Codeine - Computing over Decentralized Network, with P2P

AGH University of Science and Technology
Faculty of Electrical Engineering, Automatics, Computer Science and Engineering in Biomedicine

Kacper Tonia   Przemysław Nocoń   Jakub Komnata   Sławomir Kalandyk

1 Glossary

- Agent - single application instance, is able to compute one subproblem at a time
- Computational problem - problem solvable with Codeine. It should be divisible into a finite amount of subproblems which can be solved independent of each other
- Computational network (usually referred to as "network") - a network of agents communicating with each other, who together solve one computational problem
- Subproblem - a single subproblem of the computational problem

2 Requirements

- Project should implement peer-to-peer networking on LAN
- We should assume that about 5 agents at once can work on our computational problem
- Every agent should have exactly the same application
- After application launch, agent should automatically attempt to discover other agents in the network
- The computational problem itself doesn't matter, it should only allow for long enough computing time to let us see the network working as intended
- Subproblem assignment should be decentralized
- Agents should be immune to other agents disconnecting from the network, there should be no side effects
- Results should be visualized, accessible (in the best case in real time)

3 Assumptions & Constraints

- Packet type - a string consisting of only upper case letters (e.g. IMALIVE, NETTOPOLOGY)
- The solution of a single subproblem should be able to fit in a single UDP packet (`<64kB`)
- Every subproblem has it's ID and immutable State, common for all subproblems
- Every subproblem can be solved

4 Networking

4.1 Packets

- Topology discovery, registering agents
  - IMALIVE - send empty packet informing that agent is in the network `<>`
  - NETTOPOLOGY - send network topology `<agent[]>`
- Subproblem assignment
  - REGSUBPROB - register new subproblem `<subproblem_id>`
  - DROPSUBPROB - stop working on this subproblem `<subproblem_id>`
- Result distribution
  - SUBPROBRES - send subproblem result `<subproblem_id, subproblem_result>`
- Confirmation
  - ACKNOWL - acknowledge `<previous_package_control_sum>`

4.2 Rules

# - broadcast

- #IMALIVE → NETTOPOLOGY (optional)
- #REGSUBPROB → DROPSUBPROB | SUBPROBRES
- #SUBPROBRES → ACKNOWL

4.3 Network scenarios

4.3.1 Scenario 1

Story:
Agent tries to join the network right after launching Codeine.

Prerequisites:

- None

Scenario:

1. Agent broadcasts IMALIVE packet.
2. Agent waits for ??? seconds for replies from every other agent already in the network.
3. Every other agent already in the network replies with NETTOPOLOGY packet.
4. Agent registers the network. End of Scenario 1.

Scenario extensions:

3a. There is no response from the network (proceed to calculations). End of Scenario 1.

4.3.2 Scenario 2

Story:
Agent periodically informs the network that he's still alive.

Prerequisites:

- Agent is already in the network

Scenario:

1. Agent broadcasts IMALIVE packet. End of Scenario 2.

4.3.3 Scenario 3

Story:
Agent wants to register a subproblem

Prerequisites:

- Agent is already in the network

Scenario:

1. Agent broadcasts REGSUBPROB packet.
2. Agent starts calculations.
3. No response from network. End of Scenario 3

Scenario extensions:

3a. An agent replies with DROPSUBPROB.

  3a.1. Agent stops calculations.
  3a.2. Agent sets that subproblem's state to WIP.
  3a.3. Agent chooses another subproblem. Repeat from point 1. End of Scenario 3.

3b. An agent replies with SUBPROBRES.

  3b.1. Agent stops calculations.
  3b.2. Agent registers received subproblem result.
  3b.3. Agent chooses another subproblem. Repeat from point 1. End of Scenario 3.

4.3.4 Scenario 4

Story:
Agent wants to broadcast subproblem results.

Prerequisites:

- Agent is already in the network
- Agent has calculated a subproblem and received a concrete result

Scenario:

1. Agent broadcast SUBPROBRES packet.
2. Agent waits for ??? seconds for replies.
3. Every other agent registered in his net topology base replies with ACKNOWL. End of Scenario 4.

