Part I:

Self-generating Programs – Cascade of the Blocks

Part II:

State Machine Abstraction Layer

Josef Kufner

kufnejos@fel.cvut.cz

April 2014

Part I

Self-generating Programs – Cascade of the Blocks

Connecting components together

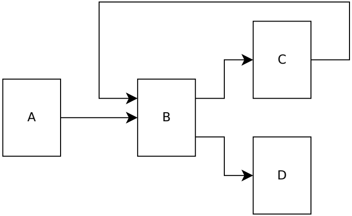
- **Unix pipeline:** (Douglas McIlroy, 1964/1973)

  cat | grep | sort | sed

  - Isolated programs
  - Uniform interfaces – input and output (stdio)
  - Single data stream
  - Static structure

Connecting components together

- **Function Blocks:** (IEC 61131, 1992)



  - Isolated blocks
  - Uniform interfaces – inputs and outputs
  - Multiple data streams
  - Static structure
  - Feedback is possible

What if . . .

Data streams + Static structure

What if . . .

Data streams + Static structure

$\Downarrow$

Static data + Dynamic structure

?

Static data + Dynamic structure = ?

- Preserved features:
  - Isolated **blocks**
  - Uniform interfaces – **inputs** and **outputs**

Static data + Dynamic structure = ?

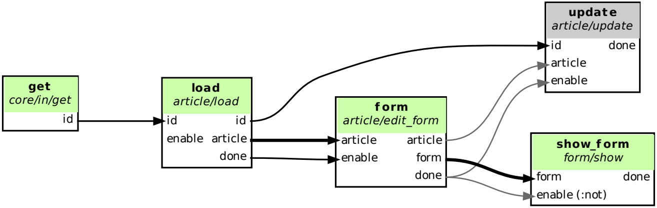
- Preserved features:
  - Isolated **blocks**
  - Uniform interfaces – **inputs** and **outputs**
- Static data?
  - Outputs can be set **only once**.
  - Inputs receive only a single value or an object.

Static data + Dynamic structure = ?

- Preserved features:
  - Isolated **blocks**
  - Uniform interfaces – **inputs** and **outputs**
- Static data?
  - Outputs can be set **only once**.
  - Inputs receive only a single value or an object.
- Dynamic structure?
  - Blocks are **created during evaluation**.
  - New blocks may be connected to the current structure.

The Cascade

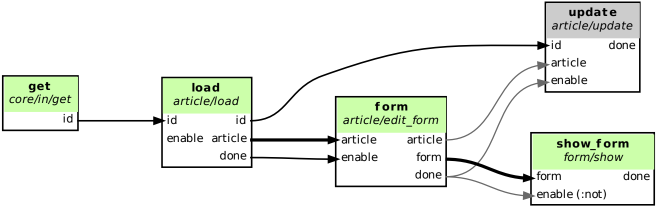
- The **Cascade** is a dynamic acyclic structure built of **blocks**.



- Values are passed from **outputs** of one block to **inputs** of another. The data are transferred as a single wave – no streams.

Evaluation of the Cascade

- Connections between blocks = Precedence constraints.
  - Output must be set before input is read.
- **Execution order is determined automatically.**
  - Programmer does not have to specify it explicitly – less work, more flexible cascade constructing.

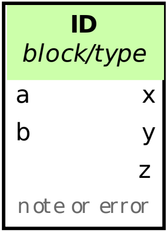


$$g \prec \overline{g},\ l \prec \overline{l}, \dots, \overline{g} \prec l,\ \overline{l} \prec u,\ \overline{l} \prec f,\ \overline{f} \prec u,\ \overline{f} \prec s$$

$$\implies g \prec \overline{g} \prec l \prec \overline{l} \prec f \prec \overline{f} \prec u \prec \overline{u} \prec s \prec \overline{s}$$

The Block

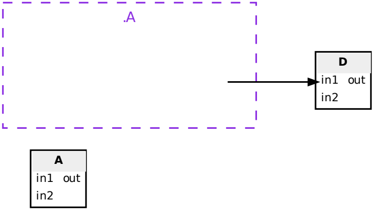
- Block is an OOP object with `main()` method.
  - Type (class)
  - ID of the instance
- Life time of the block:
  1. Read inputs.
  2. Process data.
  3. Set outputs.
- Strict encapsulation:
  Blocks do not know their connections.
- Similar to Function block, but semantics is different.



