QWeSST

Type-Safe Web Programming

Thierry Sans and Iliano Cervesato

Carnegie Mellon University Qatar
Just start **typing**...

...and share the link above with a friend to try **collaborating** in real-time.

Note: This demo document will only be available for 24 hours from the time it was created. To start using Google Docs and create your own docs, **sign up for an account**.
• Project Goal

  ➡ Study the foundations of web programming

• Outcomes

  ➡ **QWeSST**: a type-safe programming language for the web
  ➡ Faithful semantics description for parallel languages
  ➡ **QWeSST**\(\phi\): managing distributed flow of data on the web
Web Programming
Anatomy of a Web Application

- Mobile code
- Remote execution
- State
- Security
Limitation of current web technologies

➡ Use of heterogeneous languages
   (not originally designed with distributed computing in mind)

➡ Require heavy testing

○ Setting up the communication machinery is expensive and error prone
Partial solution – Better libraries

- Simplifying the communication machinery
  ➡ Abstract libraries (such as JQuery and Prototype)
  - But we still have to care about requests and callbacks
Partial solution – **One** language

Write an entire webapp in the same language

➡ Google Web Toolkit, LINKS, HOP

- Programmer designates code as client or server
- Compiled to JavaScript or Java

➡ Flash, Silverlight

- Interpreted in the browser
Complexity is rising

• Webapps are getting more and more sophisticated and distributed

○ Current technologies are unlikely to be able to support this growing complexity
QWeSST

A Type-Safe Programming Language for the Web
Looking for foundations of web programming

- A language to carry out local computations
  ✓ A \(\lambda\)-calculus

- Constructs to **publish** code and **call** it through a URL
  ✓ Remote procedure mechanism

- Constructs to **suspend** and **resume** a computation
  ✓ Mobile code

in a well-typed fashion
Remote Procedures

Types \( \tau ::= \ldots | \tau \rightarrow \tau' \)

Expressions \( e ::= \ldots | \text{w/u} | \text{publish } x: \tau. \ e | \text{call } e_1 \ \text{with } e_2 \)

- Browser to web server
  - Web pages
  - Ajax
- Web server to web server
  - XML/RPC (web service)
A new service has been published at www.server.com/fact/

Server

```
let
fun fact(n) =>
    if n = 0 then 1
    else n * fact(n-1)
in
publish x => fact(x)
```

Client

```
let
fun f(x) =>
    call url('www.server.com/fact/') with x
in
f(4) + f(6)
```

- **fact(4)** calculates 24
- **fact(6)** calculates 720
- **(24+720)** calculates 744
Mobile Code

Types \[ \tau ::= \ldots \mid \text{ susp}[\tau] \]

Expressions \[ e ::= \ldots \mid \text{ hold } e \mid \text{ resume } e \]

- Web server to browser
  - Javascript code
- Web server to web server
  - Not done in practice
A new service has been published at www.server.com/fact/

Server

```ocaml
let
fun fact(n) => if n= 0 then 1
  else n * fact(n-1)
in
  publish x => hold(fact)
```

Client

```ocaml
let
f = resume (call url('www.server.com/fact/') with ()
in
f(4) + f(6)
```

Calculations:
- `fact(4)` calculates 24
- `fact(6)` calculates 720
- `(24+720)` calculates 744
Web pages vs. Web services

✓ Web pages and web services are treated uniformly

➔ It is all about calling a URL (with some parameters) and getting a result back

➔ The difference is how the result is used
QWeSST - A language for web programming

- A simple abstraction of the way we program the web
  ✔ Easier to reason about complex web programs

- Currently a **pure** language (no effects)
- **Static** and **localized** type semantics
  - **Localized** type checking
  ✔ **Globally type safe** language
More examples

- Custom Web Service
- Web API
- Custom Web API
- Web service auto-installer

➡ Check the Qwesst website:
http://tsans-mac.qatar.win.cmu.edu/
An API

Server

```
let
search = url('www.server.com/search/')
script = hold (fn x => call search with x)
in
publish x => script
```

Client

```
let
api = url('www.server.com/api/')
s = resume (call api with ())
in
s('myRequest')
```

A new service has been published at www.server.com/api/
A Web Service Auto-installer

Server

```plaintext
let
search = url('www.server.com/search/')
f = (fn x => call search with x)
script = hold (publish x => f(x))
in
publish x => script
```

A new service has been published at
www.server.com/inst/

Customer

```plaintext
let
installer = url('www.server.com/inst/')
in
resume (call installer with ())
```

A new service has been published at
www.server.com/inst/

Client

```plaintext
let
f = url('www.client.com/search/')
in
call f with 'myQuery'
```

A new service has been published at
www.client.com/search/
Demo

Server's Qwesst Page

```
let fun hello (name:string):string => 'hello ' + name + '

in publish x:string => (hello x)
```

Documentation

<table>
<thead>
<tr>
<th>Id:</th>
<th>70300</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>srv[string][string]</td>
</tr>
<tr>
<td>url</td>
<td><a href="http://localhost/~tsans/qwesst/server/70300/">http://localhost/~tsans/qwesst/server/70300/</a></td>
</tr>
</tbody>
</table>
QWeSST

