Implementing migration in SOFA

- Václav Remeš
  - Master thesis – Migration and load balancing in distributed hierarchical component systems
Outline

- Introduction
  - Motivation - component migration
- Handover protocol
- How it works
  - Implementation in SOFA 2
- Writing migrable components
- Load balancing
- Conclusion and future work
Motivation

- Possibilities coming with component migration
  - Maintenance
  - Load balancing

- Example:
  - In a SOFANode with all components migrable
    - Deployment Docks can be added/removed dynamically
    - It doesn't matter where components are instantiated
    - With the help of load balancing, the SOFANode can be seen like a cloud
Migration in general

- Several steps
  1. Stopping
  2. Transfer
  3. Reconnecting
  4. Error handling + finalization

- Composed X Primitive components
- Anticipated/unanticipated
Migration in general

- Unanticipated
  - The framework should take care of stopping the component and its transfer alone
  - Hard to implement without low-level access
    - Stopping threads
    - Transfering different resources (files, sockets, ...)

- Anticipated – the developer implements
  - Component stopping/resuming
  - State saving/loading
Migration example

Dock A

Dock B

Dock C

Dock D

Component A

Component B

Component C
Migration example
How it works

Dock A

Dock B

Dock C

Dock D
Stopping component's jobs

- Stop incoming calls
  - In skeletons
  - Just delayed – they are passed through when the migration is over
Preventing deadlocks while stopping incoming calls

- When a call that was executed just before stopping the component calls back the migrated component

  =>$ \text{deadlock}$

- Solution – ThreadObserver
  
  - Every incoming and outgoing thread is noted down
  
  - ThreadContext used
  
  - Incoming calls which originated from a thread observed as outgoing (or from one of its successors) can't be delayed and must be passed through
  
  - Threads are removed from the observer once they lose their meaning for the migration purposes
Stopping component's jobs

- Stop the component
  - Anticipated – void pause()
    - The component can have inner threads
  - Wait for ongoing jobs to finish
Stopping component's jobs

- Stop the outgoing calls
  - After the pause() method terminates
  - Just to be sure
Transfer the component's state

- Transfer the inner state of the component
  - and information about required interfaces
- Implicit/explicit serialization
Reconnecting required interfaces

- Reconnect the required interfaces using RemoteRefBundles transferred from the previous instance
- The new instance can "resume()" now
Reconnecting required interfaces

- Local/Remote bindings decided by first trying local references; if they fail, remote references are used
  - RemoteRefBundle local stringified references "know" which Deployment Dock they belong to
Reconnecting provided interfaces

- Send back information about provided interfaces
Quit delaying the incoming calls and pass the delayed ones through

Other components continue with their requests
- Throw MigratedException
  - RemoteRefBundles included inside
Other components reconnect their required interfaces pointing to the migrated component.
Possible problem: Dock B is shut down before Component A reconnects
- Solution: checkAlive method in connectors invoked at the migration finalization stage
Finalization

- New instance is fully functioning now
- If everything went ok, drop the old instance
Finished
Handling errors during migration

- Leave the old instance unchanged as long as possible
  - Best until it is clear that the migration went ok
- Provided interfaces reconnected at the end
- If anything goes wrong, fall back to the old instance
Implementation in Depl. Dock

- Migrable Deployment Dock
  - "emigration"
    - Executed at the "old" Deployment Dock
    - Stops the old instance
    - Sends information about the old component to the "new" Deployment Dock
  - "imigration"
    - Executed at the "new" Deployment Dock
    - Instantiates the new instance
      - Unconnected, not running
      - Loads its state
      - Resumes the component's jobs when everything is finished
    - Binds the required connectors
    - Returns back information about "imigration"
Writing migrable components

- Implement SOFAObjectMigratable interface
  - void pause()
  - void resume()
  - void saveState (Map<String, Serializable> storage)
  - void loadState (Map<String, Serializable> storage)

- Note:
  - In passive components (without inner threads) pause() and resume() can contain no code
  - Components not using external resources could be serialized implicitly
Example migrable application

- Migrable LogDemo
- Example: MigrableTester

- Show in MConsole
Load balancing

- Prepared framework for simple extending
  - Periodically asks a given balancing implementation for orders to migrate components
- Currently only a simple balancing implementation
Conclusion and future work

- Support for component migration added to SOFA 2
- Implement migration for communication styles other than just RPC
- Migration of composed components
  - Only the instantiated connectors
  - Will be included in the master thesis
- Load balancing
  - A more sophisticated implementation will be provided with the master thesis
The end

- Questions?