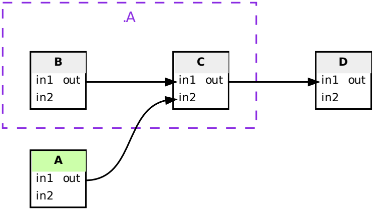
The Growing Cascade

- Blocks can **insert** additional blocks and connect their inputs.

Before execution of A:



After:

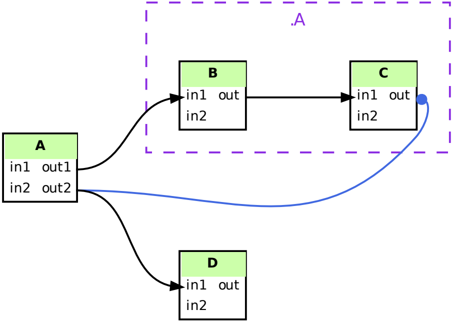


$$\overline{A} \prec D$$

(connection into namespace of A)

Namespaces

- Each block can insert blocks into its own namespace only.
- Connections can be established across all namespaces.
- Recursive nesting is allowed. – The cascade is a **3D structure**.

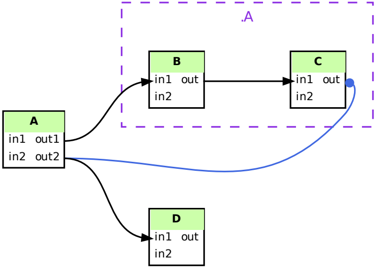


Block A inserted blocks B and C, and requested forward of the C's output.

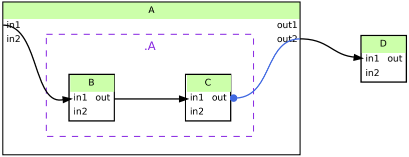
Output forwarding

- Return value replacement – nothing is going back in the cascade.
- Scope of forwarding is not limited by namespace.

Actual cascade:



Idea of nesting:



Block A inserted blocks B and C, and requested forward of the C's output.

Cascade features

- Automated visualization:
  - Cascade snapshot represents the entire previous evaluation.
  - Easy to render automatically using Graphviz.
  - **Debugger in a single picture.**

Cascade features

- Automated visualization:
  - Cascade snapshot represents the entire previous evaluation.
  - Easy to render automatically using Graphviz.
  - **Debugger in a single picture.**
- Data flow orientation:
  - Modelling data paths instead of data types or algorithms.
  - **Cascade says what happens to data, not how.**
  - Details are contained in blocks.

Cascade features

- Automated visualization:
  - Cascade snapshot represents the entire previous evaluation.
  - Easy to render automatically using Graphviz.
  - **Debugger in a single picture.**
- Data flow orientation:
  - Modelling data paths instead of data types or algorithms.
  - **Cascade says what happens to data, not how.**
  - Details are contained in blocks.
- Strict block encapsulation and unified API
  - **Block can be replaced, cascade reconfigured.**
  - Any input can be connected to any output (if it makes sense).
  - Higher code reusability. Limited scope for bugs.
  - Side-effects :(

Cascade composition

- Cascade is:
  - simple data structure
  - declarative
  - **machine-friendly**

Cascade composition

- Cascade is:
  - simple data structure
  - declarative
  - **machine-friendly**
- Cascade is designed to be generated on-the-fly.
- **Ready for sophisticated composition mechanisms.**
  - More in Part II.

