Distributed Multiplayer Checkers Platform

Distributed Systems & Middleware Technologies

Reza Almassi
Supervisor: Prof. Alessio Bechini

Academic Year 2024/2025

University of Pisa

Project Overview

- Real-time distributed multiplayer Checkers game
- Java game servers, Erlang/OTP coordinator, JavaFX GUI
- Fault-tolerant, live registry, and session management
- Demonstrates: synchronization and coordination
- Code: `https://github.com/Rezacs/DistributedSystems`

Agenda

Motivation and Goals

System Architecture

Technology Stack

Key Features

Demonstration

Results & Live Demo

Deployment

Discussion and Future Work

References

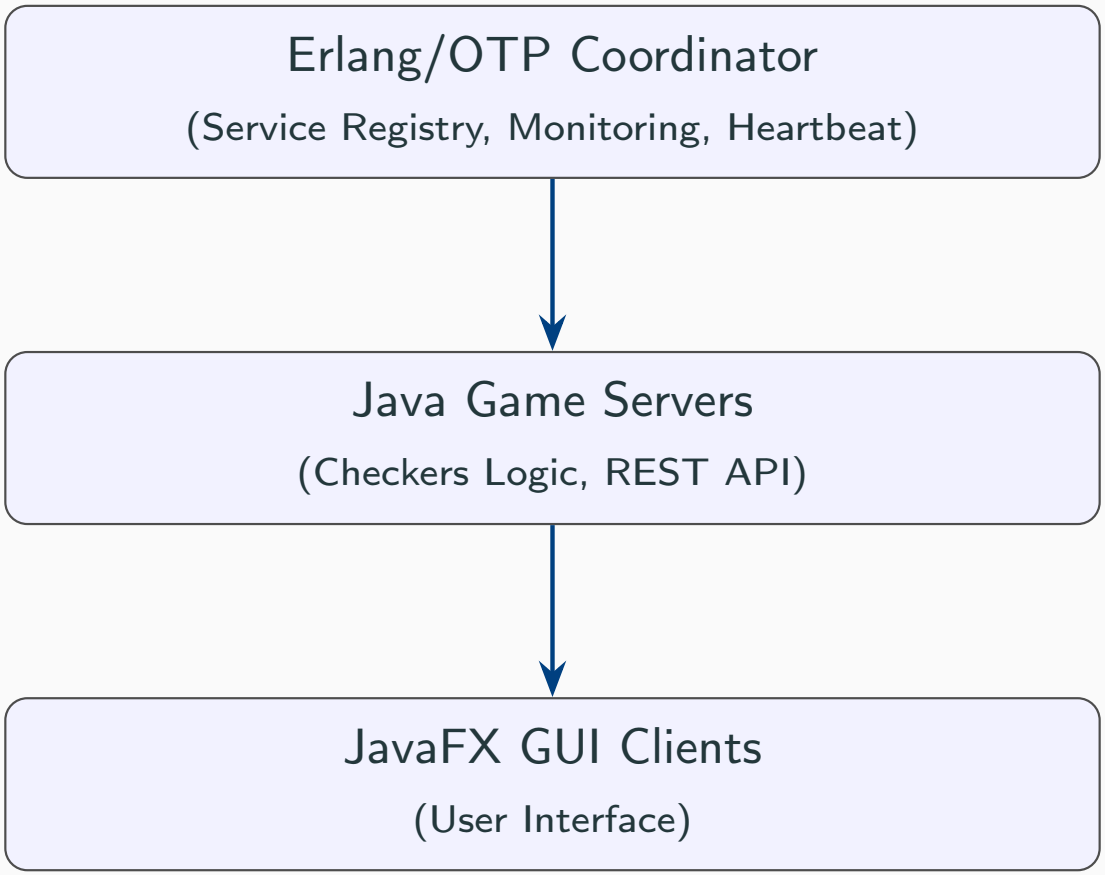
Motivation and Goals

Motivation and Goals

- Explore and demonstrate core principles of distributed systems:
  - Synchronization, coordination, distributed fault tolerance
  - Live service registry, session management, recovery
- Provide a hands-on platform for multiplayer gaming across servers
- Leverage modern technologies: Erlang/OTP (concurrency, monitoring), Java (game logic), JavaFX (GUI)
- Prepare a reproducible, scalable, and robust academic demo