Scenario extensions:

3a. If any agent hasn't replied with ACKNOWL, send packet SUBPROBRES again to him. Repeat ??? times.

  3a.1. An agent hasn't replied after ??? sent SUBPROBRES packets. Remove him from net topology base.

4.3.5 Scenario 5

Story:
Agent has received a SUBPROBRES packet with subproblem ID of a subproblem he already has a result of.

Prerequisites:

- Agent is already in the network
- Agent already has results of at least one subproblem

Scenario:

1. Agent received a SUBPROBRES packet.
2. Agent tries to register the subproblem result. Subproblem with that ID already has a registered result.
3. Subproblem result of received packet is ignored.
4. Reply with ACKNOWL packet. End of Scenario 5.

4.3.6 Scenario 6

Story:
Agent A tries to register a subproblem with ID == X. Agent B replies with STOPWIP packet. Agent A sets subproblem X's state to WIP. Agent B then disconnects.

Prerequisites:

- Agent is already in the network

Scenario:

1. Agent A broadcasts REGSUBPROB packet with subproblem ID == X.
2. Agent B replies with STOPWIP.
3. Agent A sets subproblem X's state to WIP.
4. Agent B disconnects.
5. Agent A doesn't receive IMALIVE packets from agent B for ??? minutes.
6. Agent A sets subproblem X's state to unregistered. End of Scenario 6.

5 Computational problem

The chosen problem is finding a hash created with SHA1 cipher corresponding to a hardcoded hash of a 6 letter password. The goal is to find hashes for all possible 6 character combinations of lower case letters and digits and compare them to the hardcoded password until the correct one is found. In the worst case scenario, 36^6 hashes have to be calculated.

To solve it with a decentralized computing network, it has been split into 36^2 subproblems, each consisting of 36^4 hashes to calculate. They are divided based on the first two characters, e.g. one subproblem is to find hashes of all 6 character long strings that start with "bg". Each agent can compute only one subproblem at a time.

