Lively4 Services

Fabio Niephaus, Philipp Otto
13.07.2016

Outline

- Project Idea
- Demo
- Implementation and Architecture
- Evaluation and Conclusion

Lively4 Technologies

Development Approach

- Live
- Web-based
  - On any Website
  - S3
- Collaborative
  - sync
  - async
- Testing
  - CI
  - live

Language Support

- ES6 Modules
- Source Code-based Workflow
  - .html, .md, .js
- Multi-language System
  - Runtime Adaptation
  - Debugging
  - Projectional Editing
  - Web Assembly Modules

Language Extensions

- Active Expressions
- COP
- OCP

Browser Support

- Semantic Web
- Windows App

Runtime Technologies

- Unified Data Access
- Cloud Variables
- WebStrates

Cloud Programming

- Project Deployment
- Cloud Dataflow
- Service Deployment

Performance

- Offline First
- Cache & Sync
- Cached Transformation

UI Technology

- HTML as Morphic
- Web Components
- Monkey Patching
  - by example
  - for testing
- Data Bindings

Project Idea

“Sometimes I want to write something in Lively that continues to run after I closed the browser, but I still want to be able to control it anytime.”

- “Heroku for Lively”
- Development and deployment of stand-alone services in Lively4
- Interactive debugging capabilities
- Seamlessly integrated into Lively4 UI and workflow

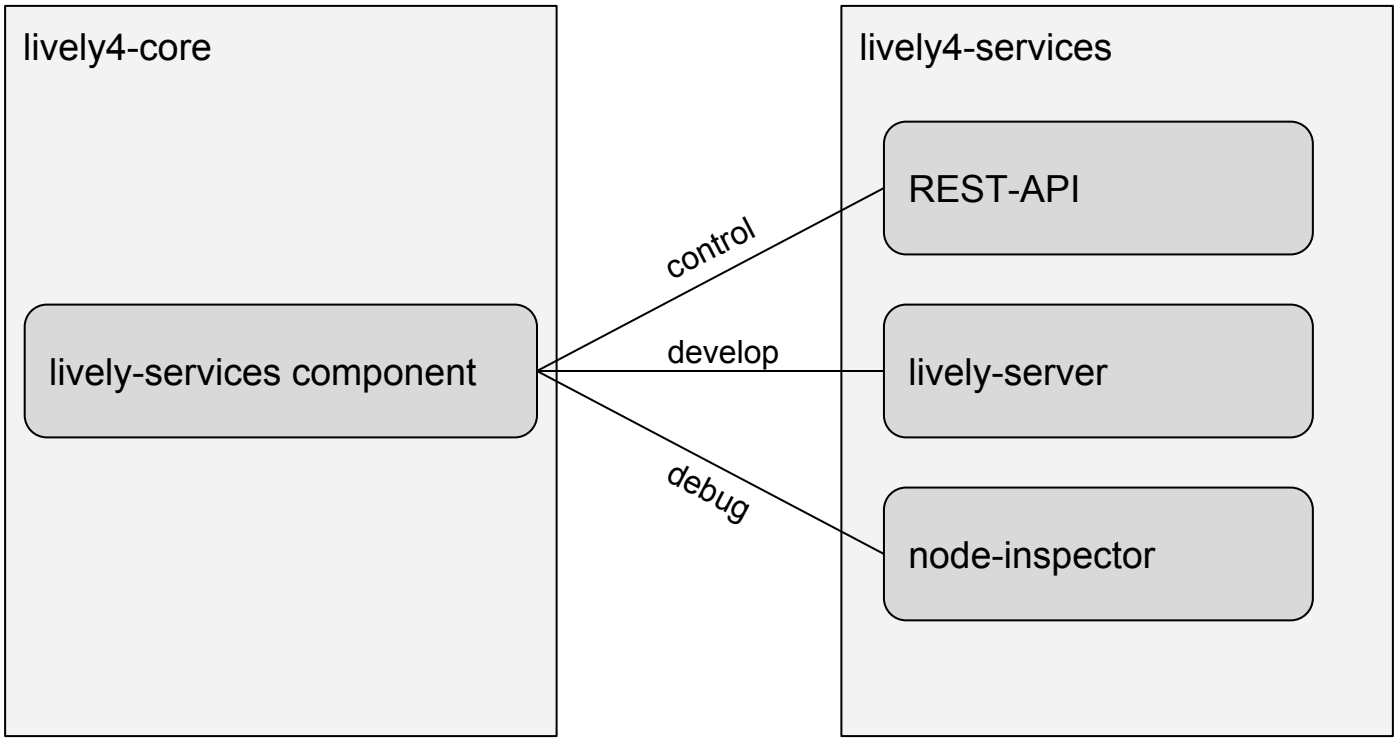
- Could we use this to develop and deploy lively4-server?

Demo

Implementation

- Node.js application that can be deployed anywhere (e.g. on Heroku, AWS, GCE)
- User interface as part of lively4-core
- Tests for lively4-services and UI component

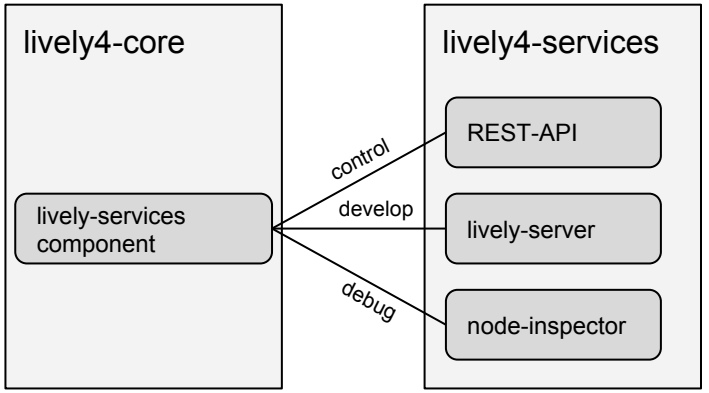
Architecture (simplified)



Components

lively-services component

- Lively4 UI



lively4-services instance

- Exposes REST-API used by UI (e.g. control, log files, …)
- Starts a lively-server instance to allow remote development of services in Lively environment
- Attaches node-inspector instances to services for debugging purposes
- Builtin port forwarding capabilities

Related Work

- Heroku, AWS, GCE etc. allow relatively simple deployment, but no debugging
- Debugging of nodeJS projects is quite common with node-inspector, but mostly done locally
- Remote debugging is possible by executing node-inspector via ssh and opening the appropriate port
- IntelliJ IDEA supports remote deployment and debugging, but can’t hook into running applications
- Chrome supports remote debugging of web sites (not server code) on mobile devices
- Remote debugging for Windows driver development

Evaluation and Conclusion

- It is now possible to develop and deploy services within Lively4
- xterm.js demo shows how easy it is to deploy existing projects
- Interactive debugging allows to develop services the “Lively way”
- Infrastructure is easy to deploy and comes with builtin port forwarding capabilities

Future Work

- Client-side Blink debugger
- Collaborative debugging
- Load balancing of services
- Seamless switch between local and remote execution
- Automated deployments