LiveWeb
Core Language for Web Applications

Miguel Domingues João Costa Seco

CITI – Departamento de Informática – FCT/UNL

Most Web Application Development is not Type Safe

- Heterogeneous development environments
  - User Interface, Business Logic and Database
  - Programming languages and tools
- Ad-hoc integration code
  - Queries and data as strings (no static checks, hard to change)
- Object-Relational Mappings (safer but less efficient)
- Explicit coding of sophisticated features
  - Authentication, access control, confidentiality, resource usage
  - No real support for code evolution

Web Development Frameworks

- Dynamic Language Frameworks
  - Ruby On Rails, CakePHP
  - Dynamically typed, scaffolding code generation
- General Purpose Language Extensions
  - ScalaQL (for-comprehensions queries) [Spiewak2010], LINQ (specific query syntax)
  - Typing of database operations, user interface built as HTML strings
- Domain Specific Languages
  - OutSystems DSL, Ur/Web [Chlipala2010], WebDSL [Visser2008], Links [Cooper2006]
  - Basic static verifications, code generation, higher abstraction level
- Our goal is to provide a language that can leverage the verification of web applications
  - Certified Interfaces – NGN44-CMUPortugal
  - Security, confidentiality, dynamic reconfiguration
- Typed core language with primitive interface and database operations

Core Language for Web Applications – Syntax

$$
\begin{array}{lll}
\mathcal{D} ::= & & \text{(Definitions)} \\
& \textbf{def entity } t \ \{ \ label \ : \ \textbf{Id}, \ \overline{label \ : \ \mathcal{BT}} \ \} & \text{(Entities)} \\
| & \textbf{def action } a \ (\ \overline{x : \mathcal{T}} \ ) : \mathcal{T} \ \{ \ e \ \} & \text{(Actions)} \\
| & \textbf{def screen } s \ (\ \overline{x : \mathcal{T}} \ ) \ \{ \ b \ \} & \text{(Screens)}
\end{array}
$$

$$
\begin{array}{lll}
e ::= & & \text{(Expressions)} \\
& \textbf{insert } e \textbf{ in } t & \text{(Entity Insert)} \\
| & \textbf{update } x \textbf{ in } t \textbf{ with } e \textbf{ where } e & \text{(Entity Update)} \\
| & \textbf{from } (\ \overline{x \textbf{ in } t} \ ) \textbf{ where } e \textbf{ select } e & \text{(Entity Select)} \\
| & \textbf{let } x = e \textbf{ in } e & \text{(Variable Declaration)} \\
| & \textbf{if } e \textbf{ then } e \textbf{ else } e & \text{(Condition)} \\
| & \textbf{foreach } x \textbf{ in } e \textbf{ do } e & \text{(List Iterator)} \\
| & \ldots &
\end{array}
$$

$$
\begin{array}{lll}
b ::= & & \text{(Web Page Blocks)} \\
& \textbf{link } \{ \ b \ \} \textbf{ to } e & \text{(Link)} \\
| & \textbf{iterator } (\ x \textbf{ in } e \ ) \ \{ \ b \ \} & \text{(Iterator)} \\
| & \textbf{textfield } x \textbf{ with } e & \text{(Text Field)} \\
| & \textbf{button } e \textbf{ to } e & \text{(Button)} \\
| & \ldots &
\end{array}
$$

$$
\begin{array}{lll}
\mathcal{BT} ::= & & \text{(Basic Types)} \\
& \mathit{String} & \text{(Strings)} \\
| & \mathit{Int} & \text{(Integers)} \\
| & \mathit{Bool} & \text{(Booleans)} \\
| & \mathit{entity}.\textbf{Id} & \text{(Entity Identifier)}
\end{array}
$$