Formal Semantics
Typing

- Inspired to *ML5’s type system for localized computation* by Tom Murphy VII, Karl Crary and Robert Harper
Typing Semantics

Remote Procedure Call

\[
\begin{align*}
\tau &\triangleright \tau' \text{ mobile} \\
\Sigma, \ w/u: \tau &\triangleright \tau'; \ \Gamma |-w \ w/u : \tau \triangleright \tau'
\end{align*}
\]

\[
\begin{align*}
\tau &\triangleright \tau' \text{ mobile} \\
\Sigma; \ \Gamma, \ x : \tau &|-w \ e : \tau'
\end{align*}
\]

\[
\begin{align*}
\Sigma; \ \Gamma |-w \ \text{publish } x:\tau. \ e &: \tau \triangleright \tau'
\end{align*}
\]

\[
\begin{align*}
\Sigma; \ \Gamma |-w \ e_1 &: \tau \triangleright \tau'
\end{align*}
\]

\[
\begin{align*}
\Sigma; \ \Gamma |-w \ e_2 &: \tau
\end{align*}
\]

\[
\begin{align*}
\Sigma; \ \Gamma |-w \ \text{call } e_1 \text{ with } e_2 &: \tau'
\end{align*}
\]

Mobile Code

\[
\begin{align*}
\Sigma; \ \Gamma |-w \ e &: \tau
\end{align*}
\]

\[
\begin{align*}
\Sigma; \ \Gamma |-w \ \text{hold } e &: \text{ susp} [\tau]
\end{align*}
\]

\[
\begin{align*}
\Sigma; \ \Gamma |-w \ \text{resume } e &: \tau
\end{align*}
\]
Evaluation

\[ \Delta; e \Rightarrow_w \Delta'; e' \]

\( (\Delta; e \text{ steps to } \Delta'; e') \)
Evaluation Semantics

Remote Procedure Call

\[ \Delta; \text{publish } x:\tau. e \Rightarrow_w (\Delta, w/u = x:\tau. e); w/u \]

\[ \nu_2 \text{ val} \]
\[ (\Delta', w'/u = x:\tau. e); \text{call } w'/u \text{ with } \nu_2 \Rightarrow_w \Delta; \text{expect } [\nu_2/x] e \text{ from } w' \]
\[ \Delta \]

\[ \Delta; e \Rightarrow_w \Delta'; e' \]
\[ \Delta; \text{expect } e \text{ from } w' \Rightarrow_w \Delta'; \text{expect } e' \text{ from } w' \]

\[ \nu \text{ val} \]
\[ \Delta; \text{expect } \nu \text{ from } w' \Rightarrow_w \Delta; \nu \]

Mobile Code

\[ \Delta; e \Rightarrow_w \Delta'; e' \]
\[ \Delta; \text{resume } e \Rightarrow_w \Delta'; \text{resume } e' \]
\[ \Delta; \text{resume (hold } e) \Rightarrow_w \Delta; e \]
Meta-theory

✓ QWeSST is **type safe** (proof verified using Twelf)

▶ Type preservation

If $\Sigma; . |-_{w} e : \tau$ and $\Sigma |- \Delta$ and $\Delta; e \Rightarrow_{w} \Delta'; e'$, then $\Sigma'; . |-_{w} e' : \tau$ and $\Sigma' |- \Delta'$

▶ Progress

If $\Sigma; . |-_{w} e : \tau$ and $\Sigma |- \Delta$, then

- either $e$ val
- or $\Delta; e \Rightarrow_{w} \Delta'; e'$
Parallel Semantics
A Semantic Mismatch

\[ \Delta; e \xrightarrow{w} \Delta'; e' \]

- One expression at a time is evaluating
  - Single-threaded

- This is not the way the web works
  - Millions of executions occurring simultaneously
  - Possibly on the same node
Serialized semantics

- Parallelism reduced to non-deterministic interleaving
- Macro-step as series of micro-steps

\[
\Delta; \cdot \Rightarrow \Delta; \cdot \\
\Delta; (e \oplus w, E) \Rightarrow (\Delta, \Delta', \Delta''); (e' \oplus w, E')
\]

- Serialized typing semantics

\[
\Sigma; \cdot \vdash_w e : \tau \\
\Sigma \vdash (e \oplus w, E) : \tau, T
\]

- Serialized safety proof if working with sequences
- Large overhead if working with multisets
Multiset-Oriented Rules

• Rules can talk about multisets
• Rules can have multisets of premises
• Specified by **parametric multiset comprehension**

\[
\frac{\{ e_i \text{ val} \}}{\{ e_i @ w_i \text{ final} \} (i \in I)}
\]
Linear Destination Passing Style

