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](http://docs1.google.com/demo/edit?id=scACRSmYFK48VguV9WdrFKPCk#document).
• **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*φ: managing distributed flow of data on the web
Web Programming
Anatomy of a Web Application

- Mobile code
- Remote execution
- State
- Security

JavaScript

Ajax

id=scACRSm...

anything

HTML
PHP
Java
ASP/.Net
Ruby
Python
Server JS
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

```javascript
<script type="text/javascript" src="prototype.js"></script>
<script language="javascript" type="text/javascript">
function sendRequest(arg)
{
    // Do something with the Ajax response
    function doSomething(result)
    {
        $('#resultDiv').update(result.responseText);
    }

    // send the Ajax request
    new Ajax.Request('helloService.php',
    { method: 'post',
      parameters: arg,
      onComplete: doSomething});
}
</script>
```
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
Looking for foundations of web programming

• A language to carry out local computations
  ✓ A λ-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
\[ \tau ::= \ldots \mid \tau \rightarrow \tau' \]

Expressions
\[ e ::= \ldots \mid w/u \mid \text{publish } x:\tau.\ e \mid \text{call } e_1 \text{ with } e_2 \]

- 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 = 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)
```

Calculates:
- \(\text{fact}(4)\) = 24
- \(\text{fact}(6)\) = 720
- \(24 + 720 = 744\)
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**

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

**Client**

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

**Calculations**

- **fact(4)**
- **fact(6)**
- **(24+720)**
- **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

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

Client