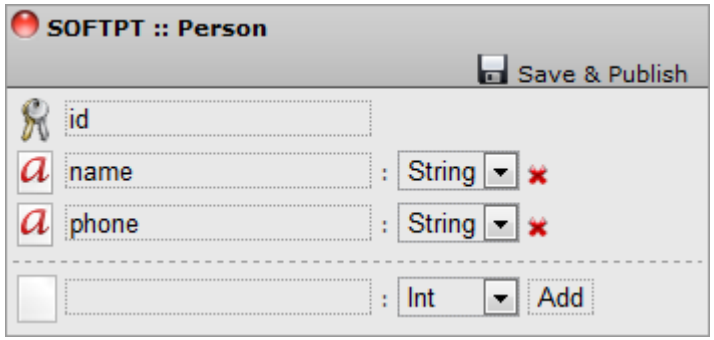
$$
\begin{array}{lll}
\mathcal{T} ::= & & \text{(Types)} \\
& \mathcal{BT} & \text{(Basic Type)} \\
| & \mathit{Block} & \text{(Web Page Block)} \\
| & \{ \ \overline{label \ : \ \mathcal{T}} \ \} & \text{(Structure)} \\
| & \mathit{List}\langle \mathcal{T} \rangle & \text{(List)}
\end{array}
$$

Example

```
def entity Person { id:Id, name:String, phone:String }
```

Identifiers

Types (String, Int, Bool)



Database table with simple integer Primary Key

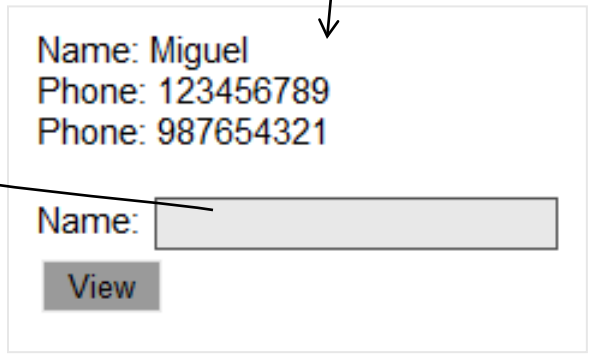
Example

```
def entity Person { id:Id, name:String, phone:String }

def screen userDetail(nm:String) {
    label "Name: " + nm; br;
    iterator (row in (from (p in Person) where p.name==nm select p)) {
        label "Phone: " + row.phone; br
    }; br;

    label "Name: "; textfield name; br;
    button “View" to userDetail(name)
}
```

From Query Expanded



Screen Call
(textfield input data as argument)

Example

```
def entity Person { id:Id, name:String, phone:String }

def screen userDetail(nm:String) {
    label "Name: " + nm; br;
    iterator (row in (from (p in Person) where p.name==nm select p)) {
        label "Phone: " + row.phone; br
    }; br;

    label "Name: "; textfield name; br;
    button “View" to userDetail(name)
}

def action addPerson(nm:String, ph:String):Block {
    insert { name = nm, phone = ph } in Person;
    userDetail(nm)
}
```

Application flow in the language

Insert Query Expression

Semantics

Standard semantics with store and lists

- Action Call (call-by-value)

$$\frac{P(a) = a(\overline{x})\{e\} \quad S_i; f_i \Downarrow S_{i+1}; g_i \qquad S_{n+1}; e\{\overline{g}/\overline{x}\} \Downarrow S'; v \quad i = 1, \cdots, n}{S_1; a\ (\overline{f}) \Downarrow S'; v}$$

- Insert Query

$$\frac{S(t) = [v_1, \cdots, v_n] \quad S; e \Downarrow S'; v \quad S'(t) = [v_1, \cdots v_n, v] \quad \forall i, v \neq v_i}{S; \textbf{insert}\ e\ \textbf{in}\ t \Downarrow S'; \textbf{true}}$$

