Web-based modeling tools with EMF.cloud

Philip Langer & Maximilian Koegel
EclipseSource
Why web-based modeling tools?

● Cloud-based environment
  ○ No client installation
  ○ Access through a web link
  ○ Simple client updates
  ○ Physical resource sharing

● Usability
  ○ Modern UI look and feel
  ○ SWT vs. HTML5
  ○ GEF 3 vs. SVG

● Maintainability
  ○ Evolution
  ○ Hiring
Key enablers for building web-based modeling tools

- Eclipse Theia: Extensible cloud IDE
- Eclipse EMF.cloud: Web-based modeling tools
- Eclipse GLSP: Web-based diagramming
- Eclipse Che and Kubernetes: Cloud deployment
A prototypical modeling tool
What is EMF.cloud?

- Goal: Building web-based modeling tools
- Components for the cloud
  - covering frequent use cases
  - encapsulating best practices
- Example modeling tools
  - providing architecture blueprints
  - demonstrating EMF.cloud component usage
  - demonstrating usage of other components
- Incubator for new web-based components
How is EMF.cloud related to EMF?

- Independent project but many similar concepts
- EMF-independent components
- Model Server
  - defines a common interface
  - encapsulates EMF dependency
  - facilitates reusing EMF-based models
Example product: Coffee Editor

- Purpose: Demo tool to model coffee makers
- Features:
  - Model server to support multiple editors
  - Tree-Editor for structural modeling with forms
  - Graphical editor for behavioral model
  - Code generation
  - Working with source code
  - Textual Modeling
  - Model Analysis
Demo: Model Server
Why do we need a Model Server?
Model Server Features

- Runtime state of loaded models ("shared editing domain")
  - Share potentially "dirty" runtime state of a model
  - Command-based change interface
  - Notification mechanism via sockets

- Java-based server
  - Reuse existing model implementations
  - Based on popular REST framework

- Convenient model access
  - REST API: create, update, patch, save, ... models
  - Multi-format: JSON or XMI
  - Multi-platform: Java and JavaScript-based client APIs
Model Server API: Load

- Initialize model server for workspace folder
- Load or get model

```java
ModelServerClient client = new ModelServerClient("http://localhost:8081/api/v1/");
client.get("foo.xmi")
    .thenAccept(response -> ... do something with response.body() ...);
```

- `foo.xmi` is only loaded once per session!
- Other methods: `getSchema()`, `delete()`, `update()`, `save()`, ...

---

Model Server API: Subscribe

Subscribe, react to changes, optionally synchronize

```java
client.subscribe("foo.xmi\&format=json", new JsonToEObjectSubscriptionListener() {
  private final CommandCodec codec = new DefaultCommandCodec();

  @Override
  public void onIncrementalUpdate(EObject message) {
    Command command = codec.decode(editingDomain, (CCommand) message);
    editingDomain.getCommandStack().execute(command);
  }
});
```

Subscription via sockets and serialized commands!
Model Server API: Change

Change models via commands:

- Editor sends command
- Model server applies command
- Model server notifies subscribed editors
- Subscribed editors react, e.g. update
Demo: Form-based editing
Involved components: Form-based editing

- Tree: EMF.cloud Tree Component for the tree
- Forms: JSON Forms for the forms
  - JSON Schema and UI Schema
- Integration: EMF.cloud model server
  - Typescript-based Model Server client API
Demo: Graphical Modeling
Involved components: Graphical Modeling

- **Graphical Language Server Platform (GLSP)**
  - LSP for Graphical Editors
  - GLSP client:
    - generic
    - renders graphical visualization
  - GLSP server:
    - specific to DSL
    - maps model to graphical visualization
    - synchronization with model server
  - Based on Eclipse Sprotty

![Involved components: Graphical Modeling](image)
Demo: Generators - Model to Text
Involved components: Generators

- Generator framework: Eclipse Xtend
- Generator: jar build with Maven
- Integration:
  - Launched on demand via CLI
  - Generates into selected Theia workspace folder
Demo: Working with source code
Involved components: Working with source code

- Code editor frontend: Theia Code Editor (LSP)
- Code editor backend: jdt.ls (JDT)
- Debugging frontend: Theia Debug Extension (DAP)
- Debugging backend: Java Debug Server
Demo: Textual Modeling

```
1 machine: SuperBrewer3000
2 workflow: BrewingFlow
3 probabilities
4 low : 0.1
5 medium : 0.5
6 high : 0.75
7 assertions
8 Preheat => Brew, Preheat =>
```

```
Brew
Check drip tray
Check Water
fffsdfsdf
Preheat
Push
Refill water
Water Ok
```
Involved components: Textual Modeling

- Text Editing: Theia Editor (Monaco)
- LSP Server: XText LSP Server
- Integration: Java-based Model Server client API
Demo: Model Analysis

50.0% of executions take this path
Involved components: Model Analysis

- Visualization: D3.js
- Analysis: Custom Kotlin code
- Integration: Java-based Model Server client API
Migration strategy: iterative, iterative, iterative...

- **Now:** Define a strategy and timeplan, build POC
- **Short-term:** Consider for architectural decisions
- **Mid-term:**
  - Prepare architecture for migration **iteratively**
  - Migrate high-value use cases **iteratively**
  - Single-source components, enable **reuse**
- **Long-term:**
  - Migrate use-case by use-case **iteratively**
  - Deprecate desktop-based solution
Summary

- **EMF.cloud**
  - Components to build web-based modeling tools
  - Example products
    - Coffee Editor: [Github](https://github.com)
    - Ecore Editor: [Github](https://github.com)
- **Building Web-based tools**
  - Feasible today
  - Reuse and migration facilitated
- **Live-Demo of coffee editor** available
- **More information:** [http://emf.cloud](http://emf.cloud)

→ **Important now:** Define strategy and timeplan, build POC
Evaluate the Sessions

Sign in and vote at EclipseCon.org:

-1 0 +1