DMPP: main concepts

Bruno Rocha Pereira

June 14, 2016

Chapter 1

Introduction

- **Deutsch fallacies (you don't need to know the list by heart but to understand what each fallacy implies)**.

  In a perfect world:

  1. The network is reliable!
  2. The network is secure!
  3. The network is homogeneous!
     → All devices have the same configuration, ports and OS
  4. The topology does not change!
  5. Latency is zero!
  6. Bandwidth is infinite!
  7. Transport cost is zero!
  8. There is one administrator

- **Distribution vs concurrency, distribution vs. mobility**.
- **Distribution transparency and its fundamental issues (again, you don't need to know the list of problems by heart but to understand what each issue implies and why distribution transparency is said to be a myth).**
- **Language vs. middleware vs. reflective approach.**
- **Essential complexity vs. accidental complexity.**

Chapter 2

Prototype-based Programming

- Why are prototypes relevant for distributed programming?
- Symbiosis: understand the concept, how symbiosis works between AmbientTalk and Java.
- Object creation exnihilo vs. cloning, cloning and instantiation, delegation and cloning.
- Scoping: lexical vs object scope,methods vs. closures.
- Forwarding vs. delegation.
- The uniform access principle (UAP), UAP and closures, firstclass methods.
- Classifying objects with type tags.
- The concept of traits, objects as traits.

Chapter 3

Event-loop Programming

- Threads: understand the model, how to achieve inter and intraobject synchronization, understand why and how deadlocks happen.
- Actors: the basic principles of the model, and difference with the thread model.
- Active Objects: understand the model, and its limitations for distribution.
- Event Loops: understand the model, the concept of far references, and the three concurrency control properties that the model can enforce.
- Actors in AmbientTalk, asynchronous message sending and parameter passing rules.
- Asynchronous message sending and return values:
- The concept of customer objects, and its limitations.
- Futures: the concept, blocking vs. nonblocking futures, future pipelining, how to achieve conditional synchronization with futures.

Chapter 4

Event-based Distributed Programming

- Understand the different design issues to provide a distributed object model, and its relation to the different types of networks.
- Know the discriminating properties of mobile networks, and understand the requirements for a distributed object model for MANETs.
- Far references and partial failures: how far references deal with intermittent failures.
- Ways of obtaining a first far reference, service discovery vs. service lookup.
- Distributed object scoping and isolates.
- Far references and permanent failures, leased object references.
- Understand the difference between pessimistic and optimistic distributed object model.

Chapter 5

Meta-level Engineering

- Metaprogramming: firstclass messages, quasiquoting and splicing, first-class abstract grammar
- Reflective programming: base vs meta level, meta vs. reflective program.
- Terminology on reflection (reification, introspection, intercession), structural vs. behavioural reflection.
- The concept of meta objects, and understand the problems with popular meta-level architectures
- Mirror-based reflection: understand the model and the three properties that the model can enforce.
- Mirrors in AmbientTalk: explicit vs. implicit mirrors on objects.
- Mirages: the concept.
- Mirrors on Actors: understand which operations actor mirrors reify in comparison to object mirrors.
- MOP: You are not expected to know by hearth the message invocation protocol but the overall idea and understand the relevant parts that need to be altered in order to implement language constructs like futures and leased references.

Chapter 6

Coordination using Tuple Spaces

- Terminology: coordination, datadriven vs. control driven coordination.
- Tuple Spaces: the basic interaction mechanism, understand the Linda model and how tuple matching works.
- Know what decoupling in time and space, and synchronization decoupling is, and being able to reason about the forms of decoupling that tuple spaces and other communication paradigms seen at the course exhibit.
- Federation of mobile tuple spaces: understand the basic interaction model of tuple spaces in a mobile setting, and being able to explain the two variations (LIME & TOTA), know what are the differences and similarities of the two variations.
- TOTAM: understand the goal for this hybrid approach, and key features, and how the model supports memory management in face of failures.

Chapter 7

Peer-to-peer systems

- Typical characteristics from P2P systems.
- The concept of an overlay network.
- First generation of P2P systems: motivation and limitations.
- Second generation of P2P systems: understand the basics, flooding, TTL propagation
- Understand the motivation for P2P third generation and which guarantees they provide w.r.t. previous generations, distributed hash tables
- Chord: understand how keys are distributed, the lookup algorithm and the different cost models of lookup, how join and leave of nodes affect the network, and the role of periodic stabilization
- CAP theorem: understand consistency, availability, partition tolerance means and their tradeoff. Be able to reason which guarantee Chord and other distributed algorithms or applications cannot provide.
- Distributed storage: know why P2P systems scale well specially for immutable objects, being able to reason about different replication strategies.
- Beernet: the goal of the approach and basic architecture. Understand the differences with Chord with respect to management of peer failures, and why it is relevant in a mobile setting.

Fundamentals

- Understand why external synchronization with physical clocks is not employed for coordinating distributed processes.
- Lamport's clocks: the algorithm, know which properties Lamport's clocks exhibit, understand what it means that Lamport clocks are not strongly consistent.
- Partial vs. total ordering of events.
- Vector clocks: Understand the difference with Lamport's clocks, which properties they exhibit, and how you can compare vector clocks. Understand the limitations and assumptions of vector clocks.
- Be able to reason about application domains where lamport and vector clocks can be applied.
- Consensus: two Generals' problem, know what the consensus problem is.
- Know what it means the safety and liveness properties, and being able to reason about those properties for distributed algorithms.
- Know the FLP Theorem and understand its implications.
- Paxos:
  - understand the goal, the roles of nodes and the purpose of each phase.
  - Be able to explain why the algorithm cannot reach agreement sometimes, and how the safety properties are guaranteed.
  - Be able to reason about application domains where paxos is useful.

Chapter 8

Distributed Programming on the Web

- Know the basic interaction model with a client-server architecture communicating by means of HTTP request.
- Understand why programming on the web 1.0 is said to be programming with strings.
- Know what a RIA is, and the role of JavaScript in the development of RIAs.
- AJAX: understand the idea that scripts can asynchronously communicate with the server, and why the interactions run asynchronously.
- Thread-based vs. event-based servers.
- Know the two ways how to achieve communication amongst clients (Publish/ Subscribe on top of HTTP and websockets) and reason about their advantages and disadvantages.
- Mobile cross-platform solutions:
  - The idea of employing web-based technologies (i.e. JavaScript) for mobile computing, and which advantages brings to mobile software development.
  - Understand the basics of two big family of cross-platform solutions (hybrid vs. interpreted) and be able to reason about their advantages and disadvantages.

Chapter 9

Reactive Programming

- Understand the differences between classic sequential and event-driven software.
- Know the advantages of event-driven client and servers for RIAs.
- Know what the issue of "Inversion of Control" (a.ka. the callback hell effect) is, and its implications (w.r.t shared state)
- Reactive programming: the basic model of evaluation, and key abstractions (event sources vs. behaviours), lifting, glitches and the most common dataflow evaluation strategy to reevaluate code without causing glitches.
- Flapjax:
  - understand how pages can be made reactive
  - you do not need to know all the operations on event stream combinators, but you should be able to reason about a piece of code with respect to which parts are reactive, the dependency graph, and how reactive code communicates with the DOM and server).
  - Understand what it means that there are no callbacks in Flapjax in contrast to JavaScript.