Type System

Standard type system rules

- Action Call

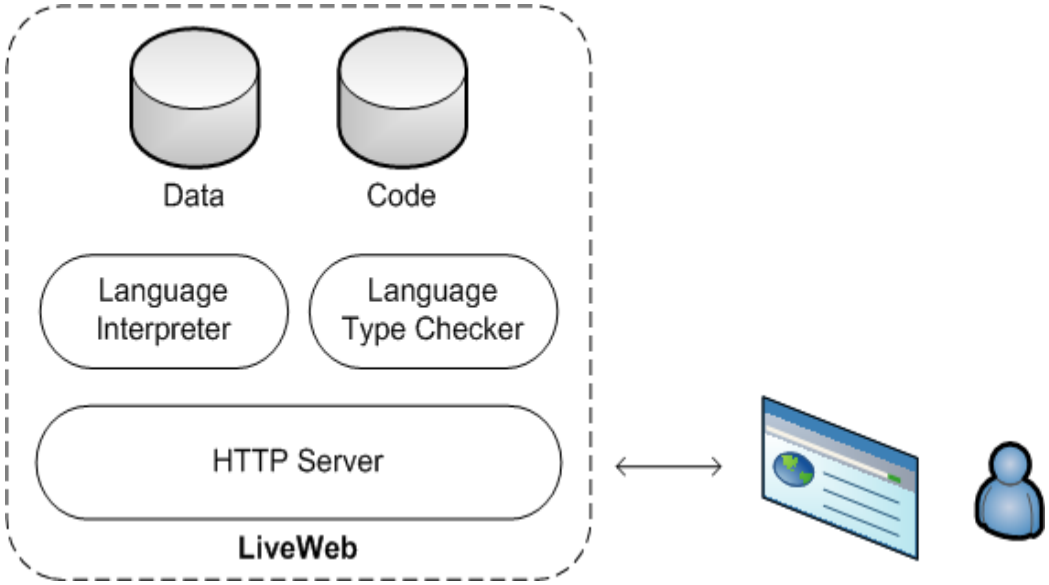
$$\frac{P(a) = a(\overline{x : \mathcal{T}}) : \mathcal{T}_r \, \{ \cdots \} \qquad \Delta, x_i : \mathcal{T}_i \vdash e_i : \mathcal{T}_i \qquad i = 1, \cdots, n}{\Delta \vdash a(\overline{e}) : \mathcal{T}_r}$$

- From Query

$$\frac{\Delta \vdash t : List\langle \mathcal{T} \rangle \qquad \Delta, x : \mathcal{T} \vdash e : Bool \qquad \Delta, x : \mathcal{T} \vdash f : U}{\Delta \vdash \textbf{from} \; (x \; \textbf{in} \; t) \; \textbf{where} \; e \; \textbf{select} \; f : List\langle U \rangle}$$

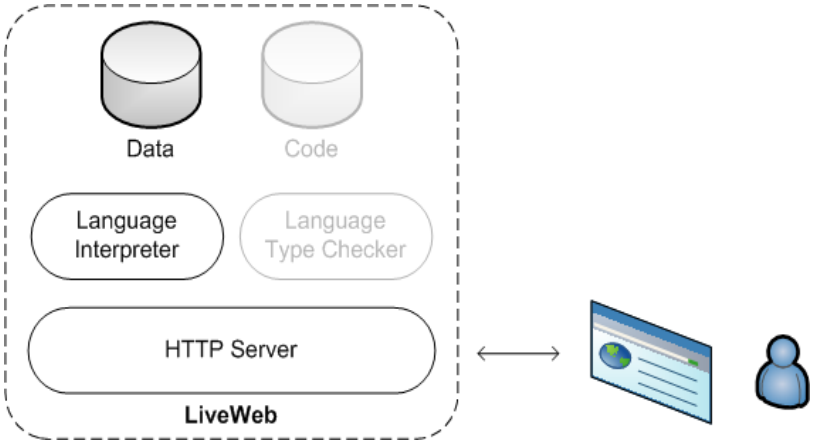
Runtime Support System

- Web based development environment
  - Language interpreter and type checker
- Version control
- Dynamic reconfiguration



