Reo paradigm

CONCEPTS, SEMANTICS AND APPLICATIONS

DAVID COSTA
CWI, AMSTERDAM, NETHERLANDS
COSTA@CWI.NL

Roadmap

1. Motivation and Contextualization
2. Key Concepts
3. Reo
   1. Primitives: channels, nodes, I/O operations
   2. Operations: Composition, Encapsulation
4. Reo semantics
   1. Connector colouring *based*
   2. Automata *based*

Morning

5. Reo animations
6. Reo Eclipse Plug-in

Afternoon

Motivation and Contextualization

- Distributed, Concurrent, Parallel programming

It is **hard** to write *quality* software in general.
But it has become **even harder** to do so:

1. For most recent hardware,
2. For the cloud,
3. For most flexible architectures such as SOA
4. Etc…

- Loosely coupled components
  - Message passing
  - No method calls, no access to public fields, no statics of any kind.

Motivation and Contextualization

- How does a **language** that aims at making **this things easier** looks like?
- **Reo** is a long-term strategic project at SEN3 group in CWI
  - Addressing this question
  - And others:
    - Expressiveness
    - Scalability
    - Develop Tools that leverage the **small gap** between the conceptual model of Reo and the software architect/engineer devised solutions

What Reo like languages propose to be?

- A programming model where programmers:
  - First decompose in functional components their solution
  - Then arrange the coordination between components in a way that is close to their natural conception of the solution.
- If you can model your solution in terms of interactive components, encoding it in Reo should be straightforward
- Will likely avoid many common concurrency-related bugs
- Leverage the promised performance boost of recent hardware
- prevent the proliferation of anti-patterns.

Perspectives

- 2 different perspectives on a language such as Reo
  - As a programming language designer
    - How to design a language such as Reo?
    - What are the main language constructs and concepts?
    - What about its semantics?
  - As a software architect/engineer
    - What changes in the way software solutions are devised?
    - What are the benefits?
    - Does it really pays off?

Development in mainstream PL (Java)

- Isolation
  - google-guice
    - Ultra-lightweight, next generation dependency injection container for Java
      - Easy unit testing
      - Maximal flexibility and maintainability
      - Minimal repetition

Developments in mainstream (.NET)

- Parallelization
  - .NET 4 Task Parallel Library
- Other advanced concepts:
  - Dataflow networks
  - Asynchronous methods
  - Type annotations for taming side-effects
  - Decentralized Software Services

"Axum is a language that builds upon the architecture of the Web and principles of isolation, actors, and message-passing to increase application safety, responsiveness, and developer productivity"

http://msdn.microsoft.com/en-us/devlabs

Reo

- A paradigm for *composition* of *distributed* software components and services, based on the notion of *mobile channels*.
- Enforces an *exogenous* channel-based coordination that defines how designers can build complex *coordinators*, called *connectors*, out of simpler ones.
- Reo connectors orchestrate the cooperative behaviour of components or services in a component-based system or service oriented application.

Reo Key Concepts

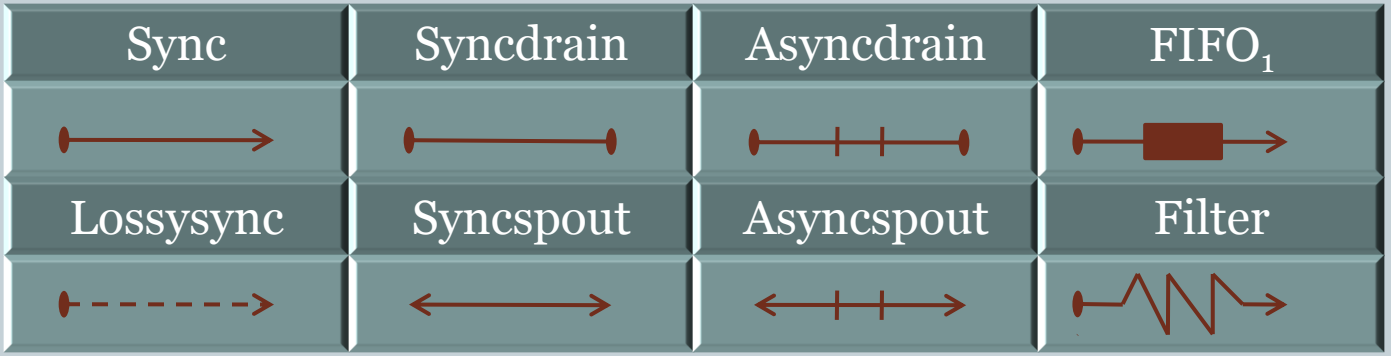
- Loose coupling among component and services
- Compositional construction of connectors
- User defined channels
- Arbitral mix of synchrony and asynchrony
- Context dependent behaviour by constraint propagation
- Mutual exclusion
- Effort to provide *accessible* Graphical syntax
- Dynamic reconfigurability of connectors
- Support for distribution and mobility of heterogeneous components