```javascript
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
```

Client

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

Customer

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

A new service has been published at [www.server.com/inst/](http://www.server.com/inst/)
A new service has been published at [www.client.com/search/](http://www.client.com/search/)
Demo

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

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

Documentation

Id: 70300

Type: srv[string][string]

URL: http://localhost/~tsans/qwesst/server/70300/
QWeSST

Formal Semantics
Typing

\[ \Sigma; \Gamma |-_w e : \tau \]

(e has type \( \tau \) at \( w \) in \( \Sigma \) and \( \Gamma \))

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

Remote Procedure Call

\[
\frac{\tau \rightarrow \tau' \text{ mobile}}{\Sigma, w'/u: \tau \rightarrow \tau'; \quad \Gamma |-_w w'/u : \tau \rightarrow \tau'}
\]

\[
\tau \rightarrow \tau' \text{ mobile} \quad \Sigma; \quad \Gamma, x : \tau |-_w e : \tau' \quad \Sigma; \quad \Gamma |-_w e_1 : \tau \rightarrow \tau' \quad \Sigma; \quad \Gamma |-_w e_2 : \tau
\]

\[
\Sigma; \quad \Gamma |-_w \text{publish} \quad x : \tau. \; e : \tau \rightarrow \tau' \quad \Sigma; \quad \Gamma |-_w \text{call} \; e_1 \text{ with } e_2 : \tau'
\]

Mobile Code

\[
\frac{\Sigma; \quad \Gamma |-_w e : \tau}{\Sigma; \quad \Gamma |-_w \text{hold} \; e : \text{ susp}[	au]}
\]

\[
\frac{\Sigma; \quad \Gamma |-_w e : \text{ susp}[	au]}{\Sigma; \quad \Gamma |-_w \text{resume} \; e : \tau}
\]
Evaluation

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

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

Remote Procedure Call

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

\[ \nu_2 \text{ val} \]

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

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

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

\[ \nu \text{ val} \]

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

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 \( \Sigma; . \vdash_w e : \tau \) and \( \Sigma \vdash \Delta \) and \( \Delta; e \Rightarrow_w \Delta'; e' \), then \( \Sigma'; . \vdash_w e' : \tau \) and \( \Sigma' \vdash \Delta' \)

⇒ Progress

If \( \Sigma; . \vdash_w e : \tau \) and \( \Sigma \vdash \Delta \), then

• either \( e \text{ val} \)
• 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

\[
\frac{\Delta; \cdot \Rightarrow \Delta; \cdot}{\Delta; e \Rightarrow_w (\Delta, \Delta'); e' \Rightarrow (\Delta, \Delta''); \Delta; E \Rightarrow (\Delta, \Delta'''); E'}
\]

- Serialized typing semantics

\[
\frac{\Sigma; \cdot \vdash_w e : \tau \quad \Sigma \vdash E : T}{\Sigma \vdash (e @ w, E) : \tau, T}
\]

- 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

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

- “Branching” stack machine with explicit return addresses
  - \((e)^d\) – evaluate \(e\) for \(d\)
  - \((v)^d\) – return \(v\) to \(d\)
  - (call \(d_1\) with \(d_2\))^d – wait for results

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

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

\begin{align*}
(cal e_1 \text{ with } e_2)^d & \Rightarrow_w (call d' \text{ with } d'')^d, (e_1)^{d''}, (e_2)^{d''} \\
\end{align*}

\begin{align*}
\frac{w'/u = x: \tau. e \in \Delta}{(call d' \text{ with } d'')^d, (w'/u)^{d'}, (v)^{d''} \Rightarrow_w (\text{expect } d''' \text{ from } w')^d \\
& \Rightarrow_w ([v/x]e)^{d'''} \\
& v' \text{ \texttt{val}} \\
& (\text{expect } d''' \text{ from } w')^d \Rightarrow_w (v')^d \\
& (v')^{d'''} \Rightarrow_w .
\end{align*}
Orchestration

- **Evaluation**

  \[
  \frac{\{ \Delta; e_i \Rightarrow^w_i (\Delta, \Delta_i); e_i' \}}{\Delta;\{e_i @ w_i\}, E \Rightarrow (\Delta, \{\Delta_i\}); \{e_i' @ w_i\}, E}
  \]

  \((i \in I)\)

- **Typing**

  \[
  \frac{\{ \Sigma; d_i : \tau_i \mid^w_i e_i \} \quad \Sigma \mid \Delta}{\Sigma; \{d_i : \tau_i\} \mid \Delta; \{e_i @ w_i\}}
  \]

  \((i \in I)\)
## Substructural meta-theory

<table>
<thead>
<tr>
<th>Local</th>
<th>Type Preservation</th>
<th>Progress</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>If $\Sigma; d:\tau \mid_w e$ and $\Sigma \mid \Delta$ and $\Delta; e \Rightarrow_w \Delta'; e'$, then $\Sigma'; d:\tau \mid_w e'$ and $\Sigma' \mid \Delta'$</td>
<td>If $\Sigma; d:\tau \mid_w e$ and $\Sigma \mid \Delta$, then • either $e$ val • or $\Delta; e \Rightarrow_w \Delta'; e'$</td>
</tr>
<tr>
<td>Global</td>
<td>If $\Sigma; \Lambda \mid \Delta; E$ and $\Delta; E \Rightarrow \Delta'; E'$, then $\Sigma'; \Lambda \mid \Delta'; e'$</td>
<td>If $\Sigma; \Lambda \mid \Delta; E$, then • either $E$ final • or $\Delta; E \Rightarrow \Delta'; E'$</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?
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
  \( \Rightarrow \tau ::= \ldots \mid \tau[\mu] \rightarrow \tau' \mid \tau[\mu] \rightsquigarrow^w \tau' \)

✓ Type annotations are inferred

- Policies in call
  \( \Rightarrow \text{call } e_1 \text{ with } e_2[\rho] \)
Incorporating paths and policies into Qwesst

- Flow inference and control in type checking

\[
\frac{\Sigma; \Gamma \vdash_w e_1 : \tau[\mu] \xrightarrow{w'} \tau'}{\Sigma; (\Gamma \parallel (\Gamma' \circ (w'; \mu))) \vdash_w \text{call } e_1 \text{ with } e_2[\rho] : \tau'}
\]

- 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?