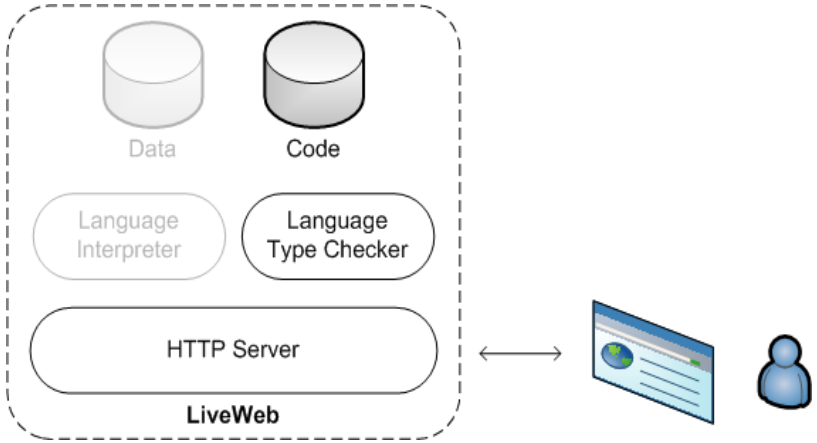
Runtime Support System – Execution Mode

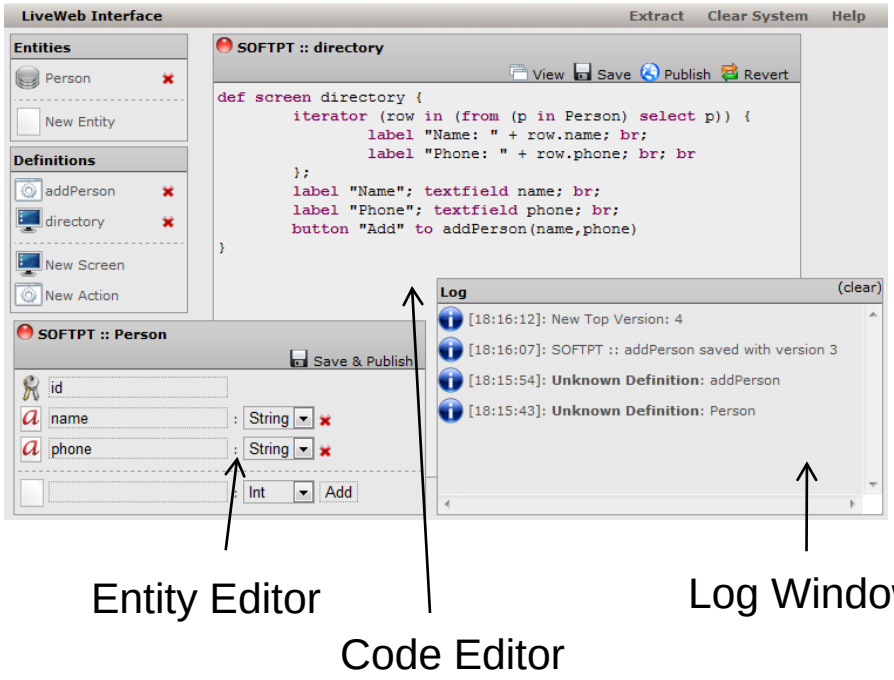
- Evaluation in the server side
- Applications parameters passed through standard URLs conventions
  - http://server:port/module/element/arg0/arg1/.../
- Screens are obtained by evaluating interface expressions



Runtime Support System – Development Mode

- Web based development environment
  - Create, modify and delete application elements
- Dynamic reconfiguration
  - After submitting a modification the new definitions are checked and activated
- Version control
  - Active version is always well typed





Ongoing and Future Work

- ✓ Extended to demonstrate security related properties based on refinement types [Freeman1991]
- Web features like AJAX, sessions, cookies, etc.
- Extension of the language with modules
- Lazy query evaluation and query optimization
- Improve closure support

Demo available during the break

Project Homepage: http://ctp.di.fct.unl.pt/INTERFACES/