• “Branching” stack machine with explicit return addresses
  • \((e)^d\) – evaluate \(e\) for \(d\)
  • \((\nu)^d\) – return \(\nu\) to \(d\)
  • \((\text{call } d_1 \text{ with } d_2)^d\) – wait for results

\[
\text{(hold } e)^d \Rightarrow_w (\text{hold } e)_d
\]

\[
(\text{resume } e)^d \Rightarrow_w (\text{resume } d')^d, (e)^{d'}
\]

\[
(\text{resume } d')^d, (\text{hold } e)^{d'} \Rightarrow_w (e)^d
\]
LDP rules for call

\[(\text{call } e_1 \text{ with } e_2)^d \Rightarrow_w (\text{call } d' \text{ with } d'')^d, (e_1)^d', (e_2)^d''\]

\[
\frac{w'/u = x:\tau. \ e \in \Delta}{(\text{call } d' \text{ with } d'')^d, (w'/u)^d', (v)^{d''} \Rightarrow_w (\text{expect } d''\text{ from } w')^d
\cdot \Rightarrow_{w'} ([v/x]e)^{d''}}
\]

\[
\frac{\nu' \ \text{val}}{(\text{expect } d''\text{ from } w')^d \Rightarrow_w (\nu')_d
\left(\nu'ight)^{d'''} \Rightarrow_{w'} \cdot}
\]
Orchestration

- Evaluation

\[
\frac{\{ \Delta; e_i \Rightarrow w_i (\Delta, \Delta_i); e_i' \}}{\Delta; \{ e_i @ w \}, E \Rightarrow (\Delta, \{ \Delta_i \}); \{ e_i' @ w \}, E} \quad (i \in I)
\]

- Typing

\[
\frac{\{ \Sigma; d_i : \tau_i \mid w_i e_i \}}{\Sigma; \{ d_i : \tau_i \} \mid \Delta; \{ e_i @ w_i \}} \quad (i \in I)
\]

Simplified for typesetting reasons
### Substructural meta-theory

<table>
<thead>
<tr>
<th>Local</th>
<th>Type Preservation</th>
<th>Progress</th>
</tr>
</thead>
<tbody>
<tr>
<td>If $\Sigma; \sigma: \tau</td>
<td>-_w e$ and $\Sigma</td>
<td>- \Delta$ and $\Delta; e \Rightarrow_w \Delta'; e'$, then $\Sigma'; \sigma: \tau</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Global</th>
<th>Type Preservation</th>
<th>Progress</th>
</tr>
</thead>
<tbody>
<tr>
<td>If $\Sigma; \Lambda</td>
<td>- \Delta; E$ and $\Delta; E \Rightarrow \Delta'; E'$, then $\Sigma'; \Lambda</td>
<td>-_w \Delta'; e'$</td>
</tr>
</tbody>
</table>
Managing Data Flow on the Web
Services [use other services]*

• How does a service provider describe data paths through the web?

• How can a client control where her data goes?
Scenario

Client → Service provider → Third-party service providers

url(w₀,u₀) → url(w₁,u₁) → url(w₂,u₂)
url(w₃,u₃) → url(w₄,u₄) → url(w₅,u₅)

Services

Data Flow
Describing data paths

\[ \mu ::= \bullet \mid w; \mu \mid \mu \circ \mu' \mid \mu \parallel \mu' \]
$w_0; (w_1; (w_2 \circ w_3)) \circ (w_4 \parallel w_5)$
Describing flow policies

\[ \rho ::= T \mid F \mid \neg \rho \mid \rho \land \rho' \mid \rho \lor \rho' \mid \cdot \mid w; \rho \mid \rho \circ \rho' \mid \{w_i\}^*; \rho \mid \{w_i\}?; \rho \mid (\rho)^* \circ \rho' \mid (\rho)? \circ \rho' \]

➡️ Can describe

✓ Basic permissions and prohibitions
✓ Strict sequencing (e.g., anonymization policies)
✓ Flow isolation (a la Chinese wall policy)
Incorporating paths and policies into Qwesst

- Data paths in local and remote function types

\[\tau ::= \ldots \mid \tau[\mu] \rightarrow \tau' \mid \tau[\mu] \leftarrow^w \tau'\]

- Type annotations are inferred

- Policies in call

\[\text{call } e_1 \text{ with } e_2[\rho]\]
Incorporating paths and policies into Qwesst

- Flow inference and control in type checking

\[
\Sigma; \Gamma |-_w e_1 : \tau\#[\mu] \triangleright^w \tau' \quad \Sigma; \Gamma' |-_w e_2 : \tau \quad \mu \models \rho \\
\Sigma; (\Gamma \ll (\Gamma' \circ (w' ; \mu))) |-_w \text{call } e_1 \text{ with } e_2 [\rho] : \tau'
\]

- Evaluation remains unchanged
Meta-theory

- The language remains type safe
Perspectives and Future Work
Short Term

- More expressive constructs and data structures
- Features for “real” web development
  - Browser embedded interpreter
  - DOM implementation

✓ We want to build a higher level language that relies on Javascript and markup languages
Longer Term

• More security
• Effects & concurrency
• A way to track and manage dead links
• A logical framework based on multiset comprehension
Thank You

Any Question?