Cascade usage

- Designed for **non-interactive applications**.
    - HTTP server:

$$\text{One HTTP request} = \text{one cascade evaluation.}$$

Cascade usage

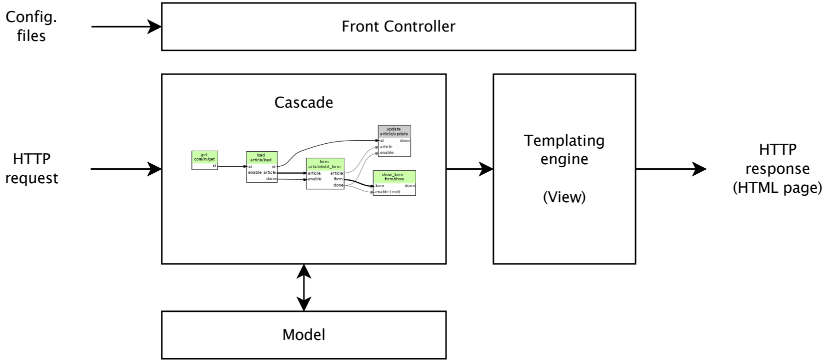
- Designed for **non-interactive applications**.
  - HTTP server:

    One HTTP request = one cascade evaluation.

- It is hard to implement cycles.
  - Usualy not required in real applications.
  - Pass **lists** and **iterators** between blocks to process collections.

Use Case: Web Framework

- Cascade was created as a core of a web framework.
- Cascade replaced a traditional controller in MVC.
- Blocks can produce output fragments, which are composed into a web page after the cascade evaluation is finished.
- Push architecture.



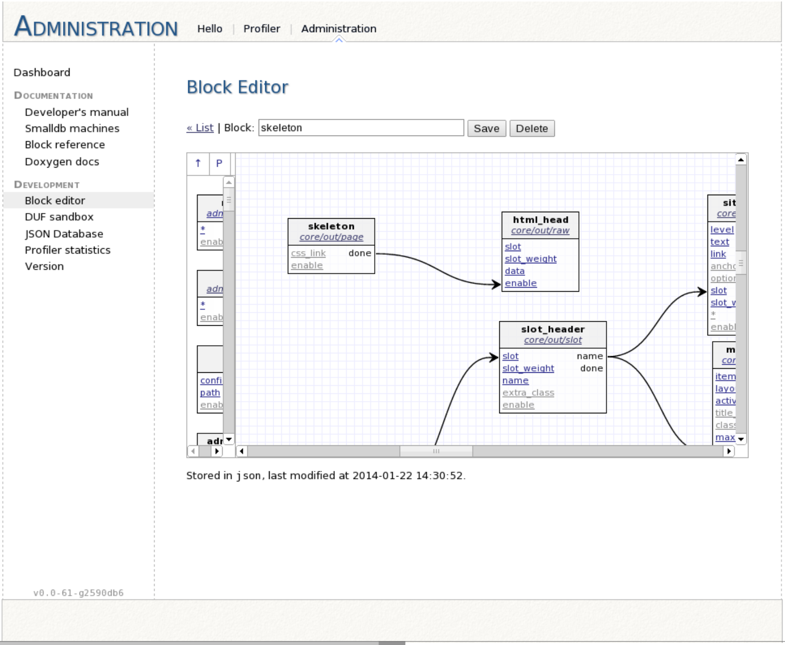
Framework Features

- Real-time fully automatic **cascade visualization**:
  - Every web page can contain an automatically generated diagram of the cascade which generated the page (Graphviz).
  - **Easy to trace where data come from and what happened to them.**
- Visual cascade editor
  - User-friendly web application composition – both **logic** and layout.
- Plugin infrastructure
  - Plugin is a library of blocks + config.
  - Good code reusability (prototypes).
- Generated block documentation
  - Fully integrated into an application.
  - Less code to remember.

Movie

- Automaticaly generated movie !
- Shows how the cascade is evaluated.
- Silent movie only. Sorry.

Movie – the result



Block Editor

« List | Block: skeleton

Stored in json, last modified at 2014-01-22 14:30:52.