System Architecture

High-Level Architecture (Vertical)



Component Overview

- Erlang Coordinator:
  - Cowboy-based HTTP server
  - Service registry (server registration/deregistration)
  - Heartbeat monitoring and failure detection
- Java Game Servers:
  - Game session logic, REST API endpoints
  - Register/heartbeat with coordinator
  - Moves, game state, player/session management
- JavaFX GUI Clients:
  - Discover servers, create/join games, play in real time
  - View board, turns, player names, winner

Technology Stack

Technology Stack

- Languages & Frameworks:
  - Erlang/OTP, Cowboy (HTTP server)
  - Java (Spark framework), Unirest (HTTP client), Gson (JSON)
  - JavaFX (GUI)
- Protocols:
  - RESTful APIs for all communication (server registration, move, game state, etc.)
- Deployment:
  - Distributed across two university-assigned containers
  - VPN-secured access for demonstration

Key Features

Key Features

- Distributed Game Servers: Multiple Java servers, each with independent sessions
- Service Registry: All servers register and heartbeat with central Erlang coordinator
- Fault Tolerance: Dead servers removed from registry automatically
- Dynamic Recovery: New/restarted servers rejoin at any time, no downtime
- Session Management: Games tracked by session ID, player rejoin with same name and ID
- Live GUI: Players discover, join, and play games across distributed servers

Demonstration

Demonstration Scenario

1. Start Erlang coordinator on Container 1 (`10.2.1.54`)
2. Launch Java game servers on both containers (`10.2.1.54:8081`, `10.2.1.55:8082`)
3. Start JavaFX GUI client (on local PC)
4. Discover available servers, create and join games
5. Demonstrate real-time play and automatic failure detection (kill a server, see it removed)
6. Show session recovery and continued play on remaining servers

Results & Live Demo

Live Registry and Server Monitoring

- Both Java servers registered and monitored by Erlang coordinator
- Registry updates live as servers start/stop

Distributed Game Play

- Players join games on different servers
- Moves synchronized, turns enforced, winner detected
- GUI updates in real time

Server Failure and Recovery

- When a server crashes, it disappears from registry automatically
- Clients are notified and can reconnect to remaining servers
- Demonstrates robust fault tolerance and dynamic recovery

Deployment

Deployment on University Containers

- Container 1: `10.2.1.54` (`root/root`)
- Container 2: `10.2.1.55` (`root/root`)
- Accessed via UNIPI VPN and SSH
- Coordinator runs on Container 1; servers on both containers
- Client GUI connects over VPN to play from anywhere
- All commands, scripts, and build steps documented in the project repo and documentation

Discussion and Future Work

Discussion and Future Work

- Current Limitations:
  - Game sessions are in-memory only (no persistent backup)
  - Player authentication is name-based (not secure)
  - No web or mobile client yet (only JavaFX)
- Possible Improvements:
  - Persistent session storage and automated failover
  - Secure authentication and HTTPS support
  - Web-based and mobile GUIs for wider accessibility
  - Enhanced monitoring, analytics, and scaling (cloud deployment)

References

References

- Erlang/OTP documentation: `https://www.erlang.org/doc/`
- Java: `https://docs.oracle.com/en/java/`
- SparkJava: `http://sparkjava.com/documentation`
- Unirest: `https://kong.github.io/unirest-java/`
- JavaFX: `https://openjfx.io/`
- Project Code: `https://github.com/Rezacs/DistributedSystems`
- Tanenbaum & van Steen, *Distributed Systems: Principles and Paradigms*
- Course Material: Prof. Alessio Bechini, University of Pisa