6 Project Structure

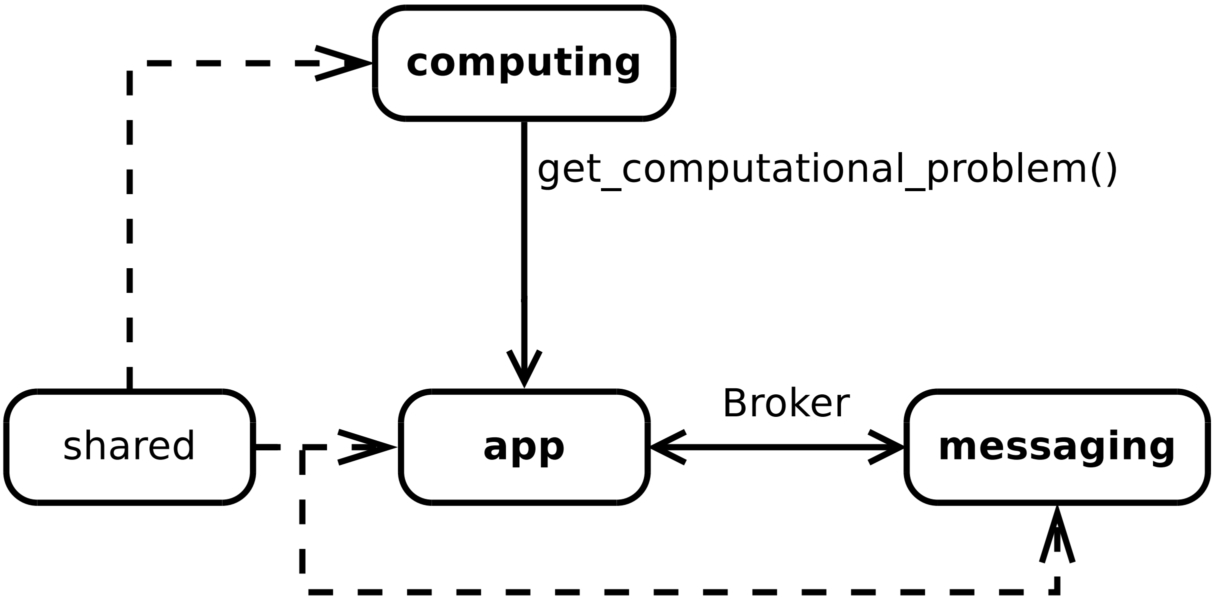

Figure 1: Project Structure Diagram

Packages:

- **app:** top-level package. It is the main thread of an application, responsible for managing other threads. It manages subproblem execution and communicates with the Broker. The app package does not deal with network details directly: high-level commands are used instead of network packets.
- **messaging:** contains a Broker definition, an abstraction layer between the app and the computer network. The Broker runs on a separate thread and listens to incoming packets, as well as sends commands received from the app package.
- **computing:** contains implementation details of computational problem.
- **shared:** contains utilities which are either used by all other packages, or are not unique to our project. For instance, NetworkConnection (a socket wrapper class) could be easily used in other projects that utilise communication over network.
- **tests:** contains unit and integration tests that cover the rest of packages.