Current Research

- Automatic cascade composition from incomplete specification.
  - Combining existing implementations, relevant metadata, and various forms of user input to generate a new implementation.
- **Cascade is designed for automated processing:**
  - Simple data structures.
  - Everything is declarative and machine-friendly.

$$\text{Customer} \longleftrightarrow \begin{array}{c}\text{Software}\\\text{Analyst}\end{array} \longleftrightarrow \underbrace{\begin{array}{c}\text{Trained}\\\text{Monkey}\end{array} \longrightarrow \begin{array}{c}\text{Framework}\\\text{(PHP)}\end{array}}_{\textit{To be replaced.}}$$

End of Part I

Josef Kufner
kufnejos@fel.cvut.cz

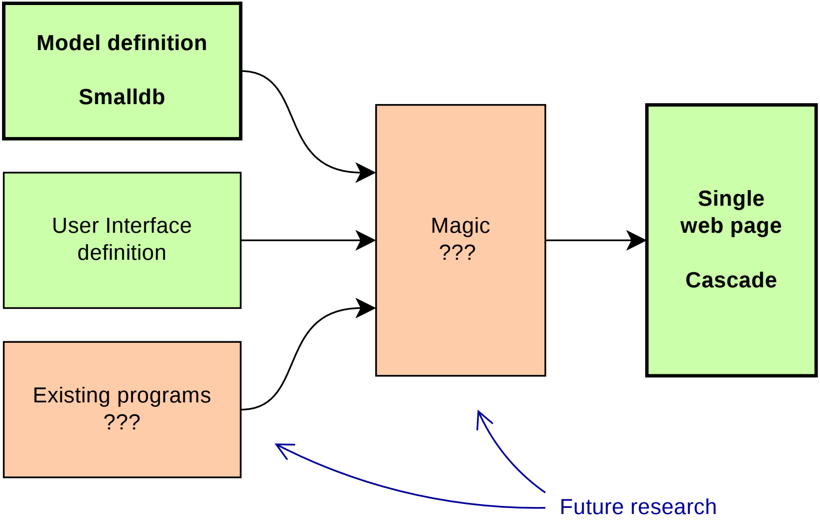
Framework demo:
http://cascade.frozen-doe.net/

Comming soon: *Magic vs. Trained monkeys*

Part II

State Machine Abstraction Layer

The Big Picture



Model definition

Smalldb

User Interface definition

Existing programs ???

Magic ???

Single web page

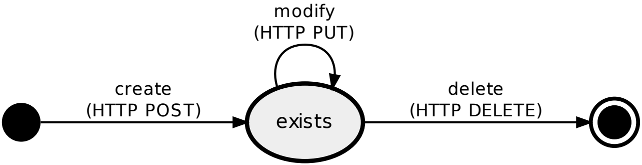
Cascade

Future research

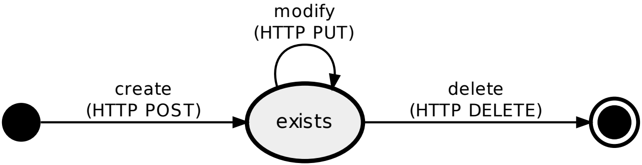
What is Smalldb?

- Smalldb is a framework for creating models in MVC-like applications.
  - But it is not only a model.
- Smalldb is RESTful.
  - But a little different from usual REST applications with HTTP API.
- Smalldb use state machines to describe the model …

