(X, Y).

% Rule 3: X is an ancestor of Y if X is a parent of Z, and Z is an ancestor of Y (recursive rule).
ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y).

?- parent(john, jim).
% This query will return true, indicating that john is a parent of jim.

?- ancestor(john, sara).
% This query will return true, indicating that john is an ancestor of sara
through the parent-child relationships defined.
```

Case Study: Network Configurations

Datalog is commonly used in the domain of **network configuration** for expressing and reasoning about relationships within network policies.

How Datalog is Used:

- *Rule-Based Policies*: Datalog is used to define rule-based policies that govern network configurations.
- *Efficient Querying*: Datalog's querying capabilities are utilized for efficiently retrieving information about the current state of the network.
- *Network Optimization*: Datalog rules are employed to optimize network configurations. For example, rules can be defined to identify redundant or conflicting policies.
- *Dynamic Configurations*: Datalog's ability to express recursive rules is valuable for handling dynamic network configurations.
- *Security Analysis*: Datalog is used for security policy analysis within network configurations.

Comparisons and Discussions

Feature	Reactive Programming(RP)	Logical Programming(LP)
Core focus	Data flows and events	Knowledge representation and reasoning
Programming style	Declarative, data-centric	Declarative, logic-based
Execution model	Asynchronous	Inference-based
Error Handling	through data streams	through backtracking
Typical applications	User interfaces, real-time systems	Knowledge bases, NLP

Table: Key Differences

RP and LP offer distinct approaches to programming, each excelling in specific problem domains. Their declarative nature and focus on data and logic, respectively, make them powerful tools for building complex and intelligent systems.

References

1. https://www.linkedin.com/pulse/reactive-programming-principles-explained-luis-soares-m-sc-/
2. https://medium.com/@kevalpatel2106/what-is-reactive-programming-da37c1611382
3. https://react.dev/learn
4. https://brainhub.eu/library/famous-apps-using-reactjs
5. https://hackr.io/blog/programming-paradigms
6. https://www.linode.com/docs/guides/logic-programming-languages/
7. https://en.wikipedia.org/wiki/Datalog
8. Generative-AI tools: ChatGPT and Bard