7 Diagrams

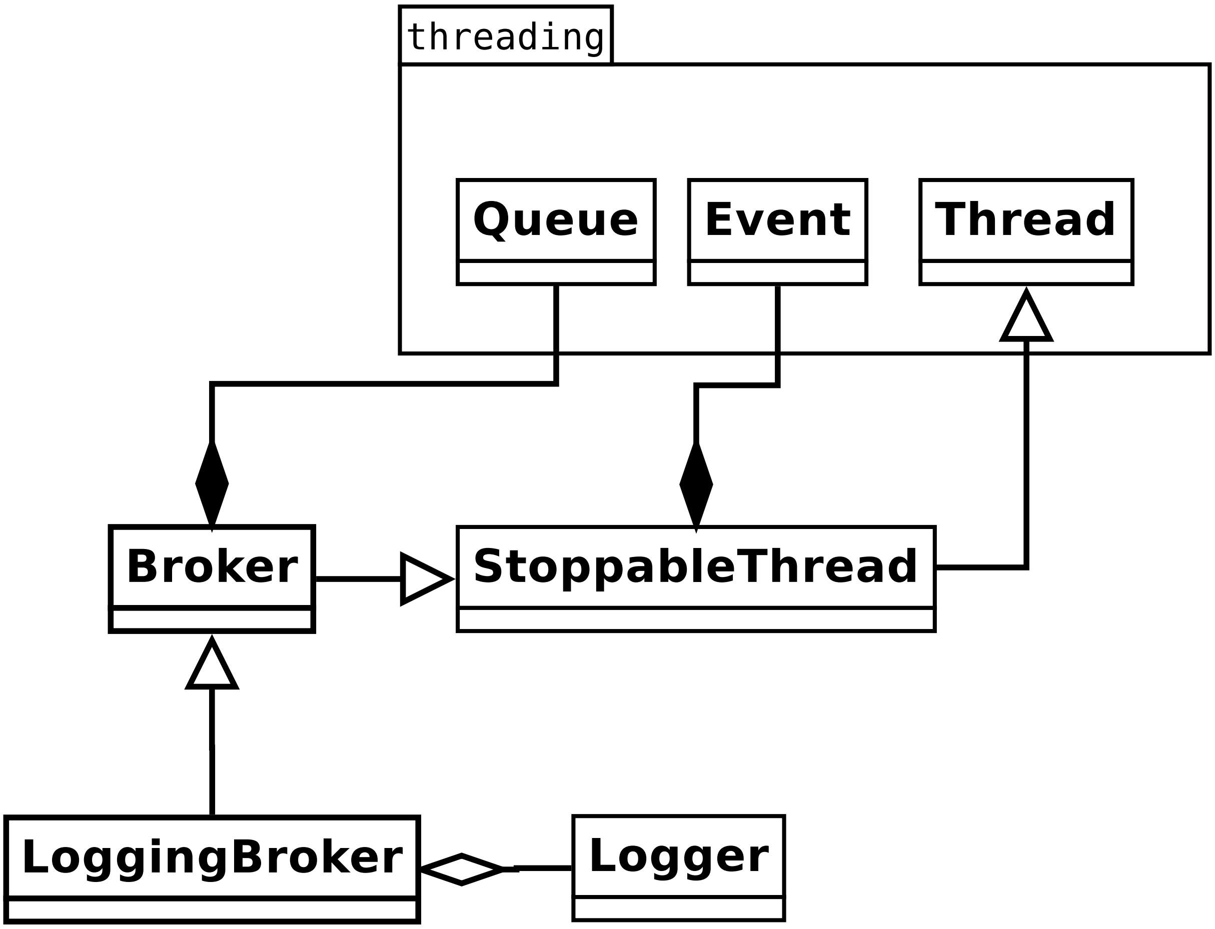

Figure 2: Broker Class Diagram

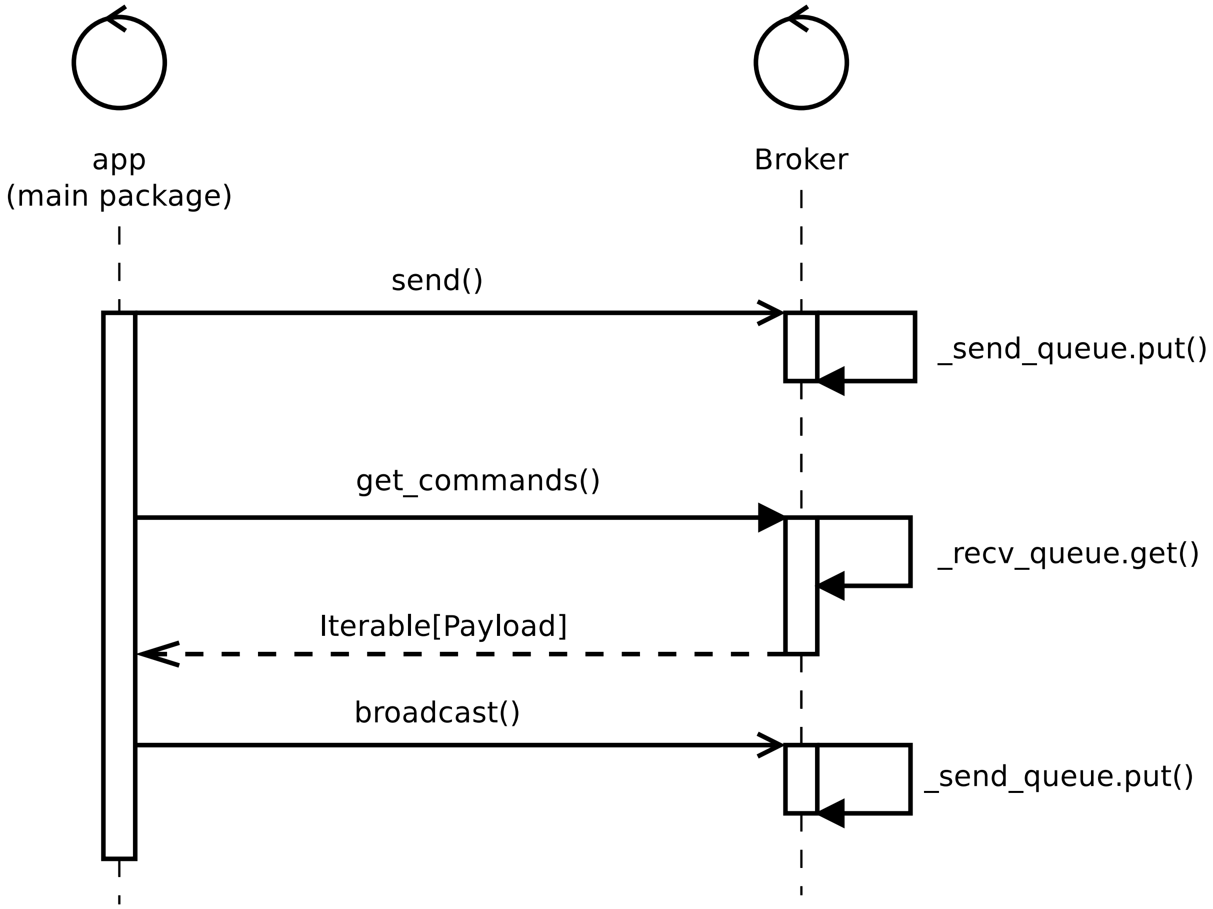

Figure 3: App-Broker Sequence Diagram

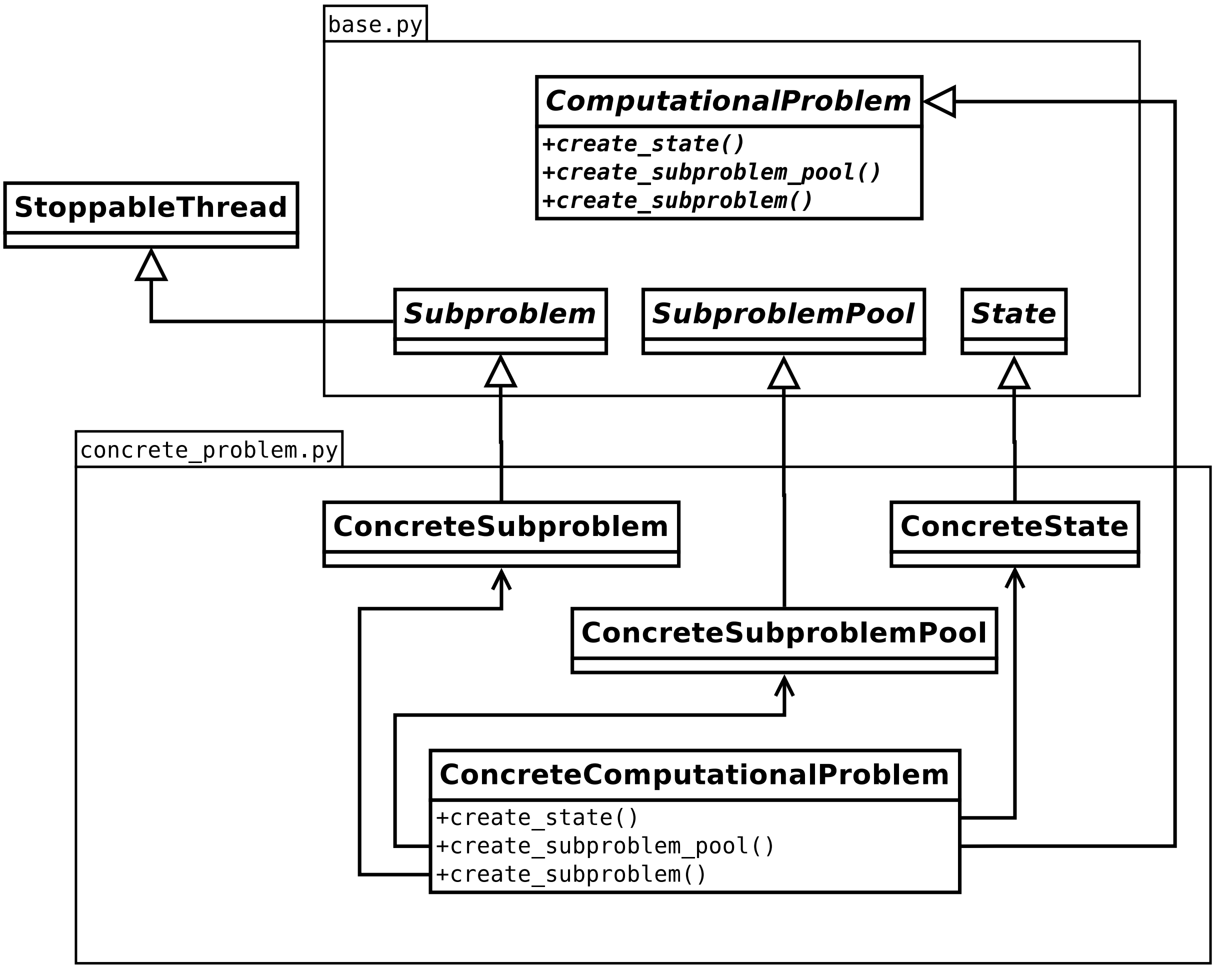

Figure 4: Computational Problem Abstract Factory Diagram

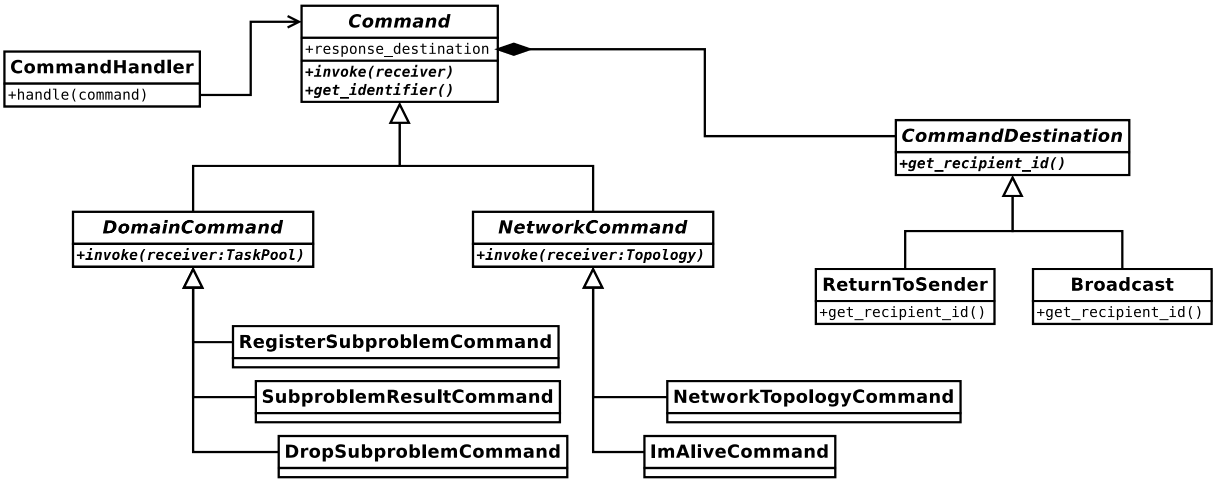

Figure 5: Command Diagram

8 Security

The possibilities to disrupt a computational network are plentiful:

- "Flood" the network with fake result packets
- Deny all "register subproblem" requests
- Solve subproblems using a "malformed" state (e.g. replacing an original texture while rendering an image)
  - Exchanging a state hash/control sum would help detect accidental mistakes. Malicious changes could be caught by solving a certain subproblem several times (which still wouldn't help if an attacker had control over nearly 50
- Disruption of data stored locally, in a file
  - Little can we do to protect data against someone with a physical access to a machine
- One computational network per actual computer network
  - Introducing a kind of a session key might resolve the issue
- Communication between agents is unencrypted
  - Encryption itself will be easy to implement, if needed. It may be based on the aforementioned session key.

9 Technology

- **Python 3.7:** Python version 3.7. introduced @dataclass, which makes it easier to work with data structures. It allows creating immutable data structures.
- **Network Communication:** transmission of data in JSON format, using UDP protocol.
- **Tkinter:** Python library Tkinter will be used to create GUI.