REST Resource as a State Machine

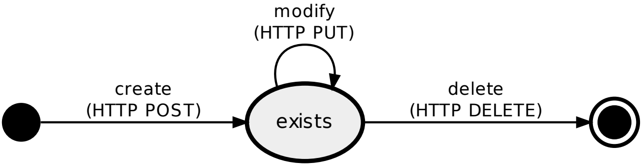


REST Resource as a State Machine



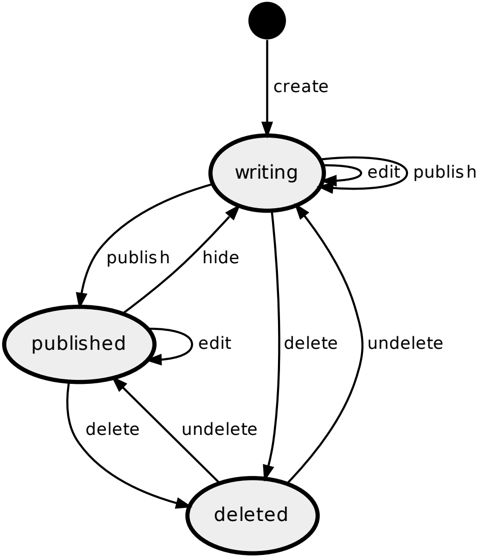
- How to undelete a resource?
- How to manage long-running tasks?

REST Resource as a State Machine



- How to undelete a resource?
- How to manage long-running tasks?
- **What if we add more transitions?**

Example: An article in a content management system



REST API for Smalldb

- Required operations:
  1. Read state (HTTP GET)
  2. Invoke a transition (HTTP POST)
     - transition name
     - parameters

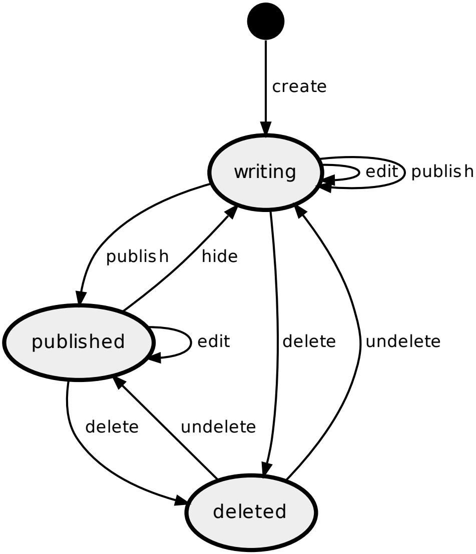
REST API for Smalldb

- Required operations:
  1. Read state (HTTP GET)
  2. Invoke a transition (HTTP POST)
     - transition name
     - parameters
- REST is not only HTTP API.
  - Uniform interface – Resources, URL
  - Hypermedia – Resources linking to each other.
  - Stateless communication

REST API for Smalldb

- Required operations:
  1. Read state (HTTP GET)
  2. Invoke a transition (HTTP POST)
     - transition name
     - parameters
- REST is not only HTTP API.
  - Uniform interface – Resources, URL
  - Hypermedia – Resources linking to each other.
  - Stateless communication
- Smalldb preserves REST features.
- Compatible with good old HTML forms.
  - No complex clients needed.

Example: An article in a content management system



- What self-loops do?

Finite automaton + Kripke structure

- Self-loops may change the state!

Finite automaton + Kripke structure

- Self-loops may change the state!

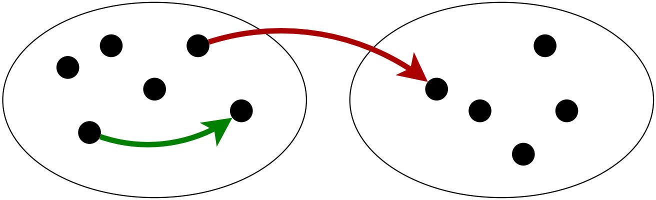
- State machine has **properties** (key–value).
- **State is function of the properties.**

Finite automaton + Kripke structure

- Self-loops may change the state!

- State machine has **properties** (key–value).
- **State is function of the properties.**

- Self-loop is transition between sub-states within the state.



Definition of Smalldb State Machine … see the paper.