Special word about Channels

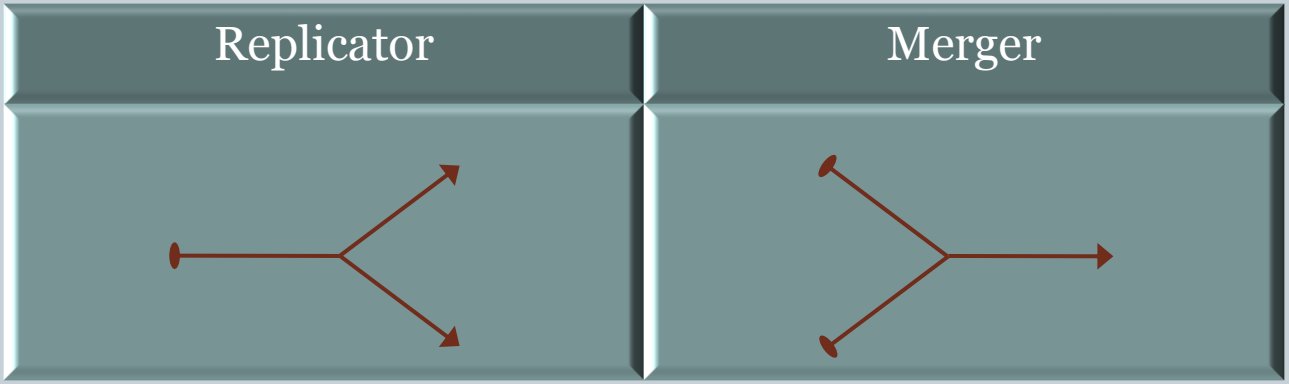
- The two components communicating over a channel are decoupled from each other:
  - Each component doesn't know or care how the other one is implemented.
  - The "contract" between them is specified by the channel only.
- To borrow an analogy from the OOP, the channel acts as an interface, and the component as the class implementing the interface.

Reo primitive connectors

- Channels

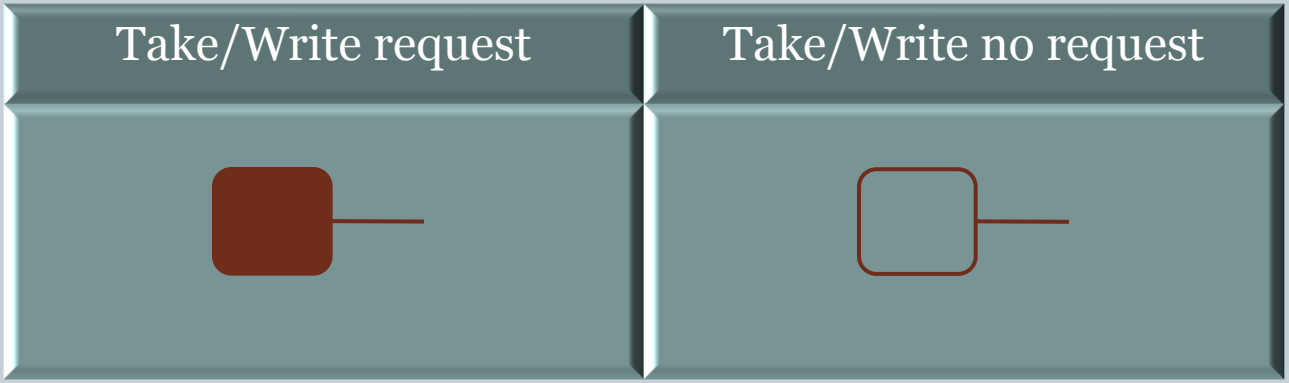


- Nodes



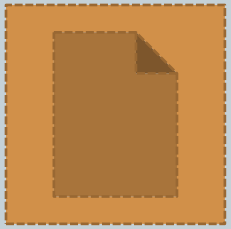
Reo primitive connectors

- I/O operations

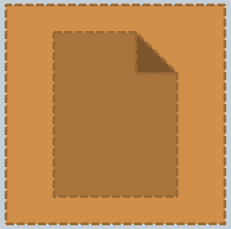


Reo Semantics

- Connector Colouring



- Intentional Automata



Instructions for the Tools sessions

- Requirements: Java 1.5 or higher
- Download eclipse
  - http://www.eclipse.org/downloads
    - Eclipse IDE for Java Developers (85 MB)
- Install eclipse by unzipping the downloaded file
- Run eclipse and install Reo plug-in
  - Go to Help > Software Updates…
    - Click on available software
    - Click on add site and write the following location:
      - http://reo.project.cwi.nl/update
    - Mark to install the following plug-in:
      - Reo Core Tools
    - Click Install
- Change the eclipse perspective to Reo