Smalldb state machine is modified non-deterministic parametric finite automaton, defined as a tuple $(Q, P, s, P_0, \Sigma, \Lambda, M, \alpha, \delta)$, where:

- Q is finite set of states.
- P is set of named properties. P^* is (possibly infinite) set of all possible values of P. P_t is state of these properties in time t. $P_t \in P^*$.
- s is state function $s(P_t) \mapsto q$, where $q \in Q$, $P_t \in P^*$.
- P_0 is set of initial values of properties P, $P_0 \in P^*$.
- Σ is set of parametrized input events.
- Λ is set of parametrized output events (optional).
- M is finite set of methods: $m(P_t, e_{in}) \mapsto (P_{t+1}, e_{out})$, where $P_t, P_{t+1} \in P^*$, $m \in M$, $e_{in} \in \Sigma$, $e_{out} \in \Lambda$.
- α is assertion function: $\alpha(q_t, m) \mapsto Q_{t+1}$, where $q_t \in Q$, $Q_{t+1} \subset Q$, $e_{in} \in \Sigma$.

$$\forall m \in M : s(P_{t+1}) \in \alpha(s(P_t), m) \Leftrightarrow (\exists e_{in} : m(P_t, e_{in}) \mapsto (P_{t+1}, e_{out}))$$

- δ is transition function: $\delta(q_t, e_{in}, u) \mapsto m$, where $q_t \in Q$, $e_{in} \in \Sigma$, $m \in M$, and u represents current user's permissions and/or other session-related attributes.

Key features of Smalldb State Machine (1/2)

- Nondeterministic parametric finite automaton.

Key features of Smalldb State Machine (1/2)

- Nondeterministic parametric finite automaton.
- … finite automaton
  - Finite set of **states** and **transitions**.
  - Single initial state.

Key features of Smalldb State Machine (1/2)

- Nondeterministic parametric finite automaton.
- … finite automaton
  - Finite set of **states** and **transitions**.
  - Single initial state.
- … parametric …
  - State is a function of **named properties** (key–value structure).
  - **State function** is one-way mapping.

Key features of Smalldb State Machine (1/2)

- Nondeterministic parametric finite automaton.
- … finite automaton
  - Finite set of **states** and **transitions**.
  - Single initial state.
- … parametric …
  - State is a function of **named properties** (key–value structure).
  - **State function** is one-way mapping.
- Nondeterministic …
  - Multiple transition of the same name.
  - Transition may fail, or it depends on unknown variables.
  - Equivalent to deterministic automaton with guards.

Correctness and Provability

- Smalldb separates formally provable definition and a messy code with transition implementations.

Correctness and Provability

- Smalldb separates formally provable definition and a messy code with transition implementations.
- Formal model (state machine definition) is part of implementation.
  - Almost no space for mistakes while converting formal model to a real code.

Correctness and Provability

- Smalldb separates formally provable definition and a messy code with transition implementations.
- Formal model (state machine definition) is part of implementation.
    - Almost no space for mistakes while converting formal model to a real code.
- Easy to visualize.
    - Graphviz (again)
    - Costumer may understand state diagram and confirm validity. (No chance to do so with source code.)
    - Easier for new programmers to start working on an old code.

Key features of Smalldb State Machine (2/2)

- Implementation of a transition?

Key features of Smalldb State Machine (2/2)

- Implementation of a transition?

- Transition is implemented in code as OOP method.
- State machine validates a state after a transition is finished using **assertion function**.

Key features of Smalldb State Machine (2/2)

- Implementation of a transition?

- Transition is implemented in code as OOP method.
- State machine validates a state after a transition is finished using **assertion function**.

- Messy code is packed and supervised.
- Machine implementation is well tested.
- Machine definition can be formally verified.

What could go wrong?

Metadata

- State machine definition can be easily extended with related metadata.
- Convenient „Single Source of Truth“.

Metadata

- State machine definition can be easily extended with related metadata.
- Convenient „Single Source of Truth“.
- Parts of application may be generated from these metadata.
  - User interface, …
- Access control – per transition.

Interaction with outter world

- Cooperating state machines can be modeled and formally verified.
- Other entities in a bussiness process may be modeled as a state machines too.
- Possibility to formally verify entire bussiness process.

Future research

- Smalldb was created as a source of metadata for the „magic“ part.
- Who wants to play with state machines ?

Thank you !

Josef Kufner

kufnejos@fel.cvut.cz

To be continued . . .